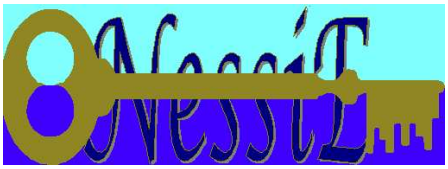

Project Number	IST-1999-12324
Project Title	NESSIE
Deliverable Type	**Report**
Security Class	Public
Deliverable Number	D20
Title of Deliverable	**NESSIE security report**
Document Reference	NES/DOC/ENS/WP5/D20/2
Contractual Date of Delivery	Y3 M9
Actual Date of Delivery	Y3 M11
Revised Version	Y4 M2
Editors	ENS

Abstract

A first security evaluation was published under deliverable number D13 and has served as a basis of a selection of the primitives that have been studied more in detail. This report summaries the new results together with a comprehensive overview of the security evaluation made by the NESSIE consortium.

Keywords NESSIE, Security evaluation.

Version 2.0

February 19, 2003

B. Preneel[1], A. Biryukov[1], E. Oswald[1],
B. Van Rompay[1],
L. Granboulan[2], E. Dottax[2],
S. Murphy[3], A. Dent[3], J. White[3],
M. Dichtl[4], S. Pyka[4], M. Schafheutle[4], P. Serf[4],
E. Biham[5], E. Barkan[5], O. Dunkelman[5],
J.-J. Quisquater[6], M. Ciet[6], F. Sica[6],
L. Knudsen[7,8], M. Parker[7], H. Raddum[7]

NESSIE security report[†]

February 19, 2003
Version 2.0

[†] The work described in this report has been supported by the Commission of the European Communities through the IST program under contract IST-1999-12324. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

[1] Katholieke Universiteit Leuven, Dept. Elektrotechniek-ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
[2] École Normale Supérieure, Département d'Informatique, 45 rue d'Ulm, Paris 75230 Cedex 05, France
[3] Royal Holloway, Information Security Group, Egham, Surrey TW20 0EX, UK
[4] Siemens AG, Otto-Hahn-Ring 6, München 81732, Germany
[5] Technion, Computer Science Dept., Haifa 32000, Israel
[6] Université catholique de Louvain, UCL Crypto Group, Laboratoire de Microélectronique (DICE), Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
[7] Universitetet i Bergen, Dept. of Informatics, PO Box 7800 Thormoehlensgt. 55, Bergen 5020, Norway
[8] Tech. U. of Denmark, Dept.of Mathematics, Building 303, DK-2800 Lyngby, Denmark

Executive Summary

NESSIE security report (NESSIE Deliverable D20)

The NESSIE project is a three year project (2000-2003) that is funded by the European Union's *Fifth Framework Programme*. The main objective of the NESSIE project is to put forward a portfolio of strong cryptographic primitives of various types. An open call in March 2000 led to the submission of forty cryptographic primitives to the NESSIE project. The NESSIE project is evaluating (with some external assistance) these submitted primitives from both a security and performance perspective. This document gives the collective view of the NESSIE partners about the submissions from a security perspective.

The NESSIE evaluation process is an open process. Thus as part of the evaluation process, the NESSIE project welcomes comments about the submitted primitives and the evaluation process, including this report. To facilitate the open evaluation process, there are to be four NESSIE workshops. The first workshop was dedicated to the presentation of the submitted primitives and the second workshop was dedicated to early results concerning the primitives. The third workshop was dedicated to new results and also to the discussion of version 1.0 of this report. The fourth NESSIE workshop takes place at the end of the project and disseminates the results.

This document forms deliverable D20 of the NESSIE project. Version 1.0 was published to be available for comments before the Third NESSIE Workshop and version 2.0 is the final security report.

Table of Contents

1. Introduction

1.1 NESSIE project

The NESSIE project is a three year project (2000-2002) that is funded by the European Union's *Fifth Framework Programme*. The main objective of the NESSIE project is to put forward a portfolio of strong cryptographic primitives of various types. Further details about the NESSIE project can be found at the NESSIE website `http://www.cryptonessie.org/`.

The start of the NESSIE project was an open call [396] for the submission of cryptographic primitives as well as for evaluation methodologies for these primitives. This call includes a request for the submission of block ciphers (as for the AES call), but also of other cryptographic primitives including hash functions, stream ciphers, and digital signature algorithms. The call also asked for evaluation methodologies for these primitives. The scope of the call was defined in conjunction with the project industry board, and was published in March 2000. This call resulted in forty submissions. The NESSIE project aims to assess these submissions with the goal of producing a portfolio of cryptographic primitives for use in different environments. The NESSIE project proposes to disseminate the project results widely and to build consensus based on these results by using the appropriate bodies: a project industry board, NESSIE workshops, the 5th Framework programme, and various standardisation bodies.

The NESSIE project has been divided into two phases. In the first phase of the security evaluation, these submissions were analysed by the NESSIE partners. The NESSIE partners have also received external comments for some submissions. The outcome of the first phase was a preliminary assessment of all submissions: a security evaluation [399] and a performance evaluation [398]. This was used to decide which of the submissions are to be considered in the second phase [397]. In the second phase, the remaining submissions were subject to more detailed scrutiny to produce the portfolio. This report summarises all the security evaluation of the submissions and is the conclusion of the second phase of security evaluation. The NESSIE project also compiles a performance evaluation of the submissions. This security report together with the performance report form the basis of the decision as to which primitives should be part of the NESSIE portfolio.

The NESSIE evaluation process is an open process. Thus as part of the evaluation process, the NESSIE project welcomes comments about the submitted

primitives and the evaluation process, including this report. To facilitate the open evaluation process, there are to be four NESSIE workshops. The first NESSIE workshop took place on 13-14 November 2000 at Katholieke Universiteit Leuven (Belgium) in which the submitted primitives were presented. The second NESSIE workshop took place on 12-13 September 2001 at Royal Holloway, University of London (UK) in which early results concerning the primitives were presented. The third NESSIE workshop took place on 6-7 November 2002 at Siemens AG, Munich (Germany) in which new results concerning the primitives were presented. The fourth NESSIE workshop takes place on 26 February 2003 in Lund (Sweden) and disseminates the results of the project.

1.2 Security evaluation methodology

The NESSIE project has attempted to define a high–level methodology to compare in a fair and acceptable way the submitted primitives. This methodology may evolve according to technical advances, remarks of the NESSIE members, Industry Board or cryptographic experts, and with problems encountered. However it should be noted that it is impossible to produce a definitive security methodology. Cryptographic primitives with completely inadequate security can often be identified. However, for other cryptographic primitives, the situation is nothing like as clear-cut. There is neither an automatic method of assessing the security of such a primitive nor a general consensus on the relative importance of different security criteria. The few previous initiatives that have undertaken a similar task to the NESSIE project, such as AES, have been more limited in scope and have reached a subjective judgement by experts on the security of such primitives. We first give the evaluation criteria specified in the NESSIE call [396] and then a list of important issues that NESSIE has considered in making its security evaluations of a submitted primitive. This list is extensive but not complete.

1.2.1 Evaluation criteria in NESSIE call

1. An attack should be at least as difficult as the generic attacks against the type of primitive (exhaustive search, birthday attack etc.).
2. Primitives will be evaluated against the security claims of the submitter. An attack requiring lower computing resources than claimed would usually disqualify the submission.
3. Primitives will be evaluated within the stated environment. Thus, consideration of vulnerability to side channel attacks (e.g. timing attacks, power analysis) may be appropriate.

1.2.2 Methodological issues

Resistance to cryptanalysis. Clearly, any submission should be resistant at the relevant security level to cryptanalytic attacks. Indeed, in the NESSIE call

for submissions [396], it is stressed that failure to be resistant to such an attack would usually disqualify a submission. However, when assessing the relevance of a cryptanalytic attack, other factors such as the volume and type of data required to mount the attack will be considered.

Design philosophy and transparency. An important consideration when assessing the security of a cryptographic primitive is the design philosophy and transparency of the design of that primitive. It is easier to have confidence in the assessment of the security of a primitive if the design is clear and straightforward, and is based on well-understood mathematical and cryptographic principles. This is particularly relevant when making relative comparisons between primitives (see below).

Strength of modified primitives. One common technique used to assess the strength of a primitive is to assess a modified primitive, for example by changing or removing a component or reducing the number of rounds. Conclusions about the original primitive based on an assessment of the modified primitives have to be carefully considered as the inference may or may not be straightforward.

Relative security. When assessing primitives designed to operate to the same security level in similar environments, it is natural to wish to compare their security. However, care has to be taken when making such comparisons. One measure that has been suggested for primitives based on an iterative algorithm is the *security margin*, which measures the gap between the maximum number of broken rounds and the total number of rounds, but there is no general consensus about its definition or use. Furthermore, whilst the NESSIE project tries to ensure that each submitted primitive receives equivalent cryptanalysis, it is the case that some designs are easier to analyse than others (as discussed above). However, it is felt that there should be some security margin to protect against cryptanalytic advances.

Cryptographic environment. In certain cryptographic environments, a cryptographic primitive may have been designed to possess intrinsic security advantages or disadvantages. An example would be a primitive that is resistant to power or timing attacks when implemented on a smart card. Such properties would be considered when assessing the security of a primitive.

Statistical testing. The NESSIE project is carrying out statistical testing of submitted primitives (where relevant). The purpose of this statistical testing is to highlight anomalies in the operation of the primitive that may indicate cryptographic weakness and require further investigation.

1.3 Structure of the Report

This report is split into distinct chapters for distinct categories of primitives. For a reader with some knowledge in cryptology, each chapter is intended to be self-contained. However, this report is not a course on cryptology; the reader should refer to textbooks [498, 366, 222, 226]. Each chapter has the following sections:

1. **Introduction.**
   The introduction defines what is the category of primitives that is considered.
2. **Security requirements.**
   The security model and common attacks are explained. The assessment process by NESSIE is described.
3. **Overview of the common designs.**
   Common designs are described, and how the primitives submitted to NESSIE fit in these. Current standards are reviewed.
4. **Analysis of all primitives submitted to NESSIE.**
   The analysis of each primitive submitted to NESSIE is summarised. Primitives that were not selected for Phase II have a shorter review than the ones that were studied during the whole NESSIE process.
5. **Conclusion.**
   Recommendations are made for a choice of primitives that should be part of NESSIE portfolio.

The categories of primitives considered in this report are slightly different from the categories defined in the call. This is more consistent with the list of what has been submitted to NESSIE and with the way these primitives were analysed. They are:

Ch. 2. *Block ciphers*
Ch. 3. *Stream ciphers and pseudo-random numbers generators*
Ch. 4. *Hash functions*
Ch. 5. *Message authentication codes*
Ch. 6. *Asymmetric encryption*
Ch. 7. *Digital signature schemes*
Ch. 8. *Digital identification schemes*

An appendix on side-channel attacks and the bibliography conclude this report.

1.4 The submissions received by NESSIE

– **Block ciphers**
  – **Anubis.** [29]
  – **Camellia.** [19]
  – **CS-cipher.** [198]
  – **Grand Cru.** [100]
  – **Hierocrypt.** [412] – Hierocrypt-L1 and Hierocrypt-3
  – **IDEA.** [325]
  – **Khazad.** [30]
  – **MISTY1.** [355]
  – **Nimbus.** [344]
  – **Noekeon.** [149]
  – **Nush.** [329]
  – **Q.** [360]

  - **RC6.** [273]
  - **Safer++.** [353]
  - **SC2000.** [475]
  - **SHACAL.** [237] – SHACAL-1 and SHACAL-2.

- **Stream ciphers and pseudo-random numbers generators**
  - **BMGL.** [241]
  - **SNOW.** [180]
  - **SOBER.** [245] – SOBER-t16 and SOBER-t32
  - **LEVIATHAN.** [361]
  - **LILI-128.** [154]

- **Hash functions**
  - **Whirlpool.** [31]

- **Message authentication codes**
  - **UMAC.** [317]
  - **Two-Track-MAC.** [507]

- **Asymmetric encryption**
  - **ACE-KEM.** [469] – Upgrade of ACE-Encrypt
  - **EPOC.** [203] – EPOC-1, EPOC-2 and EPOC-3
  - **ECIES.** [269]
  - **PSEC.** [202] – PSEC-1, PSEC-2, PSEC-3 and PSEC-KEM
  - **RSA-OAEP.** [274] – Revised to RSA-KEM [489]

- **Digital signature schemes**
  - **ACE Sign.** [469]
  - **ECDSA.** [268]
  - **ESIGN.** [207]
  - **FLASH family.** [428] – FLASH, SFLASH and SFLASHv2
  - **QUARTZ.** [136]
  - **RSA-PSS.** [275]

- **Digital identification schemes**
  - **GPS.** [437]

- **Evaluation methodologies**
  - **General Next Bit Predictor** [249]
    This evaluation methodology appeared to be useless [167], it will not be further mentioned in this report.

1.5 Some mathematical notations

$\mathbb{Z}$	the set of integers
$\mathbb{Z}/n\mathbb{Z}$	the set of integers modulo n
$(\mathbb{Z}/n\mathbb{Z})^*$	the non-zero elements of $\mathbb{Z}/n\mathbb{Z}$
$(\mathbb{Z}/n\mathbb{Z})^\times$	the invertible elements of $\mathbb{Z}/n\mathbb{Z}$, which form a multiplicative group with $\phi(n)$ elements
QR_n	the squares in $(\mathbb{Z}/n\mathbb{Z})^\times$ (quadratic residues)
$\mathbb{F}_q$	the finite field with q elements
$(\mathbb{F}_p)^n$	the n-dimensional vector space on $\mathbb{F}_p$
$\langle G \rangle$	the cyclic group generated by G
1^λ	a bitstring of length λ of all 1, used as a security parameter

NB: if p is prime then $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ and $(\mathbb{Z}/p\mathbb{Z})^* = (\mathbb{Z}/p\mathbb{Z})^\times$.

1.6 Acknowledgements

This document is the result of a collective work from the NESSIE team. Chapter coordinators are editors and not sole authors of the chapter.

Many other people contributed to this document. We thank Antoon Bosselaers, Dario Catalano, Julien Cathalo, Christophe De Cannière, François Koeune, Joe Lano, Jorge Nakahara, Jr, Phong Q. Nguyen, Gilles Piret, David Pointcheval, Rachel Shipsey, Christine Swart,

2. Block ciphers

2.1 Introduction

Block cipher encryption provides confidentiality by transforming a plaintext message into a secure ciphertext message, where the precise function implemented by the block cipher is determined by a secret key. This secret key, or series of keys, is only known by legitimate users of the block cipher. Whereas a stream cipher contains a memory, embodied in its current state, a block cipher is memoryless outside its current block and therefore has no current state. A mode of operation of a block cipher partitions a plaintext message into a series of blocks which are then encrypted one block at a time, although a block cipher can be used as a component in a stream cipher, and also for pseudorandom number generators, MACs, hash functions, and signature schemes. Block cipher encryption is the most well-known form of symmetric-key encryption — the word *symmetric* implies that both transmitter and receiver of the ciphertext have knowledge of the secret key. Block ciphers have been around in one form or another for a very long time, for instance the substitution cipher, and the transposition cipher. However, many of the ideas that permeate modern block cipher design were inspired by the work of Shannon [473] around 1949. He first elucidated the concepts of confusion and diffusion that are still primary design criteria for any state-of-the-art block cipher. The rate at which cryptographic and information theory have developed has accelerated over the last thirty years or so, with the result that many *rules* for block cipher design are now well-accepted amongst the majority of cryptographers [297]. However, it remains the case that no practical block cipher is provably secure and, consequently, new design criteria are still being discovered, these often as a response to emerging novel attacks on block ciphers. Typically, a block cipher design is proposed according to well-accepted and well-founded *rules*, and this inevitably forces the cryptanalyst to attempt to attack the cipher in some unforeseen way. These unforeseen attacks, if successful, lead in turn to the extending of the canon of design criteria — and so the discipline progresses. It should therefore come as no surprise that, during the three-year lifespan of NESSIE, new designs and new attack methods have been proposed, and new questions raised. To some extent, some of the block cipher designs submitted to NESSIE have been influenced by the knowledge gained from the search for the new Advanced Encryption Standard (AES), immediately prior to NESSIE.

[0] Coordinator for this chapter: UiB — Matthew Parker

In particular the inclusion of high-diffusion ciphers in both AES and NESSIE has encouraged the development of algebraic and non-statistical attacks on block ciphers. It should be emphasised that most of the block ciphers submitted to the NESSIE project appear to be secure, acceptably efficient, and practical for use in real systems. The criteria for selection for the NESSIE portfolio therefore rests, to some extent, on secondary considerations, in particular on the identification of potential weaknesses and expected performance on different platforms. In this chapter, attacks on the various block ciphers submitted will be described, and potential security weaknesses of the candidates identified.

It should also be noted that there are four usual modes of operation for a block cipher. The most common mode is called Electronic Codebook Mode (ECB) which takes disjoint plaintexts and outputs disjoint ciphertexts. There is also Cipher Block Chaining (CBC) mode, where the encryption of a block depends on the encryptions of previous blocks. Then there is Cipher Feedback (CFB) mode which is the first of the two stream cipher modes, where one m-bit character at a time is encrypted. Finally there is Output Feedback (OFB) mode where, in contrast to CFB mode, the stream bits are not dependent on the previous plaintexts, i.e. only the stream bits are fed back, not the ciphertext as in CFB mode. NIST has now published a new standard for block cipher modes [385], and Counter Mode (CTR) has been added.

2.1.1 Block Cipher — A Formal Definition

Before discussing security aspects we first give a more precise definition of a block cipher [226]. A block cipher is a function $E : \{0,1\}^K \times \{0,1\}^N \to \{0,1\}^N$ that takes two inputs, a K-bit key k and an N-bit plaintext P, to return an N-bit ciphertext $C = E(k,P)$. For any block cipher, and any key k, the function E_k is a *permutation* on $\{0,1\}^N$. This means that it is a *bijection*, i.e. a *one-to-one* function of $\{0,1\}^N$ to $\{0,1\}^N$. Accordingly, it has an inverse, E_k^{-1}. Both the cipher and its inverse E^{-1} should be easily computable, meaning that given k, P we can compute $E(k,P)$, and given k, C we can compute $E^{-1}(k,C)$.

2.2 Security requirements

Block cipher encryption is a method to transform a plaintext message of blocklength N bits by encrypting it to a ciphertext message of blocklength N bits, where the encryption operation is determined by a secret key string of length K bits, where the key is often chosen uniformly at random. The inverse operation, block cipher decryption, takes the N-bit ciphertext and decrypts it back to the N-bit plaintext using the same secret key string of length K bits. The aim is to make it practically impossible to retrieve the plaintext from the ciphertext without knowledge of the K-bit secret key. Decryption is only possible if the encryption function is invertible (i.e. if it is a bijection) and this restricts the choice of possible N-bit block ciphers to one of $(2^N)!$ block ciphers. However, parameterisation by a secret key of length K bits further restricts the set of block ciphers

realised by a particular design to a maximum of 2^K block ciphers (which, for any reasonable N and K, is an infinitesimally small fraction of the complete space of $(2^N)!$ block ciphers). The problem of block cipher design is to determine which set of 2^K block ciphers to choose such that, for an unknown fixed key, it is virtually impossible to say anything about the ciphertext resulting from a known or chosen plaintext, or to say anything about the fixed key, given prior knowledge of a few plaintext/ciphertext pairs. Note that, if the plaintext contains exploitable redundancy, then one may be able to attack the cipher using ciphertext only. Any effective block cipher scheme must be realised efficiently in time and space, with as little implementation cost as possible. The practical trade-off is therefore to design a block cipher which is both sufficiently secure, and satisfactorily efficient in terms of hardware/software space and time resources. It should be emphasised that the complete design of the block cipher, along with all ciphertexts, is considered public knowledge, with only the secret key remaining unknown to attackers of the system. Clearly, knowledge of the secret key implies knowledge of the plaintexts that were encrypted using that secret key. A block cipher with a secret key is considered perfect if, for all plaintexts, P, and ciphertexts, C, it holds that $\Pr(P) = \Pr(P|C)$ [473]. If, for a fixed K-bit key, an N-bit block cipher is used to encrypt $\lfloor \frac{K}{N} \rfloor$ plaintexts, then the cipher can always be chosen to be the one-time pad so that, in this special case, the encryption is provably secure and the block cipher perfect — a one-time pad is a symmetric key block cipher where K key bits are used, only once, to encrypt K plaintext bits, where the K corresponding ciphertext bits are the XOR of the plaintext bits with the key bits. In such a situation the ciphertext and plaintext are statistically independent. However, in most situations the one-time pad is impractical as far too many secret keys must be used. Therefore it is highly desirable to securely encrypt T plaintexts using the same, fixed K-bit secret key, where $T \gg \lfloor \frac{K}{N} \rfloor$. Most modern block ciphers seek to maximise T, whilst still achieving an acceptable security, via a combination of confusion, which makes the relationship between key and ciphertext as complicated as possible, and diffusion which seeks to eliminate any redundancy in the plaintext. Diffusion also makes it difficult for any attacker to partially approximate the cipher.

Theoretically the ideal block cipher, from a security viewpoint, would involve one very large, well-chosen N-bit Substitution Box (S-Box), keyed by K key bits and, ideally, it would be impossible to decompose this S-box into smaller sub-units. However this immediately implies a huge implementation complexity, so any practical block cipher will, instead, combine relatively small sub-units to *confuse* (e.g. S-boxes) and *diffuse* (e.g. linear transformation layers) the plaintext. Moreover, these sub-units will be applied iteratively as keyed rounds, parameterised by sub-keys which are derived from the master K-bit key. This decomposition into practical sub-units constitutes a trade-off between security and acceptable complexity. All the block ciphers submitted to NESSIE are iterated over multiple rounds, and all of them utilise a key-schedule to derive round keys from a master key. It is this decomposition into sub-processes that provides the cryptanalyst with ammunition for an attack. In spite of the above compromises, it is an accepted design principle that encryption using a block cipher, selected

via a randomly-chosen key, should look like encryption by a randomly-chosen invertible function over N bits.

2.2.1 Security model

It is the nature of scientific discovery that the initial models are, to a large extent, heuristic — intended security against well-known attacks. These heuristics later give way to formal proof of security versus resources needed. What is certain is that, with respect to practical block cipher security, very little can (yet) be proved to any high-degree of accuracy — hence the existence of NESSIE. However, there exist a number of accepted security models which can tell us something about the block cipher under consideration:

- **Unconditional Security (Perfect Secrecy).** Shannon assumed that an adversary has unlimited computational resources. In his model, secure encryption only exists if the size of the key is as large as the number of secret bits to be exchanged remotely using the encryption system. Perfect secrecy is possible only if no more than $\lfloor \frac{K}{N} \rfloor$ plaintexts are enciphered using a fixed key (e.g. the one-time pad), so unconditional security is not a useful model for practical block ciphers.
- **Security Against Polynomial Attack.** In contrast to Uncondition Security, modern cryptography assumes the adversary's computation is resource-bounded. Specifically, it is assumed that the adversary is a probabilistic algorithm which runs in polynomial time, and security is claimed with respect to the feasibility of breaking the cryptosystem. This model arises out of complexity theory considerations where adversaries are assumed to possess only polynomial computational resources — polynomial in the size of the input to the cipher in bits. The model typically conducts worst-case and asymptotic analyses to determine whether polynomial attacks on a cipher exist. Even if they do exist, it is not guaranteed that such attacks are practical. This security model tends to provide an understanding as to the type (class) of problem embodied by a block cipher, without providing exact figures.
- **"Provable" Security.** Typically this can mean one of two things. Firstly, if it can be shown that breaking a block cipher is as difficult as solving some well-known hard problem (e.g. discrete log or factoring) then the cipher is considered provably secure. This is, of course, misleading as the hard problem on which it is based is usually not provably hard. This relates to a very fundamental open question in computer science as to whether these hard problems are in P or in NP. In fact, provable security requires a proof that P $\neq$ NP, and the existence of one-way functions which are hard *on the average*, but which can be solved quickly given some extra information [224] (pages 27-28). Note that these are *asymptotic* complexity measures — one is assessing the level of complexity as the input size, in bits, asymptotes to infinity. The strategy of mapping cryptosystems to hard problems is very useful for practical analysis of the cipher, although this model is more often applied to public-key cryptosystems. Secondly, a block cipher may be shown to be provably secure against a known

sub-class of attacks. One example of this is the provable security against linear and differential cryptanalysis used, for example, by the designers of MISTY1. It should be emphasised however that this obviously does not mean that the cipher is secure against all attacks.
- **Practical Security.** In this model a block cipher is considered computationally secure if the best-known attack requires too much resource by an acceptable margin. This is a very practical model as one can test the cipher with different known attacks, probing for weakness, and then give an assessment of the cipher's strength against such attacks in terms of time/space resources needed. This model tends to provide the most answers, and most of the analysis in NESSIE was of this type. However, it says nothing about the security level with respect to yet unknown attacks.
- **Historical Security.** It is quite useful to assess the security level of a block cipher according to how much cryptanalytic attention the cipher has attracted over the years. For example, both Cipher A and Cipher B could be considered excellent cipher designs. But Cipher A may be ten years older and has therefore been under scrutiny for many more years than Cipher B without any serious security flaws found in it. This inevitably inspires a certain confidence in the older cipher and suggests that the time-scale over which projects such as NESSIE and AES operate is only sufficient to draw preliminary conclusions as to the security of a completely new cipher. However, it should also be noted that the effort spent on breaking a cipher cannot always be measured reliably from the time passed.

2.2.2 The Block Cipher as a Pseudorandom Permutation

It is natural to consider a block cipher as a set of permutations. In this context we can consider a *distinguisher* which differentiates between a randomly-selected pseudorandom permutation and a permutation which is randomly selected from the set of permutations generated by the block cipher. This section investigates this approach in more detail, considering the asymptotic limit as the size of the input and output to and from the block cipher approaches infinity. (We here summarise and paraphrase some of the definitions given by Goldwasser and Bellare [226] and others [222].)

Definition: Let U_N denote a random variable uniformly distributed over $\{0,1\}^N$. A *pseudorandom function* is one of an infinite set of functions with increasing input sizes, $\{f(U_N, K)\}, N = 1, \ldots, \infty$, with the property that the input-output behaviour of a random instance of the set is *computationally indistinguishable* from that of a random function. *Pseudorandom permutations* can be described similarly.

A block cipher, $E(k, P)$, can be considered as a set of 2^K permutations on the message space, each instance, $E_k(P)$, of the set being a distinct permutation and obtained by fixing the key k. In this setting one can model the following attacks on the cipher. Let g be a function drawn at random from the set of all N-bit permutations, as $N \to \infty$. The adversary gets an oracle for g, and can also

get an oracle for g^{-1}, these relating to chosen plaintext and chosen ciphertext attacks, respectively.

Definition: A *one-way function*, $f : \{0,1\}^* \rightarrow \{0,1\}^*$, is a function which is *easy* to compute but *hard* to invert. By *easy* we mean that f can be computed by a deterministic polynomial time algorithm, and by *hard* we mean that any probabilistic polynomial time algorithm attempting to invert f will succeed with *negligible* average probability (where the average probability is taken over the elements in the domain of the function f). More formally, for every probabilistic polynomial time algorithm, A', every polynomial $p(\cdot)$, and all sufficiently large N's,

$$\Pr(A'(f(U_N), 1^N) \in f^{-1}f(U_N)) < \frac{1}{p(N)}$$

where U_N denotes a random variable uniformly distributed over $\{0,1\}^N$, and 1^N is some auxiliary input to the algorithm A'.

The block cipher can be considered secure if it can be shown to be equivalent, asymptotically, to a set of pseudorandom functions or pseudorandom permutations. In other words, we cannot distinguish the output ciphertext bits from random output. This allows us to relate block ciphers to one-way functions as follows. Given an N-bit plaintext message, P, and a K-bit key, k, the block cipher function, E, produces an N-bit ciphertext output message, C, where $C = E(k, P)$. Then, for P fixed to p, we can define f such that $f(k) = E(k, P = p)$. Luby and Rackoff show how to build a pseudorandom permutation from a pseudorandom function using a few rounds of a Feistel construction [338], and in [339] they prove that, asymptotically, f is a one-way function on the assumption that E is a set of pseudorandom functions. Therefore, retrieving the key, k, using $k = f^{-1}$, is proven to be hard. However, the notion of a one-way function is weaker than the notion of a secure block cipher. For example, a one-way function may leak half of its input and still be one-way (non-invertible). This demonstrates the need to introduce the idea of '*hard-core*' bits of a function. Given an efficient algorithm to predict the value of a hard-core bit, one can construct an algorithm inverting the one-way function. For a secure block cipher all bits have to be hard-core bits.

Distinguishing Attacks. By viewing a block cipher as a set of permutations we can develop a *distinguisher* which compares and differentiates between the block cipher and the 'ideal' set of random permutations. This is done as follows.

Let g be a function drawn at random from the set, $\mathbf{D}$, of all N-bit permutations. Let g' be a function drawn at random from the set of N-bit permutations, $E(k, P)$. In practice this is achieved by drawing a K-bit key, k, at random from the set of all 2^K K-bit keys. The adversary, A, is a given a series of N-bit permutations, g'', and its job is to determine whether its series of g'' are a series of permutations, g, or a series of permutations, g'. To achieve this the adversary uses an algorithm $A^{g''}$ that takes a function, g'', as input, and returns a bit, d. The *advantage* of the adversary who uses algorithm $A^{g''}$ is given by,

$$\text{advantage}_{E,A} = |\Pr(A^g = 1) - \Pr(A^{g'} = 1)|$$

and this *advantage* measures the ability of algorithm $A^{g''}$ to distinguish between a function g taken at random from $\mathbf{D}$ and a function g' taken at random from the set of N-bit permutations, $E(k, P)$. If we now consider the set of all algorithms $\{A^{g''}\}_{t,q,\mu}$ having time complexity at most t, making at most q oracle queries, such that the sum of the lengths of these queries is at most μ bits, then we can define the pseudorandom permutation advantage (prpadvantage) of E as follows,

$$\text{prpadvantage}_E(t, q, \mu) = \max_{\{A^{g''}\}_{t,q,\mu}} \{\text{advantage}_{E,A}\}$$

where the maximum is over the set $\{A^{g''}\}_{t,q,\mu}$. We can say that a set, $E(k, P)$, which represents a block cipher, is a secure pseudorandom function if $\text{prpadvantage}_E(t, q, \mu)$ is *small* for *practical* values of the resource parameters, t, q, μ.

In this section we have defined the security of a block cipher in terms of a secure pseudorandom function. The scenario is that of a *chosen-plaintext attack* where the attacker is assumed to have control over the input plaintext, P. A similar scenario can be developed for a *chosen-ciphertext attack*, and in [226], the chosen-ciphertext attack is also assumed to give the adversary more power: not only can it query g, but it can directly query g^{-1}. In the above we have assumed that an ideal block cipher is a set of permutations (more generally, functions) with the property that the input-output behaviour of a random instance of the family is, asymptotically, *computationally indistinguishable* from that of a random permutation (function). This is a much stronger assumption than just assuming the block cipher is secure against key recovery. However, distinguishing attacks typically use a *distinguisher* similar to that defined above to recover a subset of key bits. But there could exist better attacks that break the cipher as a pseudorandom permutation (function) without recovering the key, although no such attacks are currently known. Conversely, [226] proves that any function family that is insecure under key-recovery is also insecure as a pseudorandom permutation (function).

2.2.3 Classification of attacks

As stated previously, most of the analysis done by NESSIE falls into the Practical Security model. The success-level of an attack is usually measured according to time, memory, and data complexities needed:

- Time complexity needed for an attack on a block cipher is the number of steps required for the attack algorithm. This often reduces to the number of decryptions. However this unit is not always appropriate, for instance the Gaussian elimination used to solve a system of equations describing the cipher will use very different units of time complexity.
- Memory complexity needed for an attack on a block cipher is typically measured in terms of the amount of storage required for the attack algorithm.
- Data complexity is the number of texts the attacker gets from the encryption oracle.

Typically, Memory is much more expensive than Time, for example an attack that requires 2^{64} Memory is very expensive in comparison to 2^{64} steps of an algorithm. Data is also considered expensive and should be minimised. Various tradeoffs are possible, and the complexity of the attack is usually taken as the max(Time,Data). For more practical attacks it may be useful to trade Data for Time. An attack is considered successful in theory, and the block cipher considered *broken* if the Time complexity required is of order $< 2^K$, where K is the number of key bits used to parameterise the cipher. In this case one says that the block cipher is broken if the K key bits can be guessed in time faster than exhaustive key search, and partially broken if some of the plaintext bits can be discovered in time faster than exhaustive key search. Also, for a fixed key, the complete cipher can be characterised if all 2^N different plaintexts are encrypted. This puts an upper bound on the Data Complexity required to mount an attack to 2^N. A block cipher is considered secure if no attack requires both Time and Data complexity significantly less than 2^K and 2^N, respectively. Very often it is difficult to mount an attack on the complete R-round cipher so many (most) attacks break *reduced-round* versions of the cipher. Therefore another way to assess the security of a cipher is to quote the maximum number of rounds of the cipher that have currently been broken. It is the aim of the block cipher designer to make the cipher look as much like a random bijection as possible. Therefore any process which can, for a fixed key, distinguish the cipher from a random cipher, constitutes an attack on the cipher. Such attacks are called Distinguishing attacks, which encompass many of the most effective attacks used today.

The types of attack that can be performed depend on what resources are available to the adversary. They can be classified as follows:

- Ciphertext-only attacks. The adversary has access to a set of ciphertexts and also knows something about the nature of the plaintext.
- Known plaintext attack. The adversary has access to a set of plaintext-ciphertext pairs.
- Chosen plaintext attack. The adversary is able to choose a series of plaintexts and has access to the resultant ciphertexts.
- Adaptively chosen plaintext attack. The adversary is able to choose a series of plaintexts, where the choice of each new plaintext is influenced by the ciphertexts obtained from the previous plaintexts.
- Chosen ciphertext attacks. Similar to chosen and adaptively chosen plaintext attacks, but with the roles of plaintext and ciphertext reversed.
- Combined chosen plaintext/ciphertext attacks. In this case the adversary is able to choose both plaintexts and ciphertexts. The Boomerang attack is an example of such an attack.

The chosen text attacks are always the most powerful attacks but, of course, are often less realistic in a practical context. Ideally the designer or analyst should try to prove that the cipher is secure against adaptively chosen attacks.

Because of the birthday attack on, for instance, Cipher Block Chaining, and Accumulated Block Chaining modes, it is recommended that a single key is used

to encrypt at most $2^{N/2}$ ciphertexts [297]. This is because one only requires about $2^{N/2}$ ciphertexts to obtain a matching pair of ciphertexts with probability $> \frac{1}{2}$. It should be noted that this restriction is independent of key size.

Attacks typically fall into two main groups, firstly those that are statistical in nature, and secondly those that are largely non-statistical. However we do not distinguish these two types of attacks here as most attack strategies that work with probability one can also be envisaged in a scenario with probability strictly less than one. We now briefly describe the best-known attacks.

2.2.3.1 Exhaustive Key Search

Applicable to all ciphers, the attack only needs a few known plaintext-ciphertext pairs. The adversary tries all keys one by one to check whether the given plaintext encrypts to the given ciphertext. The attack requires only approximately $\lceil \frac{K}{N} \rceil$ pairs to determine the key.

2.2.3.2 Differential Cryptanalysis

This chosen plaintext attack was first applied to DES by Biham and Shamir [69]. It was the first attack which could (theoretically) recover DES keys in time less than exhaustive search. Typically, pairs of chosen plaintexts, (P_0, P_1) with a fixed difference, $\Delta = P_0 - P_1$ are chosen, where the difference operation "$-$" is usually chosen to be the group operation that is used to add the fixed round key. Thus $\Delta = (P_0 + k) - (P_1 + k)$, so the important point is that the difference, Δ, is independent of the key, k, chosen. Moreover, the difference is chosen so that with some acceptably high probability, this difference, Δ, propagates to an output difference Δ' at the output of the round. If one can propagate differences through all rounds with sufficiently high probability, then this allows one to devise a key-invariant approximation to the core (central) rounds of the cipher which can establish a non-random difference relationship between bits specified by the mask, I, near to the input, and bits specified by the mask, O, near to the output of the cipher. This in turn enables the adversary to guess round key bits of the first and last round, and compute the differences at bits specified by I and O according to this guess. If the guess is correct then the differences at pair (I, O) will agree with the guess for the differences at (I, O) with probability p. If p is large enough then choosing enough plaintext-ciphertext pairs will enable the adversary to determine whether the guess was correct. In this way the adversary can determine the round key bits of outer layers of the cipher and work inwards. The processing complexity of a differential attack is approximately $\frac{c}{p}$, where c is a small constant. For many block ciphers the round key is added using XOR. In this case the notion of difference between plaintexts also uses XOR, thereby ensuring key-independence for the approximation. However, other notions of difference can also be useful. One should distinguish between a characteristic and a differential. Whereas a characteristic specifies one particular evolution of differences through the cipher, a differential takes into account all possible paths through the cipher that would yield the same output difference and sums their combined probabilities [326]. Therefore Differential Cryptanalysis using differentials as opposed to just characteristics always achieves higher probabilities. However, the gain is not

always significant. Maximum Average Differential Probability is also sometimes used to assess the goodness of Differential Cryptanalysis. Differential Cryptanalysis typically uses the concept of a Markov Cipher [326] where the characteristic probability is independent of the actual round inputs and is computed over all possible choices of the round key. Moreover, it was shown by Lai *et al.* [326] that, if the round keys are independent, then so are the characteristic probabilities, and this allows these probabilities to be combined relatively simply. However, typically the attacker gets multiple encryptions under the same key, which means the round keys may appear dependent. Fortunately the Markov assumption can be shown to hold approximately for virtually all keys [321].

2.2.3.3 Truncated Differential Cryptanalysis

Instead of trying to propagate a complete difference through every cipher round, one can aim to propagate only part of the difference — hence truncated differentials [294]. The advantage here is that the partial difference propagations may occur with much higher probability than full difference propagations.

2.2.3.4 Impossible Differential Cryptanalysis

Here the aim is to find a differential that occurs with zero probability over a number of rounds of the cipher. Then by key guessing outside the core approximated rounds one can rule out certain key guesses if they allow the forbidden differential to occur. Such attacks have been applied by Knudsen [296], and by Biham *et al.* [57]. One particularly useful fact in [296, 57] is that any Feistel cipher with bijective round functions has an impossible differential after 5 rounds (see Sect. 2.3.1 for a definition of Feistel).

2.2.3.5 Higher Order Differential Cryptanalysis

This is a recursive extension of differential cryptanalysis where one looks to establish probabilistic nested differences across rounds of the cipher [323]. Thus an s-th order differential requires sets of 2^s chosen plaintexts with fixed pairwise difference between members of the set [294, 322]. It follows that an $(s+1)$-th order differential of a function of nonlinear order d is zero. This attack was successfully applied by Nyberg and Knudsen against the cipher of [407] which is *provably secure* against differential attack. The boomerang attack of Wagner [514] can be viewed as a special type of second-order differential attack, and is suited to ciphers where one first-order differential applies to the first half of the cipher, and another first-order differential applies to the second half of the cipher.

2.2.3.6 Linear Cryptanalysis

This known plaintext attack was first applied to FEAL by Matsui and Yamagishi [356] and to DES by Matsui [354]. It is conceptually similar to differential cryptanalysis as the aim is to establish essentially key-invariant approximations. This follows because if the linear relationship $x = y$ holds with probability $\frac{1}{2} + \delta$, then $x = y + k$ holds with probability $\frac{1}{2} \pm \delta$. Thus the bias of the approximation away from $\frac{1}{2}$ is still δ and is unaffected by key change. One approximates sub-units of the cipher by linear approximations which hold with some probability different from $\frac{1}{2}$. If these approximations can be connected up over all core rounds of the

cipher then one can establish linear relationships between bits specified by the mask, I, near the input, and bits specified by the mask, O, near the output of the cipher. This in turn enables the adversary to guess round key bits of the first and last round, and compute bits at I and O according to this guess. If the guess is correct then the pair (I, O) will agree with the guess at (I, O) with probability $\frac{1}{2} \pm b$. If the bias, b, is large enough then choosing enough plaintext-ciphertext pairs will enable the adversary to determine whether the guess was correct. In this way the adversary can determine the round key bits of outer layers of the cipher and work inwards. The complexity of a linear attack is approximately $c \cdot \frac{1}{|b|^2}$ for some constant, c, where b is called the bias. Linear hulls are an extension of the above technique [406] and exploit the fact that there may be more than one linear characteristic (propagation route) through a cipher that begins and ends at the same place. This allows one to sum up the individual biases for each characteristic, although it is possible that the individual characteristics can cancel each other out. The Maximum Average Linear Hull Probability is also sometimes used as a metric to assess the goodness of linear cryptanalysis. A similar development of linear cryptanalysis is to re-use the data one already possesses in different ways, by applying different linear approximations and pooling the information one receives about the key. This technique is referred to as Multiple Linear Approximations [282, 121]. The technique is particularly useful if two or more of the best linear approximations have comparable biases. A further generalisation of linear cryptanalysis looks for nonlinear approximations through parts of the cipher. Although these nonlinear approximations typically exist with higher probability than linear approximations they do not usually preserve key-invariance, so are typically used at either end of the cipher to positively enhance linear biases [310, 303].

2.2.3.7 Differential-Linear Cryptanalysis

Many attack techniques can be combined to form a hybrid attack on a cipher. An example of this is Differential-Linear cryptanalysis. In [327] a differential is established through part of the cipher and used to create a linear approximation with probability 1. This technique is also used in [62, 67] where the differential part is used to create linear approximations which have probability strictly less than 1.

2.2.3.8 Boomerang Attacks

This attack is based on differential techniques and was first described by Wagner [514]. *Boomerang attacks* allow for a more extensive use of structures than is available in conventional differential attacks. More specifically, the boomerang attack is a differential attack that attempts to generate a quartet structure at an intermediate value halfway through the cipher. This quartet structure is illustrated in Fig 2.1.

The aim, as shown in Fig 2.1, is to cover the plaintext pair P, P' with the differential characteristic for E_0, and to cover the plaintext pairs P, Q and P', Q' with the differential characteristic for E_1^{-1}. In this case it can be shown that the plaintext pair Q, Q' is perfectly set up to use the differential characteristic

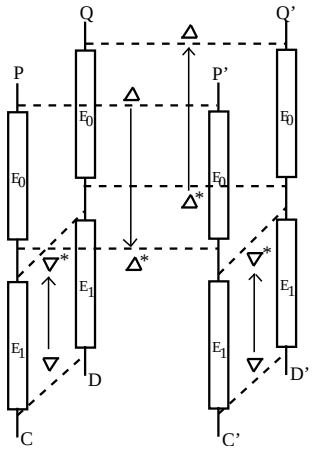

Fig. 2.1: A Typical Boomerang Attack

$\Delta^* \rightarrow \Delta$ for E_0^{-1}. We can summarise this scenario as follows:

$$\begin{aligned}
&E_0(Q) \oplus E_0(Q') \\
&\quad = E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') \\
&\quad = E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(C') \oplus E_1^{-1}(D') \\
&\quad = \Delta^* \oplus \nabla^* \oplus \nabla^* \\
&\quad = \Delta^*
\end{aligned}$$

Thus one will have the same difference in the plaintexts Q, Q' as found in the original plaintexts, P, P', which is why the attack is called the boomerang attack. To set up this quartet one can generate $P' = P \oplus \Delta$, then obtain the encryptions C, C' of P, P' with two chosen-plaintext queries. Then one generates D, D' as $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$. Finally D, D' are decrypted to obtain the plaintexts Q, Q' with two adaptive chosen-ciphertext queries.

2.2.3.9 Mod n Cryptanalysis

This attack by Kelsey *et al.* [289] is a generalisation of a linear attack, and uses the property that some bit groupings (words) within the cipher may be biased modulo n, where n is typically some small integer. It has been shown that ciphers that use only bitwise rotations and additions, mod 2^{32} are particularly vulnerable to such attacks.

2.2.3.10 Weak-Key Classes

A weak-key class refers to any subset of size 2^s of the key space such that, for this class, an attack is known requiring fewer key guesses than exhaustive search, where exhaustive search here means 2^{s-1} key guesses. For instance, for some block ciphers the round key is added using integer addition or even multiplication of some form. And some block ciphers use even more general keyed nonlinear components. In these cases, differential and linear cryptanalysis may benefit from a consideration of a sub-class of the complete space of keyed block ciphers for

this particular design — a weak-key class. Typically this class will be defined by fixing a well-chosen subset of the key bits so that the residual cipher is then open to attack. This is a useful way of assessing whether the cipher is secure for all possible keys. Block ciphers which add the key using XOR are more immune to weak-key class attacks than block ciphers which add some or all of the key nonlinearly [376]. A stronger form of the attack demands that the attacker can identify whether the key used is in the weak-key class. This is known as the key-class membership test, and a practical attack should use a low complexity membership test. For instance, Wagner [514] uses a boomerang attack as a weak-key class identifier.

2.2.3.11 Related-Key Attacks

There are several variants of this attack depending on the privileges of the adversary. Either the adversary obtains encryptions under one fixed key, or he obtains encryptions under several keys where there is either a known or a chosen relation between the keys. The fixed key variant was first used by Knudsen in [293] to establish a chosen plaintext attack, reducing an exhaustive key search by four times. The version using several keys was developed in [51, 295, 288]. It is noted by Biham [51] that the related-key attacks discussed in that paper are completely independent of the number of rounds of the cipher. Slide attacks can be a variant of related-key attacks, and these are discussed later.

2.2.3.12 Interpolation Attack

The interpolation attack was proposed by Jakobsen and Knudsen in [266, 267]. This attack is interesting in that one need not recover the key to break the cipher as the attack seeks to construct a polynomial relationship between plaintext and ciphertext given a set of known plaintext-ciphertext pairs. The attack is often easier if components of the cipher have a straightforward mathematical description or if the polynomial to be approximated is of relatively low degree and with relatively few non-zero coefficients. The number of plaintext-ciphertext pairs required depends directly on the degree of the constructed polynomial — the lower the better. The technique of choice is Lagrange Interpolation. The attack also enables the recovery of round keys once the polynomial is constructed. For instance, although an interpolation attack can predict the ciphertext from the plaintext *without* knowledge of the key, it can also be used to predict the output from the last but one round of the cipher and this, in turn, allows recovery of the last round key. Meet-in-the-middle techniques can be used to reduce the degree of the polynomial to be interpolated [266].

A probabilistic version of the interpolation attack has also been proposed by Jakobsen [265], which views the attack as a problem in coding theory and applies Sudan's algorithm for decoding Reed-Solomon codes [499]. The paper [524] further investigates ways of minimising the degree of the polynomial to be interpolated by changing the irreducible polynomial over which the S-boxes of the block cipher are described.

It is shown by Kurosawa *et al.* [320] how to use Rabin's root finding algorithm in conjunction with the Interpolation Attack so as to find all the possible last round keys that satisfy the interpolated polynomial.

2.2.3.13 Non-Surjective Attack

This attack [450] is applicable to Feistel ciphers where the round function is non-surjective. (See Sect. 2.3.1 for a definition of Feistel). It uses the possibility that the F-function of the Feistel cipher may be non-bijective. This is true for DES, and this attack breaks DES faster than exhaustive search [153, 56].

2.2.3.14 Slide Attack

Slide attacks [80] developed out of related-key attacks [51, 293] and exploit a weakness in ciphers that use identical or periodic round functions. These attacks are, in many cases, independent of the number of rounds of a cipher, and independent of the exact properties of the iterated round function. Let $F_r \circ F_{r-1} \circ \ldots \circ F_1$ denote an r-round iterated cipher, where the F_is are identical or periodically related through the rounds. The adversary looks for pairs of plaintexts P, P^* and their corresponding ciphertexts C, C^* such that $F_1(P) = P^*$ and $F_r(C) = C^*$. This gives an adversary two input-output pairs of one round of the cipher. One can expect, by the birthday paradox, to find such pairs of texts after about $2^{N/2}$ plaintexts. But, for Feistel ciphers, where the round function modifies only half the block, there is also a chosen-plaintext variant which can often cut the complexity down to $2^{N/4}$ plaintexts. The basic slide attack is shown in Fig 2.2.

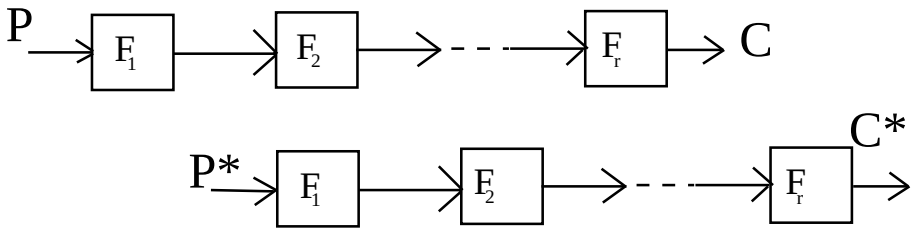

Fig. 2.2: A Typical Slide Attack

For the slide attack to work, F_i should be very weak against known-plaintext attack with two plaintext-ciphertext pairs. Some ciphers include the addition of randomised constants into the key schedule and/or round irregularities, and these protect, to some extent, against a slide attack. For instance, a slide attack by Biryukov and Wagner on MISTY1 is prevented by the inclusion of the nonlinear FL layers after every six rounds of MISTY1 [80].

2.2.3.15 Integral/Multiset Cryptanalysis

There are, in fact, a number of different attack techniques which fall under the umbrella of Integral or Multiset attacks, these being Square attacks [147, 148], Integral attacks [311], Multiset attacks [79], and Saturation attacks [342]. They also appeared in the birthday cryptanalysis of Ladder-DES [53]. Saturation attacks assume only permutations are used, and Multiset attacks cover both Saturation and Integral attacks. In fact, Gilbert-Minier's collision attack is an example of a

Multiset attack [219]. Whereas in differential cryptanalysis one considers differences of pairs of plaintexts, in integral cryptanalysis [311] one considers sums of plaintexts (integrals) to exploit the degree of balance of the output. The Square attack, which was first applied to the Square cipher by Daemen *et al.* [147, 148] and then to Rijndael [150], is an example of integral cryptanalysis. Integrals are particularly suited to the analysis of ciphers with mainly bijective components. Until now, most integrals that have been developed have probability one, and probabilistic integrals have not been examined in depth. Typically, integral attacks set up a path through the cipher where at any position in the path the collection of texts produces either a set of words which are all different (A), or all the same (C), or such that the sum of the words is S (where $S = 0$ is common), or indeterminate (?). One can then follow interacting paths of A, C, S, and ? through the cipher and predict the form of the set of words after as many rounds as possible. It is interesting to note that [311] successfully combines integral attacks with interpolation attacks, where one half of the cipher is covered by an integral, and the other half is approximated by a low-degree polynomial. In [79], Multiset attacks were used by Biryukov and Shamir to structurally cryptanalyse a block cipher. In this context, one does not exploit the weaknesses of a particular cipher, (such as bad differentials) but, instead, studies the security of cryptosystems described by generic block diagrams. Such attacks are therefore applicable to a large class of cryptosystems, and multiset strategies are particulary suited to such attacks.

2.2.3.16 χ^2 Attack

The χ^2 attack quantifies the statistical significance of certain input-output dependencies for a certain cipher approximation. It was probably first suggested in the context of a statistical cryptanalysis of DES by Vaudenay [509]. The idea has been developed as an extension of linear cryptanalysis with a more sensitive distinguisher at the output, where quantification is achieved by means of a χ^2 analysis [306, 34, 301]. If the block cipher is truly random then no set of plaintext-ciphertext pairs will produce any significant deviation from a random relationship between plaintext and ciphertext. Conversely, any deviation from random acts as a distinguisher that can then form the basis for an attack on the cipher.

2.2.3.17 Attacks Using Exact Systems of Multivariate Equations

Recent block cipher designs have discouraged the application of linear and differential cryptanalysis by deliberately designing against them. This has prompted cryptanalysts to look elsewhere for effective attack methods. One interesting new direction is also one of the most direct. Every component of a cipher can be described by means of a set of algebraic equations. These component descriptions can then be collected together to form a large system of equations which define the complete cipher. If this system of equations can be solved faster than exhaustive search then the cipher is broken. Clearly an arbitrary construction of the system will be impossible to solve as it will contain far too many variables and equations of too high a degree. However, by careful search one can find systems of

low degree equations in relatively few variables. In particular, the critical component for many ciphers is the S-box, and careful search of state-of-the-art S-boxes has found that the S-box can be exactly represented by surprisingly few low-degree equations, and recent research has led to constructions of sparse systems of quadratic equations. Whereas, say, linear cryptanalysis uses approximations with relatively low probability, the equation system holds with probability one. These algebraic attacks differ in several respects from the standard statistical approaches to cryptanalysis. In particular, these new attacks require only a few known-plaintext queries and, also, the attack complexity does not seem to grow exponentially with the number of rounds of the cipher. It has further been found that well-chosen, sparse, overdefined systems are often easier to solve than critically defined systems. Cryptanalytic methods have been developed by using such systems of equations, where nonlinear terms are treated as independent linear variables in a process called relinearisation [472] (by Shamir and Kipnis — a development from linearisation), and Extended Linearisation (XL) by Courtois *et al.* [137]. These methods, and a variant of XL called Extended Sparse Linearisation (XSL) may perhaps lead to a successful attack on Rijndael (amongst others) — in theory at least, although this is by no means clear at the moment [138, 373]. The main strategy relating to these developing techniques appears to be to find as many equations in as few variables as possible, and of as low degree as possible [74]. The aim is to minimise the number of *free terms*, where a set of free terms is a set of monomials of any degree that are linearly independent. The number of free terms is given as the number of distinct terms minus the number of equations — a term in an equation can be of any degree, in particular for quadratic multivariate equation systems, a term is of degree one or two. Recently, Murphy and Robshaw [375] have embedded Rijndael in a large cipher called the Big Encryption System (BES), which expands Rijndael by its conjugates. The work of Murphy and Robshaw [375, 377] found a system of equations from BES that is simpler than the one developed by Courtois and Pieprzyk [138] for Rijndael, and this suggests that the technique of embedding a cipher in a larger cipher can sometimes lead to simpler equation systems. The paper [138] has also observed that, for the block ciphers they analysed, it is possible that the security of these ciphers does not grow exponentially with the number of rounds. However, it should be noted that some experts do not consider this to be possible.

2.2.3.18 Exploiting Relations Between the Bit-Functions of S-Boxes

A very recent paper by Fuller and Millan [208] has observed that all output bits of the Rijndael S-box, written as Boolean functions of the input bits, can be transformed to one another by means of an affine transformation of the input bits. In other words, let b_i and b_j be two distinct output functions of the Rijndael S-box. Then we can always find a Boolean matrix, $\mathbf{A}$, and a Boolean vector, $\mathbf{B}$, such that,

$$b_i(x) = b_j(\mathbf{A}x + \mathbf{B})$$

This surprising result means that the Rijndael block cipher only uses *one* S-box from eight bits to one bit (used 128 times in each round). At the time of writing

this symmetry has not led to an attack on Rijndael but we include this observation in this section as it is possible that it may lead to attacks in the future. Note however that, if this symmetry does not lead to an attack, then it could in fact be beneficial, leading to more efficient software and hardware implementations of the cipher. Since [208] was posted, Youssef and Tavares [525] proved this result by making use of dual bases over GF(2^n) and trace functions, and generalised the observation to include all S-boxes that utilise bijective monomials. They also extended the result to show that all *coordinate* (bit) functions of the Rijndael round function are equivalent under affine transformation of the input to the round function. Further to this, Biham [54] has shown that such bit-affine relations exist for many of the S-boxes of the block ciphers submitted to NESSIE. In particular, for the S-box, S9, of MISTY1, the rotation of the input by any number of bits does not affect the least significant bit of the output. Tables which summarise these results can be found in Sect. 2.9.

2.2.3.19 Exploiting the Permutation Cycle Structure of a Cipher

This cannot yet be considered an attack technique on a cipher. However, it suggests that one may be able to use the cycle structure of the permutation that defines part or all of the cipher to distinguish the cipher from a cipher chosen at random. In [73] Biryukov and Preneel show that such a technique is particularly suited to ciphers which utilise many involutional elements. The observation was motivated by the submission to NESSIE of Khazad. Khazad is built entirely from involutions which means that one can choose to replace a constituent permutation by its inverse, so allowing the conjugation of some permutations, where conjugation preserves isomorphism of permutations and, therefore, the cycle structure of a permutation. This conjugation can be used to determine sections of the cipher over which the permutation cycle structure is invariant. It follows that a significant number of rounds of the cipher can take on the same cycle structure as the linearly keyed S-box, and, in the future, it may be possible to use this fact to distinguish the cipher from a random cipher.

2.2.3.20 Side-Channel Attacks

Side-channel attacks are a major thread for all implementations of cryptographic algorithms. Annex A is devoted to this subject and addresses the application of side-channel attacks and their countermeasures for block ciphers. Summarizing annex A and [425], algorithms such as Rijndael, Khazad and Camellia seem to be the most suited for implementations resistant to side-channel attacks.

2.2.4 Assessment process

The block cipher submissions were assessed with reference to the above generic common block cipher attacks. Clearly, although some of these attacks are universal, some of them have more relevance to certain block ciphers than others. Thus the block ciphers were assessed against the attacks that seemed most relevant. Furthermore, some block ciphers were analysed using techniques specific to that primitive.

Local statistical testing was applied to components of the block ciphers in order to demonstrate that they have good statistical properties. Global statistical testing was applied to the input-output of the block cipher submissions to show that the data demonstrated good statistical properties. In addition, these tests were also applied on reduced round versions of the block ciphers in order to determine the number of rounds needed to achieve good statistical properties. Further details can be found in [400]. None of the block ciphers tested exhibited any anomalous behaviour. We now summarise the two toolboxes developed by NESSIE.

2.2.4.1 The NESSIE statistical toolbox for block ciphers

This toolbox is part of the general NESSIE test suite for the evaluation of statistical properties of the submissions. We summarise the available tests as follows:

NESSIE Stream Cipher Tests. The block cipher can be used in OFB Mode or Counter Mode. In such cases it produces a stream output and can be viewed as a stream cipher. In these modes it can therefore be tested using the NESSIE stream cipher tests. In both modes the following tests are applied:

- Frequency Test. Splits up the bit sequence into disjoint m-tuples whose distributions are then evaluated statistically.
- Collision Test. The collision test splits up the bit sequence into blocks of a fixed size. A collision occurs if the same block appears more than once. The test statistically evaluates the number of collisions.
- Overlapping m-tuple Test. Shifted windows of m-tuples of words of fixed wordlength are examined and statistically tested along with cyclic shifts of the original sequence.
- Gap Test. Splits up the bit sequence into disjoint m-tuples which are then interpreted as binary integer representations. The length of gaps, where the numbers are not within a numerical range given as a parameter of the test, are evaluated statistically. This test is also applied to cyclic shifts of the original sequence.
- Run Test. Splits up the bit sequence into disjoint m-tuples which are then interpreted as binary integer representations. The lengths of subsequences of consecutive, strictly increasing numbers are evaluated statistically.
- Coupon Collector's Test. Splits up the bit sequence into disjoint m-tuples. The number of subsequence m-tuples it takes until all possible 2^m m-tuples have appeared, is evaluated statistically. The test is also applied to cyclic shifts of the original sequence.
- Universal Maurer Test. Splits up the bit sequence into disjoint m-tuples and evaluates statistically how many m-tuples later an m-tuple re-appears in the sequence. The result of this test is closely related to the entropy of the bit sequence.
- Poker Test. Splits up the bit sequence into disjoint m-tuples, and this sequence of m-tuples is then split up into subsequent disjoint k-tuples of m-tuples. The poker test statistically evaluates how many of the m-tuples in a k-tuple are equal. The test is also applied to cyclic shifts of the original sequence.

- Fast Spectral Test. Applies the fast Walsh transform to the given sequence and uses the spectral results to assess the randomness of the sequence.
- Correlation Test. Determines in how many places the original sequence and the sequence shifted by n bits have the same value for shifts up to the length of the original sequence.
- Rank Test. Sequence bits are used to fill square matrices and the rank of the matrices over GF(2) is evaluated statistically.
- Linear Complexity Test. The Berlekamp Massey algorithm is used to determine the length of the shortest linear feedback shift register which can produce the given bit sequence. For the linear complexity profile, this is done for the first 1,2,3.. bits of the sequence.
- Maximum Order Complexity (MOC) Test. Determines the length of the shortest possibly non-linear feedback shift register which can produce the given bit sequence For the MOC profile, this is done for the first 1,2,3.. bits of the sequence.
- Ziv Lempel Complexity Test. Measures how well a bit sequence can be reconstructed from earlier parts of the bit sequence.
- Dyadic Complexity Test. The Dyadic Complexity Test is an implementation of the complexity measure suggested by Goresky and Klapper [291] for sequences of bits. It determine the length of the shortest feedback shift register with carry which can produce the given bit sequence.
- The Percolation Test is the simulation of a forest fire. The bit sequence to be tested determines where trees are standing in the simulated forest. The test evaluates statistically how fast a fire propagates in the simulated forest.
- Constant Runs Test. For the constant runs test, the sequence of bits is subdivided into runs, that is maximal disjoint subsequences of consecutive 0s and 1s. The frequencies of these runs of the various lengths are evaluated statistically.

These stream cipher tests were applied to the block cipher submissions, and none of the block ciphers exhibited any anomalous behaviour. The detailed results of the statistical tests are available as NESSIE public reports.

NESSIE Block Cipher Tests. NESSIE has also developed a test suite to test the block ciphers as block ciphers. Details can be found in [400, 471]. The following tests are applied:

- Dependence Test. Evaluates the dependence matrix and the distance matrix of the cipher. The degree of completeness, the avalanche effect, and the Strict Avalanche Criterion (SAC) are also computed. We now define these criteria:
  - A function is complete if each output bit depends on each input bit.
  - The avalanche effect occurs when, on average, approximately one half of the output bits change whenever a single input bit is complemented.
  - The SAC is satisfied if each output bit changes with probability one half whenever a single bit is complemented.
- Linear Factors Test. Used to determine whether there are any linear combinations of output bits which, for all keys and plaintexts, are independent of one or more key or plaintext bits. Such a combination is called a linear factor. It

is impossible to do this for all keys and plaintexts, so the test is only applied to a sufficiently large number of pairs of random keys and plaintexts.

The above block cipher tests were applied to the full cipher, and then to reduced-round versions to determine the minimum number of rounds for which the various criteria were all satisfied.

These block cipher tests were applied to the block cipher submissions, and none of the block ciphers exhibited any anomalous behaviour. The detailed results of the statistical tests are available as NESSIE public reports.

2.2.4.2 The NESSIE toolbox for differential and linear cryptanalysis

NESSIE has developed a software toolbox to aid in the analysis of block ciphers. This is described in detail in [49]. The system currently examines the block cipher on three levels.

1. Statistical properties of the basic building blocks are found.
2. Properties of complete rounds are found.
3. Properties are found over several rounds.

It is intended to extend the system to include aspects of cryptanalysis of the full cipher at a later date. We now describe the analysis for each of the above three categories.

Building Blocks. Any re-arrangement mapping and any nonlinear S-box mapping between an input vector of bits and an output vector of bits can be analysed, where the input and output vectors need not be of the same size. The mapping can be validated for one-to-one mapping, tested for the period of inherent permutation cycles and the existence or otherwise of fixed-points in any inherent permutations, and checked for output balance/distribution.

The system also provides difference distribution tables and linear approximation tables for the inherent S-boxes, for use in differential and linear cryptanalysis, respectively.

One Round. The system identifies dependencies between the output bits of round i and the output bits of round $i + 1$, and provides an example for linear and differential one-round characteristics and their probability.

Several Rounds. The system identifies the most important 3-round characteristics, and suggestions for good 5-round characteristics, with corresponding probabilities.

2.3 Overview of the common designs

Modern-day ciphers are designed to withstand known cryptanalytic attacks. However there is not complete agreement as to the best design philosophy to use, hence the variety of designs submitted to NESSIE. We now summarise some of the most important of these design philosophies, where any given cipher may utilise one or more of these philosophies.

2.3.1 Feistel ciphers

A basic Feistel cipher takes $2t$ plaintext bits, and is a permutation, F, which uses m round permutations, F_i, so that,

$$F = F_0 \circ F_1 \circ \ldots \circ F_{m-1}$$

where '$\circ$' means composition.

Round i acts on half the input bits, the t bits, R, by means of the keyed function, f_i, and XORs the result with the other half of the bits, the t bits, L. It then swaps the left and right halves. Thus we have,

$$[L', R'] = F_i[L, R] = [R, L \oplus f_i(R)]$$

where $[L'\, R']$ becomes the new input $[L\, R]$ to round $i + 1$. Although F and the F_i must be permutations, the f_i need not be. It takes two rounds before all plaintext bits have been acted on in a nonlinear way. DES, MISTY1, and Camellia are all essentially Feistel ciphers. The Feistel structure allows decryption to be accomplished using the same process, but with the sub-keys used in reverse order. Sometimes the Feistel structure may be nested such that individual components also have a Feistel structure. This is the case for MISTY1.

2.3.2 Substitution-Permutation Networks (SPNs)

A substitution-permutation network (SPN) separates the role of confusion (substitution) and diffusion (permutation) in the cipher. As with most ciphers, the cipher is decomposed into iterative rounds where each round comprises a layer of S-boxes, followed by a permutation/diffusion layer. The S-box layer provides the nonlinearity (confusion), and the permutation layer provides the rapid diffusion. In fact, the SPN has more recently come to include ciphers where the diffusion layer is not a permutation (re-wiring) but is a linear or affine transformation. Moreover these linear transformations are often derived from Maximum-Distance-Separable (MDS) error-correcting codes, where the high minimum distance of the code implies a high diffusion rate. The separating of the tasks of confusion and diffusion allows the designer to maximise nonlinearity for the S-box, and maximise information spread for the diffusion layer. Rijndael, Khazad, Hierocrypt, and SAFER$_{++}$ are all examples of SPNs. In fact a Feistel cipher is also a type of SPN. Sometimes the SPN structure may be nested, such that the individual S-boxes are themselves SPNs. Hierocrypt is an example of such a cipher, and Rijndael can also be described in this way [33]. In fact, one can always compose a large S-box as a number of layers of smaller S-boxes — and this is the case for the S-boxes of Khazad and Anubis.

2.3.3 Resistance against differential and linear cryptanalysis

Many recent block cipher designs have attempted to maximise the resistance of the cipher to differential and linear attack. This is usually achieved in two ways:

- Firstly, the elemental S-boxes are designed to be highly nonlinear, i.e. so that they can only be poorly approximated by linear equations, and so that differential characteristics can only be propagated through the S-box with small probability. These design criteria are evident in the parameters of low linear bias and low differential probability associated with the cipher.
- Secondly, the diffusion components are designed to spread the information to the whole cipher block as rapidly as possible. This rapidity is sometimes measured in terms of the branch number or minimum distance of the diffusion layer. The resilience of the component functions of the S-box is also a measure of the diffusion properties of the S-box, where Resilience t indicates that the component function is completely independent of any t or fewer of its constituent variables. The result of rapid diffusion is that, in any linear or differential attack, which essentially uses elemental approximations, too many of the elements of the block cipher must be approximated, thereby rendering the attack useless.

For some recent ciphers, e.g. Khazad, the emphasis on extremely rapid diffusion has reduced the perceived need for optimal nonlinearity of the S-box — reasonably good nonlinearity is considered good enough. This has enabled the S-box to be chosen randomly from a suitable subspace of the set of S-boxes, unlike, for instance, the S-boxes of Rijndael, Camellia, or MISTY1, which optimise nonlinearity at the price of, in the case of Rijndael and Camellia, a non-random, algebraic S-box which may lay itself open to algebraic attack.

2.3.4 Mini-ciphers and reduced rounds

Although the attacks on block ciphers described in this chapter provide a means of assessing the security of a cipher, it is still the case that cryptanalysis of a real cipher is an extremely big task. As an aid to analysis, some cipher designs naturally allow the description of mini-versions of their cipher over a reduced wordsize and reduced plaintext and key blocksize. One can then analyse these small versions of the cipher and extrapolate the conclusions to the larger real cipher. Ciphers which naturally support mini-versions include RC6, SAFER$_{++}$, IDEA, Khazad, Rijndael, and numerous others. In fact, most ciphers already incorporate this philosophy in the sense that they decompose the cipher into iterated round functions. This leads to attacks on reduced rounds which can then be extrapolated to the full-round cipher. Recently, [299] has suggested a *very* rough *rule-of-thumb* computation for the minimum number of rounds, R, required to make a block cipher secure, given by $R \geq \frac{dN}{w}$ where w is the minimum word size input to a confusion stage in the cipher (for instance this could be the size of the smallest S-box used), d is the maximum number of rounds it takes for one word to be input to a confusion stage (for a Feistel cipher, $d = 2$), and N is the size of the plaintext input to the cipher, in bits.

2.3.5 Simple as opposed to complicated designs

Some ciphers are deliberately very difficult to analyse. Others are deliberately relatively simple to analyse, being built out of conceptually simple primitives. For example, ciphers that add in the key via XOR are often easier to analyse than ciphers that use nonlinear key input. The security of ciphers that use nonlinear keying can often be far more key-dependent than the security of ciphers that use linear keying. This makes them harder to analyse but also makes it more likely that there are *weak* keys for which the enciphering function is weak [376]. The current trend is towards ciphers that are simple to describe and analyse, as a cipher that cannot be analysed cannot be declared *secure*. Of course there are different notions of simplicity, for instance, although RC6 and Khazad are both *simple* designs, they are not simple in the same way.

2.3.6 A separate key-schedule

Virtually all modern block cipher designs separate the key-schedule from the enciphering/deciphering process. Typically the cipher will take in a master key and, from this key, generate round keys which will then be used to key the rounds of the encryption process. Thus there are usually two parallel processes in a block cipher. One encrypts, whilst the other generates the key-schedule.

2.3.7 The use or otherwise of S-boxes

The use of S-boxes in block ciphers is mainly as a means of introducing nonlinearity. They are usually envisaged as large look-up tables, substituting one value for another. Examples of the type of cipher that uses S-boxes are DES, MISTY1, Rijndael, Camellia. In contrast, some ciphers do not use explicit S-boxes but introduce nonlinearity by means of well-known algebraic operations such as integer addition, integer multiplication, log, or exponentiation. Examples of these ciphers are RC6, IDEA, and SAFER$_{++}$. However, this distinction is a bit tenuous as, for example, a log function or a multiplication can be viewed as an S-box, and the S-box used by Rijndael and Camellia is essentially x^{-1} over GF(2^8). A more significant distinction is between key-dependent and non-key-dependent S-boxes — for instance, whereas multiplication by a constant can be implemented using a fixed look-up table, multiplication by a key cannot. Another useful design distinction is that some ciphers use large S-boxes (e.g. 8 by 8 or bigger), and some ciphers use smaller S-boxes (e.g. 4 by 4). It has recently been argued that the use of S-boxes that are too small is not a good idea [138].

2.3.8 Ciphers which are developed from well-studied precursors

It is an accepted fact that block ciphers need many years of analysis before they can be labelled *secure*. It is therefore common for block cipher designers to incrementally improve their own previous designs and/or incorporate aspects of previous block cipher designs. This is a cautious strategy which seeks to add to

the perceived security of the new design. For instance, an attack on the key-schedule of a previous version of SAFER [295, 298], and an improved diffusion layer, have led to upgrades which have been incorporated in SAFER$_{++}$. (Note, however, that it could be argued that the designers have made drastic changes in designing SAFER$_{++}$ from previous designs which, to some extent, invalidate previous analysis — for instance they completely changed and reduced the complexity of the mixing layers). As another example, RC6 uses exactly the same key schedule as RC5.

2.3.9 Making encryption and decryption identical

It is clearly desirable for the encryption and decryption processes of a cipher to be identical in as many ways as possible, as this enables the re-use of hardware or software resources. Feistel ciphers are designed in this way. Moreover, any cipher component which is an involution will be, by definition, the same in reverse. Khazad uses only involutions so encryption and decryption are identical for such a cipher (apart from the order of the sub-keys). In contrast, Rijndael uses different encryption and decryption operations, although the encryption and decryption speeds for Rijndael are virtually identical in software, differing only slightly for 8-bit machines. In hardware, the speed of both operations for Rijndael is the same, decryption requiring just a little bit more hardware.

2.3.10 Hash functions as block ciphers

A hash function is usually not intended to be used as a block cipher, but it can be. The hash function will take a K-bit message and hash an initial N-bit value to a final N-bit value which is called the hash of the message. The hash function is designed so that it is impossible to ascertain the K-bit message from the N-bit hash. K is always bigger than N. If we redefine the K-bit message as a K-bit key, and the N-bit initial value as the input plaintext, then the output hash becomes the N-bit output ciphertext. For example, SHACAL-1 and SHACAL-2 are block cipher submissions to NESSIE and are block ciphers built from the hash functions, SHA-1 and SHA-2, respectively. A secure hash function should ideally minimise the number of (inevitable) N-bit output hash collisions for different initial K-bit messages, given that the initial N-bit value is fixed. This is called collision-resistance. Transferring this criterion to the block cipher regime implies that, given a fixed input plaintext, a secure block cipher should seek to minimise the number of ciphertext collisions for different key inputs. However, it is not at all clear at present whether weaknesses in collision-resistance for the hash function translate into attacks on the associated block cipher. It is an open problem.

2.3.11 Current standards

The Data Encryption Standard, DES, is now considered to have too small a key space for today's security requirements. Triple-DES (DES encryption, followed

by DES decryption, followed by DES encryption) uses a $3 \times 56 = 168$-bit key and is considered to be a practical solution to the security weakness of DES. Triple-DES has been included in the NESSIE evaluation as a benchmark cipher with which to compare the other block cipher submissions, although it is somewhat inefficient in comparison.

The new Advanced Encryption Standard, AES [391], also known as Rijndael, has also been included in the NESSIE evaluation, both Rijndael-128 and Rijndael-256, once again to act as a benchmark with which to compare the other block cipher submissions. Note also that RC6, one of the five AES finalists, has also been submitted to NESSIE, as has SAFER$_{++}$ which is a modified version of SAFER+ which was in the AES.

Currently there is an ongoing assessment of new cryptographic primitives to be adopted as the new ISO standard. The block ciphers being considered are as follows: For 64-bit plaintext, TDEA (Triple-DES) — 128 or 192-bit key, MISTY1 — 128-bit key, and Khazad — 128-bit key. For 128-bit plaintext, the AES (Rijndael) — 128, 192, or 256-bit key, Camellia — 128, 192, or 256-bit key, SEED — 128-bit key, RC6 — 16 - 256-byte key, and CAST-128.

2.3.12 Block cipher primitives

The deadline for submissions to the NESSIE project was September 29, 2000, just before NIST's announcement that the AES block cipher was to be Rijndael. Nevertheless, there were 17 block ciphers submitted to NESSIE.

The NESSIE call for primitives specified the following security levels for block ciphers.

- *High*. Key length of at least 256 bits. Block length at least 128 bits.
- *Normal*. Key length of at least 128 bits. Block length at least 128 bits.
- *Normal-Legacy*. Key length of at least 128 bits. Block length 64 bits.

There have been two phases to the evaluation process, Phase I and Phase II. In total, 8 ciphers were selected for Phase II, with SAFER$_{++}$ and RC6 both being selected for two security levels.

The submissions are classified in the table below according to the security levels specified in the NESSIE call, and those selected for Phase II of NESSIE are indicated.

Name of cipher	Block Size (bits)			High	Normal	Normal-Legacy	Phase II
CS-Cipher	64					X	
Hierocrypt-L1	64					X	
IDEA	64					X	√
Khazad	64					X	√
MISTY1	64					X	√
Nimbus	64					X	
NUSH	64	128	256	X	X	X	
SAFER$_{++}$	64	128		X	X	X	√ √
Grand Cru		128			X		
Noekeon		128			X		
Hierocrypt-3		128		X	X		
Q		128		X	X		
RC6		128		X	X		√ √
SC2000		128		X	X		
Anubis		128		X			
Camellia		128		X			√
SHACAL-1			160		X		√
SHACAL-2			256		X		√

Notes

– NUSH was designed with three different block sizes: 64 bits, 128 bits, and 256 bits.
– RC6 has a variable block length $4w$ bits, where $w \geq 32$ is recommended by the designers.
– SAFER$_{++}$ has two variants, one with 64-bit blocks and one with 128-bit blocks.

2.4 64-bit block ciphers considered during Phase II

The 64-bit block ciphers selected for Phase II of NESSIE were IDEA, KHAZAD, MISTY1, SAFER$_{++64}$, and Triple-DES. None of these ciphers has been broken so the following security evaluation identifies weaknesses that occur in reduced-round versions of the ciphers, and identifies weaknesses that may lead to more effective attacks in the future. We first describe each cipher in some detail, along with the most important attacks known on each cipher. Note that the algorithms given here are not complete specifications, but references are given to complete specifications which may be found on the NESSIE website. After discussing each cipher we summarise and compare some of the distinguishing features of the ciphers, identifying potential weaknesses and noting the best-known attacks, as shown in Tables 2.14 and 2.15 of Sect. 2.9.1.

2.4.1 IDEA

2.4.1.1 The design

IDEA [325] operates on 64-bit blocks of plaintext and ciphertext and is controlled by a 128-bit key. It is specified to require 8.5 encryption rounds to achieve an acceptable security margin. It claims to achieve high security by concatenated

use of three arithmetic operations from two dissimilar algebraic groups (two of the groups are isomorphic), namely:

– Addition mod 2^{16}.
– Multiplication mod $2^{16}+1$ (where the all 0 bit word is interpreted as 2^{16}).
– Bitwise exclusive OR.

Note that (since $2^{16}+1$ is prime) the multiplication operation is isomorphic to the addition operation. The combined use of these operations is used to achieve high nonlinearity and completely replace the more conventional use of S-boxes for this task. This can often result in relatively efficient implementations, in software because many processors have special-purpose multiplication operators, and in certain hardware scenarios where memory is at a premium, as S-boxes usually require large look-up tables. Although the software realisation of modular multiplication can be optimised [55], hardware implementation of multiplication will always cost many gates, and this means that hardware implementations of IDEA are not compact. The cipher is designed so that encryption and decryption are identical apart from key input. The encryption operation is shown in Fig. 2.3, and comprises 8 identical steps (rounds) followed by an output transformation. Round 1 is shown in detail in Fig. 2.3.

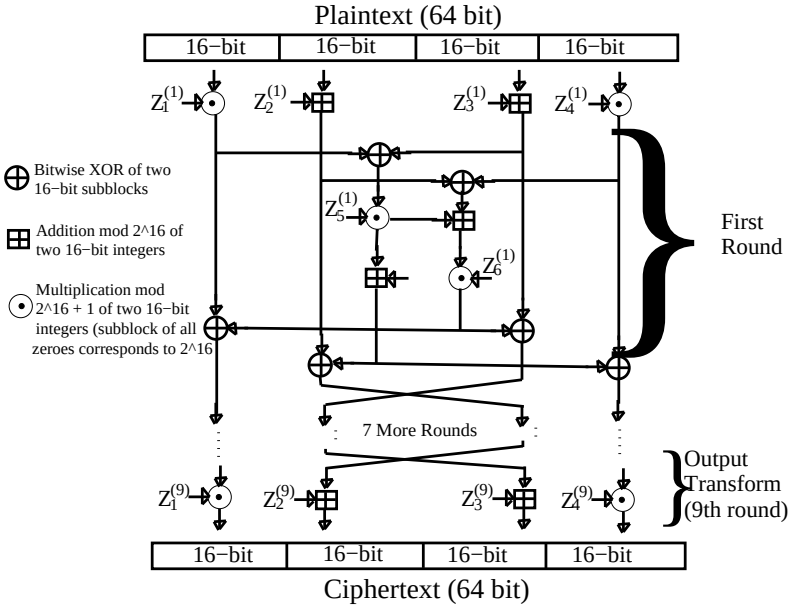

Fig. 2.3: The IDEA Block Cipher

The 64 bits of plaintext are partitioned into four 16-bit subblocks and all encryption operations take 16-bit inputs and produce 16-bit outputs. The Z's of Fig. 2.3 refer to six 16-bit key subblocks per round (with four 16-bit key subblocks

for the subsequent output transformation). A total of $8 \cdot 6 + 4 = 52$ different 16-bit subblocks are generated from the 128-bit key. We describe this key schedule below.

It should be noted from Fig. 2.3 that after each round the 16-bit partitions are re-ordered before commencing the next round. The process of round one is repeated for 7 more rounds, and after 8 rounds the output transformation is combined with the final four of the key subblocks, as shown in Fig. 2.3.

A fundamental design criterion is the following:
At no point during the encryption process is the same algebraic group operation used contiguously.

It has been shown by Wernsdorf [517] that the multiply-addition box at the centre of the round of IDEA generates the alternating group on $\{0,1\}^{32}$ and [517] conjectures that the alternating group is also generated by the complete round of IDEA. These large groups exclude the possibility of several types of regularity for IDEA.

The key schedule takes a 128-bit key and turns it into 52 16-bit key subblocks, as follows:

- First, the 128-bit key is partitioned into eight 16-bit subblocks and these form the first eight key subblocks.
- The 128-bit key is then cyclically shifted to the left by 25 positions, after which the resulting 128-bit block is once again partitioned into eight 16-bit subblocks and these form the next eight key subblocks.
- The above process is then repeated until all required 52 16-bit key subblocks have been generated.

Decryption is identical to encryption apart from the use of different key subblocks, where these 52 16-bit key subblocks are as follows: each subblock used for decryption is the inverse of the key subblock that was used for encryption, where the inverse is taken with respect to the algebraic group operation associated with that particular key subblock. Moreover, these key subblocks must be applied in reverse order to the order in which they were used for encryption.

2.4.1.2 Security analysis

IDEA has been widely studied for over a decade and few security flaws have been found. In fact, no attack against 5 or more of its 8.5 rounds has been found. However, IDEA does exhibit weak key classes of substantial size, and it can be argued that a flawless cipher should have no weak keys at all. IDEA has been included in the popular cryptographic package, PGP, although not for some time, and is one of the best known and most widely used ciphers. The following attacks are some of those that have been applied to IDEA prior to and during NESSIE. Meier proposed a differential attack on 2.5 rounds [362], where it was observed that, if $a + b < 2^{16}$, then $a + b \mod 2^{16} = a + b \mod 2^{16} + 1$, and this in turn implies that that multiply-addition part of IDEA is linear about $\frac{1}{4}$ of the time. The paper of Borst *et al.* [101] developed a truncated differential attack on 3.5 rounds, and a miss-in-the-middle attack on 4.5 rounds was proposed by Biham

et al. [58]. Moreover, it has been shown [321, 146, 244, 77] that certain weak-key classes exist for IDEA. For instance, [146] exploited the observation that $-x \mod 2^{16}+1 = x \oplus 11\ldots101$ whenever x_1, the second least significant bit of x, is 1. In fact, attacks which exploit potential weaknesses in the modular multiplication, mod $2^{16}+1$, are of particular interest for IDEA, for instance, Harpes *et al.* [239] attack IDEA by applying a quadratic residue homomorphism between $(\mathbb{Z}/n\mathbb{Z})^*$ and $\mathbb{Z}/2\mathbb{Z}$, where $n = 2^{16}+1$, and a recent paper by Borisov *et al.* [99] further exploits homomorphisms between modular multiplication and XOR to identify large weak key classes for certain variants of IDEA where addition has been replaced with XOR. They show that for 2^{112} of the keys there exists a multiplicative differential characteristic over 8 rounds that holds with probability 2^{-32}. Recently, Raddum [448] has demonstrated a better attack on this variant of IDEA, called IDEA-X, using a similar technique, but this time using XOR-differentials. The advantages of the attack in [448] are that it works for all keys, and it is only necessary to change the first two additions in each round to XORs. In [448] a differential characteristic has been found that holds with probability 2^{-30} over 8 rounds and exploits the fact that $Z_{2^{16}}$ and $\mathrm{GF}(2^{16}+1)^*$ are both cyclic groups, and therefore isomorphic. Essentially, each element a in $\mathrm{GF}(2^{16}+1)^*$ can be written uniquely as,

$$a = g_{15}^{x_{15}} \cdot g_{14}^{x_{14}} \cdot \ldots g_0^{x_{15}}$$

where $\cdot$ means field multiplication, $x_i \in \{0,1\}$, g_0 is a primitive element of $\mathrm{GF}(2^{16}+1)$, and $g_i = g_{i-1}^2$. We write this as $a = \mathbf{g}^{\mathbf{x}}$. Then $\phi(a) = \mathbf{x}$ is an isomorphism, mapping multiplication, mod $2^{16}+1$, for a to addition, mod 2^{16}, for $\mathbf{x}$. For $a, b \in \mathrm{GF}(2^{16}+1)$, $ab = \phi^{-1}(\phi(a)+\phi(b))$. The above isomorphism is then used to pass an input XOR difference through the mod $2^{16}+1$ multiplier, then unchanged through a subsequent (integer) additive key bit, and finally back to an XOR difference. The complete XOR to XOR difference holds with probability $\frac{1}{4}$. This characteristic is used to attack IDEA-X as defined by [99]. However, the attack of [448] in fact requires fewer of the integer additions to be changed to XORs so the cipher it attacks is closer to the real IDEA than the cipher of [99].

Until the commencement of NESSIE the best attack on IDEA was by Biham *et al.* [58] on 4.5 out of 8.5 rounds of IDEA.

Nakahara *et al.* [380] report on variants of the SQUARE attack applied to reduced-round versions of the PES and IDEA block ciphers (PES is a forerunner of IDEA). Attacks on 2.5 rounds of IDEA require $3 \cdot 2^{16}$ chosen-plaintexts and recover 78 key bits. A SQUARE related-key attack is applied on 2.5 rounds of IDEA and recovers 32 key bits, with 2 chosen-plaintexts and 2^{17} related keys. Implementations of the attacks on 32-bit block mini-versions of both ciphers confirmed the expected computational complexity. Although the attacks do not improve on previous approaches, this report shows new variants of the SQUARE attack on word-oriented block ciphers like IDEA and PES.

Biryukov *et al.* [77] present a large collection of new weak-key classes on the full 8.5 round IDEA cipher proposed, using the property that some multiplicative keys which are 0 or 1 turn modular multiplication into a linear operation. The weak-key classes in this paper contain 2^{53} - 2^{64} weak keys. The novelty

of the approach is in the use of the boomerang distinguisher for the weak-key class membership test, as developed by Wagner [514]. Large weak-key classes for reduced-round versions of IDEA are also shown. It appears that the existence of relatively large weak key classes for IDEA is due to the use of modular multiplication as the main nonlinear part of the cipher, and because the key schedule is also a linear process. The results suggest a redesign of the key schedule of IDEA.

Finally, statistical evaluation of IDEA [447], following the recommendations of the NESSIE statistical evaluation process for block cipher submissions, does not indicate a deviation from random behaviour.

Table 2.1. Comparison of attack requirements on reduced-round IDEA

Attack Type	Year	Reference	#Attacked Rounds	Data † Complexity	Time Complexity
Differential	1993	[362]	2	2^{10}	2^{42}
Improved-Square	2002	[158]	2	23	2^{64}
Differential	1993	[145]	2.5	2^{10}	2^{32}
Differential	1993	[362]	2.5	2^{10}	2^{106}
Square attack	2000	[380]	2.5	$3 \cdot 2^{16}$	2^{58}
Square attack	2000	[380]	2.5	2^{32}	2^{59}
Square	2000	[380]	2.5	2^{48}	2^{79}
Related-Key Square	2001	[380]	2.5	2	2^{33}
Improved-Square	2002	[158]	2.5	55	2^{81}
Differential-Linear	1996	[101]	3	2^{29}	$0.75 \cdot 2^{44}$
Improved-Square	2002	[158]	3	71	2^{71}
Improved-Square	2002	[158]	3	2^{33}	2^{82}
Truncated Differential	1997	[309]	3.5	2^{56}	2^{67}
Miss-in-the-middle	1999	[58]	3.5	$2^{38.5}$	2^{53}
Improved-Square	2002	[158]	3.5	103	2^{103}
Improved-Square	2002	[158]	3.5	2^{34}	2^{82}
Miss-in-the-middle	1999	[58]	4	2^{38}	2^{70}
Related-Key ‡ Differential-Linear	1998	[244]	4	38.3	38
Improved-Square	2002	[158]	4	2^{34}	2^{114}
Miss-in-the-Middle	1999	[58]	4.5	2^{64}	2^{112}

† number of chosen texts.
‡ the differential-linear attack requires two related keys.

In view of the comparatively large amount of cryptanalysis undertaken on IDEA, Tables 2.1 and 2.2 summarise many of the known chosen-plaintext and weak-key attacks on IDEA, respectively. The most effective of these attacks are then included in the summary table, Table 2.14, in Sect. 2.9.1 at the end of this chapter.

Table 2.2. Comparison of attack requirements on IDEA for weak-key classes.

Attack Type	Year	Reference	#Attacked Rounds	**Weak-key Class Size**	Data † Complexity	Time Complexity
Differential	1993	[146]	8.5	2^{51}	2	2^{12}
Differential-Linear	1998	[244]	8.5	2^{63}	20	4
Boomerang	2002	[77]	4.5	2^{101}	2^{18}	2^{18}
Boomerang	2002	[77]	5	2^{97}	2^{10}	2^{10}
Boomerang	2002	[77]	5	2^{95}	4	4
Boomerang	2002	[77]	8.5	2^{53}	4	4
Boomerang	2002	[77]	8.5	2^{57}	2^{11}	2^{11}
Boomerang	2002	[77]	8.5	2^{64}	2^{16}	2^{16}

† number of chosen texts.

2.4.2 Khazad

2.4.2.1 The design

Khazad [30] is a 64-bit Substitution-Permutation (SPN) block cipher with a 128-bit key and is designed to operate over 8 rounds. Khazad has many similarities to Rijndael, but works on 64-bit plaintext blocks. A primary *selling-point* of Khazad is that **all** components of the algorithm use involutions. This involutional structure is important for implementations and to make encryption and decryption equivalent operations apart from the order of the key schedule. This also implies equal security for encryption and decryption. These involutions include the S-box, S, (i.e. $S[S[x]] = x$), and the 64-bit linear diffusion mapping, based on an MDS code and represented by a matrix H, (i.e. $H^2 = I$). The matrix H that realises this diffusion is also chosen to have lowest possible Hamming Weight and, for smart-card implementation, lowest possible integer weight over 8-bit words. The cipher also emphasises the Wide-Trail Design Strategy [144] as the linear diffusion layer is based on a Maximum-Distance-Separable (MDS) code with a high branch number of 9. This ensures that there is full mixing after a single round. This, in combination with a reasonably nonlinear S-box ensures strong resistance to linear and differential attacks, and to related-key attacks. In the original submission the 8×8-bit S-box was randomly chosen to avoid any obvious internal structure. However, this made the S-box costly to implement in hardware so the authors have modified the submission by replacing this S-box with another S-box which is more amenable to hardware implementation. This replacement was also made in order to correct a small security flaw where the maximum bias of a linear characteristic through the S-box was slightly underestimated (this did not lead to any security failure, as the high diffusion of the round more than compensates for this). The S-box has also been chosen to have no fixed point and is made out of six 4×4-bit smaller S-boxes, P and Q, which are involutions with optimal nonlinearity characteristics, as shown in Fig. 2.4. The designers expect that the hardware required to implement this S-box should be around $\frac{1}{5}$ of that for the Rijndael S-box, which is also an 8×8-bit S-box. Essentially, this potential reduction in implementation cost is bought at the price of slightly weaker nonlinear characteristics. However it should be noted that, unlike in Camellia or Rijndael, the avoidance of an S-box using x^{-1} *may* increase the

minimal size of any potential set of sparse quadratic equations on which to base an attack of the form proposed by Courtois and Pieprzyk in [138] or by Murphy and Robshaw in [375]. Preliminary evidence for this claim is given by Biryukov and de Cannière [74], where a description of the 8×8 Khazad S-box requires 28 quadratic equations, but a description of the 8×8 Rijndael S-box only requires 23 quadratic equations

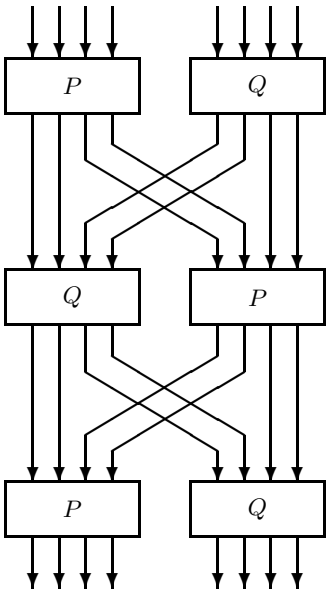

Fig. 2.4: Structure of the Khazad S-box. Both P and Q are pseudo-randomly generated involutions; the output from the upper and middle nonlinear layers are mixed through a simple linear shuffling.

Key addition is achieved via XOR (which is also an involution). This has the advantage that no weak keys exist as the nonlinearity operations are key-independent. The Khazad key schedule uses a 9-round 128-bit Feistel scheme with an internal F-function being a round of Khazad with constants used as the 64-bit key. The user-specified 128-bit key is used as a plaintext and the intermediate 64-bit values after each round become the subkeys of Khazad. The key schedule expands the 128-bit key into 64-bit round keys, $K^0, K^1, \ldots, K^R$, plus two initial round keys K^{-2} and K^{-1}, and these round keys are generated as follows:

$$K^r = \rho[c^r](K^{r-1}) \oplus K^{r-2} \quad 0 \leq r \leq R$$

where $\rho[c^r](K)$ is the round function of Khazad which takes as input parameters the pre-chosen constant c^r and round key input, K.

Running R rounds of the complete cipher apply the operation $\alpha_R[K^0 \ldots K^R]$ to the plaintext, where,

$$\alpha_R[K^0 \ldots K^R] = \sigma[K^R] \circ \gamma \circ (\bigcirc_1^{r=R-1} \rho[K^r]) \circ \sigma[K^0]$$

where σ, γ, and ρ are the operations of key addition, nonlinear substitution and the full round function, respectively.

2.4.2.2 Security analysis

The designers [30] state that, for Khazad, there is no 2-round differential characteristic with probability higher than 2^{-45} and no 2-round linear approximation with bias of more than approximately $2^{-20.7}$. Related-key attacks were also claimed to be infeasible. A SQUARE attack on 3 rounds of Khazad is described by Barreto and Rijmen in [30], requiring 2^8 key guesses $\times$ 2^8 chosen plaintexts $= 2^{16}$ S-box look-ups to recover one key-byte. A variant on this requires 2^9 chosen plaintexts, 2^{16} S-box look-ups, and 64-bit key guessing, which can be extended to an attack on 4 rounds requiring 2^{80} S-box look-ups. An extension of the Biham-Keller impossible differential attack on 5 rounds of Rijndael [66] can be applied to 3 rounds of Khazad, requiring 2^{13} chosen plaintexts and 2^{64} encryptions. The designers [30] also claim security against truncated differential attacks after 4 rounds, and against Interpolation attacks and Boomerang attacks. Analysis by Biham *et al.* [59] supports these claims. However, further analysis reveals 4 linear characteristics with bias $\simeq \frac{17}{256}$, whereas [30] originally claimed a maximal absolute bias of $\frac{13}{256}$. This resulted in the tweak to the S-box mentioned earlier. The Gilbert-Minier collision attack, which works better than the SQUARE attack on Rijndael, will not work for Khazad since it requires full 64-bit collisions, whereas Rijndael only requires 4-byte collisions owing to slower mixing (diffusion).

Generalised linear characteristics for Khazad with maximum and minimum bias are also found by Parker in [426]. It is found that Khazad already has a moderately low-bias characteristic with Peak-to-Average Power Ratio = 16.0 with respect to the Walsh-Hadamard Transform, and this confirms the bias quoted by the designers. But, as with all S-boxes, this bias increases significantly when approximated by more general linear expressions, where a more general linear expression is here meant to mean $\omega^{f(\mathbf{x})}$, where ω is an rth complex root of 1, and $f(\mathbf{x})$ is a linear expression in the variables, x_i, mod r. For $r \gg 2$ this set of linear expressions is much larger than for binary linear expressions, so much closer approximations can be found.

One advantage that Khazad may have over Rijndael is that the S-box of Khazad does not depend on a simple mathematical function, namely x^{-1}, which is potentially open to attack. However, the nonlinearity of the Rijndael S-box is stronger. Moreover, although the S-box of Khazad is claimed by the designers to be implementable with about $\frac{1}{5}$ of the hardware of that needed for Rijndael, a recent paper by Fuller and Millan [208] shows that the output functions of Rijndael are all affine transformations of the same function. This suggests a potential extra hardware saving for the Rijndael S-box, although this simplification may perhaps later be exploited as a security weakness, as all the output bits of the S-box are simply affine relations of one another. In contrast, the Khazad S-box is relatively unstructured and therefore less of a candidate for potential algebraic attacks. Furthermore, Biham [54] has shown that no such affine relationships exist for the Khazad S-box (see Sect. 2.9.1).

One of the major hindrances to cryptanalysis of Khazad is the extremely high rate of diffusion for the cipher and it may be that new attacks on such ciphers have to exploit more global *iterative* properties of the cipher, such as its permutation structure. Recently some new techniques which analyse the permutation cycle structure of Khazad have been proposed by Biryukov and Preneel [73], suggesting promising avenues for future attacks on such ciphers. The techniques exploit the involution structure of Khazad by showing that the involutions maintain the permutation cycle structure through successive rounds of Khazad. We can write 5-round Khazad as follows:

$$k_0SMk_1SMk_2SMk_3SMk_4Sk_5$$

where the k_i are the key XORs, S is the nonlinear S-box layer, and M is the MDS linear diffusion layer. By passing the k through M we can rewrite the above as

$$k_0Sk_1'[MSM]k_2Sk_3'[MSM]k_4Sk_5.$$

[73] first investigates the permutation cycle structure of MSM, noting that, as $M = M^{-1}$ (an involution), $MSM = MSM^{-1} = A$, where A is a permutation isomorphic to S. Secondly, [73] shows that the permutation cycle structure of k_1Sk_2, for arbitrary k_1 and k_2, depends only on the difference $\Delta_{k_1k_2} = k_1 \oplus k_2$. Moreover each cycle length appears in the permutation k_1Sk_2 an even number of times. It follows that k_1Sk_2 consists of more than $2^8 \prod_i l_i$ cycles, where $2l_i$ is the number of cycles of the ith permutation of the eight parallel 8-bit permutations which comprise the k_1Sk_2 layer. Using these results it can be shown that, for two randomly chosen points $x = (x_1, x_2, \ldots, x_8)$ and $y = (y_1, y_2, \ldots, y_8)$, the GCD of the cycle sizes has a good chance of being big (unlike a random permutation). The probability that the x_is and y_is will belong to the same cycle or to two different cycles of the same size is more than $\frac{2l_i}{256}$. By collecting and factoring cycle lengths we can obtain information about $\Delta_{k_1k_2}$. Next, [73] shows that,

$$[MSM]k_1Sk_2[MSM] = Ak_1Sk_2A = B \qquad \text{(covering 3.5 rounds)}$$

Because of the involutional structure of Khazad, B is an isomorphic permutation of A, so it has the same cycle structure as discussed earlier. In particular, it has the same cycle structure as k_1Sk_2. Therefore it is enough to study the fixed points of $S(x) \oplus \Delta_{k_1k_2}$. It follows that, for a randomly chosen k_1, k_2, Ak_1Sk_2A has no fixed points with probability greater than $1 - 2^{-8}$. However, if it has a single fixed point then it must have more than 2^8 fixed points. Finally, [73] examines

$$kSk[MSM]kSk[MSM]ksk \qquad 5 \text{ rounds.}$$

It is possible to show that,

$$(KH_5(x) \oplus \Delta_k)^n = k_0SM(KH_3)^nMSk_5$$

where $\Delta_k = k_0 \oplus k_5$ and $KH_j(x)$ means j rounds of Khazad. In other words we have a very strong relationship, due to the involutional structure, between

3 rounds and 5 rounds of Khazad. It follows that if one can detect peculiarities in the cycle structure of 3-round Khazad in less than 2^{64} steps, then this will provide a distinguishing attack on 5-round Khazad faster than exhaustive key search.

The paper by Biryukov and de Cannière [74] compares minimal systems of multivariate polynomials which completely define certain block ciphers, including Khazad. The work is motivated by the recent papers [138] and [375, 377] which propose attacks on block ciphers using overdefined systems of linear, quadratic and low degree equations. For Khazad the P S-box consists of 4 quadratic equations in 16 terms, and the Q S-box consists of 6 equations in 18 terms. These equations are used to define 30 equations in 32 linear and 28 quadratic terms for the 8-bit S-box of Khazad and, along with a set of linear equations to define the linear layers, the whole of the block cipher can be described by 2496 equations in 2048 variables using 3840 linear and quadratic terms. Similarly, the key schedule can be described by 2672 equations in 2638 variables using 4384 linear and quadratic terms. As the cipher takes a 64-bit plaintext input and a 128-bit key, 2 plaintext/ciphertext pairs are required to solve the system, implying a doubling of the number of state equations, variables, and terms for the cipher. However the number of equations for the key schedule remains unchanged, as the same schedule is used for both encryptions. In total [74] estimates that 7664 equations in 6464 variables using 12064 linear and quadratic terms are required. Roughly speaking it is desirable to keep the number of free terms as low as possible so as to maximise the solution speed for such a system (see Sect. 2.2.3.17). For Khazad there are 4400 free terms. It is found that twice as many free terms are required for MISTY1, suggesting that MISTY1 is more secure than Khazad [74]. However, this is perhaps misleading because, as [74] points out, they restrict themselves to quadratic equations for their count whereas S-box S7 of MISTY1 has a much more concise representation using cubic equations. If cubic equations are included in the count then MISTY1 may appear less secure than Khazad.

2.4.3 MISTY1

2.4.3.1 The design

MISTY1 was first published in 1996, is a Feistel network based on a 32-bit nonlinear function, takes 64-bit plaintext and a 128-bit key, and is recommended for 8 rounds (more generally a multiple of 4 rounds). Moreover, each pair of rounds is separated by a layer of two 32-bit FL-blocks. MISTY1 can be implemented in situations where resources are heavily constrained, and the constituent lookup tables are optimised for hardware performance. The entire algorithm is built from recursive components such that at each level the structure is again a secure Feistel-like structure. The recursive design adds a lot of complexity to the cipher, making analysis hard. The additional FL or FL^{-1} functions every odd round, for encryption or decryption respectively, take 32-bit input and output as well as taking a 32-bit subkey as input. The FL layers are added to avoid attacks other than differential and linear cryptanalysis. The modified Feistel structure

uses an FO function which has 32-bit input and output as well as taking a 64-bit subkey and another 48-bit subkey. This FO function contains an FI function which has 16-bit input and output as well as taking a 16-bit subkey. The FI function contains 7×7-bit, and 9×9-bit S-boxes. This encryption structure is shown in Fig. 2.5, where the 64-bit plaintext is first split into two 32-bit left and right parts, and then converted to ciphertext via bitwise XOR, FO and FL/FL^{-1} operations.

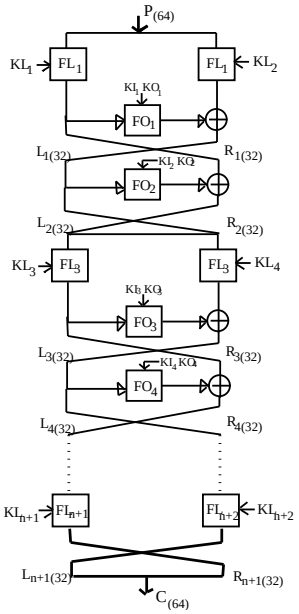

Fig. 2.5: Encryption for MISTY1

The rounds are summarised algebraically as follows:

odd rounds	$R_i = \mathrm{FL}_i(L_{i-1}, KL_i)$	$L_i = \mathrm{FL}_{i+1}(R_{i-1}, KL_{i+1}) \oplus \mathrm{FO}_i(R_i, KO_i, KI_i)$
even rounds	$R_i = L_{i-1}$	$L_i = R_{i-1} \oplus FO_i(R_i, KO_i, KI_i)$

with a final FL operation after the last round, to ensure that decryption is like encryption apart from a reverse of the subkey order and the interchange of FL and FL^{-1}. The KL's, KO's and KI's in the above round expressions are subkeys which are derived from the 128-bit key by using the following key schedule. We first partition the 128-bit key into eight consecutive 16-bit key values $K_1, \ldots, K_8$. We then generate K'_i subkeys by using the FI function as follows,

$$K'_i = FI(K_i, K_{i+1})$$

where the indices are taken cyclically. Then using these subkeys we can derive the subkeys used in the round functions by applying the subkey mapping table of Table 2.3:

Table 2.3. Subkey Mapping Table

Round	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KI_{iL}	KI_{iR}
Actual	K_i	K_{i+2}	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i+3}	$K_{\frac{i+1}{2}}$ odd i $K'_{\frac{i}{2}+2}$ even i	$K'_{\frac{i+1}{2}+6}$ odd i $K_{\frac{i}{2}+4}$ even i

The subkeys of Table 2.3 are then used in the functions FL, FO, and FI, as shown in Figs 2.6, 2.7, and 2.8 (the FL^{-1} function is similar to the FL function).

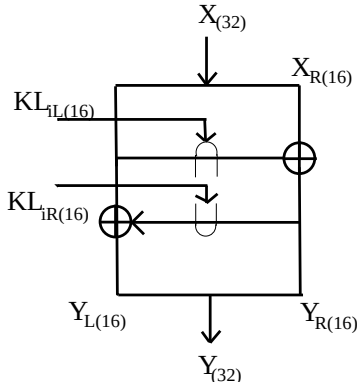

Fig. 2.6: FL function for MISTY1

The input to the FL function comprises a 32-bit data input $X_{(32)}$ and a 32-bit subkey $KL_{i(32)}$. Note that $\cap$ means bitwise AND, and $\cup$ means bitwise OR. The input data is split into two 16-bit halves, $X_{L(16)}$ and $X_{R(16)}$ where,

$$X_{(32)} = X_{L(16)}|X_{R(16)}$$

The subkey is split into two 16-bit subkeys, $KL_{iL(16)}$ and $KL_{iR(16)}$ where,

$$KL_{i(32)} = KL_{iL(16)}|KL_{iR(16)}$$

The FL function is then defined as,

$$Y_{R(16)} = (X_{L(16)} \cap KL_{iL(16)}) \oplus X_{R(16)}$$
$$Y_{L(16)} = (Y_{R(16)} \cup KL_{iR(16)}) \oplus X_{L(16)}$$

where the output is $Y_{(32)} = Y_{L(16)}|Y_{R(16)}$.

The input to the FO function comprises a 32-bit data input $X_{(32)}$ and two sets of subkeys, a 64-bit subkey $KO_{i(64)}$ and a 48-bit subkey $KI_{i(48)}$. The input data is split into two 16-bit halves, L_0 and R_0 where,

$$X_{i(32)} = L_{0(16)}|R_{0(16)}$$

The subkeys are divided into 16-bit subkeys where

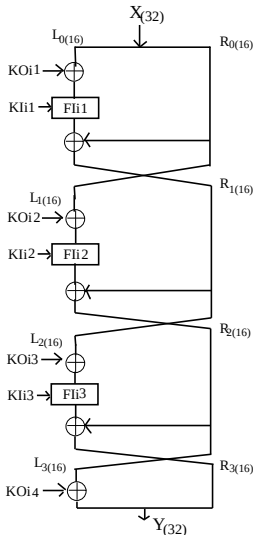

Fig. 2.7: FO function for MISTY1

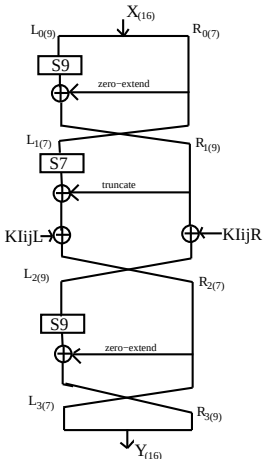

Fig. 2.8: FI function for MISTY1

$$KO_{i(64)} = KO_{i1(16)}|KO_{i2(16)}|KO_{i3(16)}|KO_{i4(16)}$$
$$KI_{i(48)} = KI_{i1(16)}|KI_{i2(16)}|KI_{i3(16)}$$

Then for each integer j with $1 \leq j \leq 3$ we define:

$$R_j = FI_{ij}(L_{j-1} \oplus KO_{ij}, KI_{ij}) \oplus R_{j-1}$$
$$L_j = R_{j-1}$$

Finally, $L_{3(16)}$ is XORed with KO_{i4} and concatenated with $R_{3(16)}$. The function returns $Y_{i(32)} = (L_{3(16)} \oplus KO_{i4})|R_{3(16)}$.

The function FI_j takes a 16-bit data input $X_{j(16)}$ and a 16-bit subkey $KI_{ij(16)}$. The input data is split into two unequal components, a 9-bit left half $L_{0(9)}$ and a 7-bit right half $R_{0(7)}$ where $X_{j(16)} = L_{0(9)}|R_{0(7)}$. The key $KI_{ij(16)}$ is similarly split into two unequal components. FI uses two S-boxes, S7 and S9, mapping 7 bits to 7 bits, and 9 bits to 9 bits respectively. FI also uses two additional functions $ZE()$ and $TR()$. These are defined as follows,

$y_{(9)} = ZE(x_{(7)})$	ZE takes the 7-bit $x_{(7)}$ and converts it to a 9-bit value $y_{(9)}$ by adding two zero bits to the most significant end.
$y_{(7)} = TR(x_9)$	TR takes the 9-bit $x_{(9)}$ and converts it to a 7-bit value $y_{(7)}$ by discarding the two leftmost bits.

Then FI is defined by the following operations.

$$\begin{array}{ll} L_{1(7)} = R_{0(7)} & R_{1(9)} = S9(L_{0(9)}) \oplus ZE(R_{0(7)}) \\ L_{2(9)} = R_{1(9)} \oplus KI_{ijR(9)} & R_{2(7)} = S7(L_{1(7)}) \oplus TR(R_{1(9)}) \oplus KI_{ijL(7)} \\ L_{3(7)} = R_{2(7)} & R_{3(9)} = S9(L_{2(9)}) \oplus ZE(R_{2(7)}) \end{array}$$

Finally, $L_{3(7)}$ and $R_{3(9)}$ are concatenated to give $Y_{(16)} = L_{3(7)}|R_{3(9)}$.

The S-boxes which are contained in the FI function are designed to be efficient for both combinational logic and look-up table implementations, owing to the relatively small numbers of terms in the Algebraic Normal Forms (ANFs) of the constituent functions of the S-boxes. This results in small hardware implementation cost and short delay time.

Both 7 and 9-bit S-boxes are chosen to optimise the *provable security* of MISTY1 against differential and linear cryptanalysis, and both S-boxes achieve the minimum possible differential and linear biases for S-boxes of their size.

2.4.3.2 Security analysis

MISTY1 [355] has been widely studied for five years and no serious security flaws have been found. It should be noted that a variant of MISTY1, namely KASUMI, has been chosen for the 3GPP standard. Therefore many attacks on MISTY1 may also be relevant to KASUMI, and vice versa. Both MISTY1 and KASUMI use nonlinear invertible FL functions to introduce AND and OR operations to the cipher. However, unlike MISTY1, KASUMI is a pure 8-round Feistel cipher, where in the odd rounds we first apply FO then FL, and in the even rounds we first apply FL then FO. Moreover, unlike KASUMI, in MISTY1 after the final swap there is an additional XOR with the subkey on the left-hand side. The S7 and S9 S-boxes of MISTY1 and KASUMI are not identical but are very similar and both exhibit affine relationships between the bit outputs [54] (see Sect. 2.9.1). In particular, for the S-box, S9, of MISTY1, the rotation of the input by any number of bits does *not* affect the least significant bit of the output (see Sect. 2.9.1) — this is, perhaps, a surprising result. It is considered that KASUMI has a weaker key schedule than MISTY1, as the key schedule of MISTY1 is nonlinear whereas that of KASUMI is linear. Further details of the differences between MISTY1 and KASUMI can be found in [176]. There is also a block cipher called MISTY2 by the same designers. MISTY2 is also a 64-bit block cipher using 128-bit key.

This cipher has a newer structure than MISTY1 and recommends the use of 12 rounds as opposed to 8 rounds for MISTY1.

Attacks on MISTY1 without the FL operations have been accomplished up to five rounds. The low algebraic degree of the constituent functions of the MISTY1 S-boxes has invited higher order differential attacks by Lai [323] and Knudsen [294] on MISTY1 without FL functions. A higher order differential attack on MISTY1 without FL functions has also been presented by Tanaka *et al.* [502]. However, it appears that the key action in the FL function can significantly modify the algebraic degree of MISTY1. The Slide attack has been proposed against MISTY1 by Biryukov and Wagner [80] where the same subkey is applied to every nth round, and this is appropriate to MISTY1 because of its simple key schedule. It turns out that without the FL functions, the slide attack works when one of 65536 keys is used. However, MISTY1 maintains some resistance to slide attacks, and this is because one of the design criteria for MISTY1 was resistance to related-key attacks. It has been shown by Biham *et al.* [57] and Knudsen [296] that any Feistel cipher with a bijective round function has impossible differentials in 5 rounds, and MISTY1 without FL layers falls into this category. However, Impossible differential attacks appear to be inappropriate for MISTY1 as the FL functions add extra dependency on the particular key at each round. MISTY1 is designed to have provable security against differential and linear cryptanalysis, and this proof is achieved by bounding the average differential/linear probabilities for the recursive layers of MISTY1; if the average differential/linear probability of each layer is p then the complete cipher has probability upper-bounded by p^4. It is claimed by the designers that the unequal division of the S-boxes into 7 bits and 9 bits has an advantage against differential and linear cryptanalysis, as the probability bound can be made lower for S-boxes that use odd as opposed to even numbers of bits. But there are hardware and software penalties resulting from this asymmetry. Knudsen and Moen have recently applied Integral Cryptanalysis [311, 372] to MISTY1 including FL functions. This includes 4 round and 5 round attacks. The integral attacks exploit the Sakurai-Zheng property that was initially applied to MISTY2. This property is as follows. Let $F(x, y)$ denote the left half and right half of the output after three rounds of MISTY2 on plaintext $< x, y >$. Then $F(x, y) = f(x) \oplus g(y)$ where f and g are key-dependent bijective mappings. Therefore, for any two arbitrary sets of 32-bit values, $\mathbf{S}$ and $\mathbf{T}$, we have, $\sum_{<x,y>\in \mathbf{S}\times\mathbf{T}} F(x, y) = 0$. This is a three-round integral for MISTY2. This property can also be applied to MISTY1. The 5-round attack by Knudsen and Wagner in [311] uses the Sakurai-Zheng property once, and the 4-round attack uses the property twice. It requires a data complexity of 2^{34} and a time/memory complexity of 2^{48}. Also a new attack, the Slicing Attack by Kuhn [319, 318] has been applied to the 4-round version of MISTY1, making use of the special structure and position of the key-dependent linear FL functions. These FL functions present a subtle weakness in the 4-round version of the cipher. Both this attack and that of [311] are particularly interesting as they include the FL layers, unlike many other attacks which ignore the FL layer.

Generalised linear characteristics through both the 7-bit and 9-bit S-boxes of MISTY1 with maximum and minimum bias are searched for in [426]. It is found

that, although both S-Boxes have an optimally low bias relative to the Walsh-Hadamard Transform (WHT), with PAR = 2.0, the bias increases significantly with respect to many generalised linear approximations, in particular those covered by the **HI** transform, where the PAR = 16.0 and 32.0 for 7 and 9-bit S-boxes respectively. This suggests that, whereas the odd-length S-box width (7 and 9) minimises the possible linear characteristic with respect to the WHT, the odd-length restriction in fact weakens the S-box with respect to certain generalised linear approximations, to give a PAR of $2^{\lceil \frac{n}{2} \rceil}$, where $n = 7$ or 9. This is a counter-argument to the argument proposed by the designer for using odd-length S-boxes — more general linear approximations suggest that even-length S-boxes may be better (although such generalised linear approximations have not yet led to an attack).

As discussed in the security analysis for Khazad, Biryukov and De Cannière [74] compare minimal systems of multivariate polynomials which completely define certain block ciphers, including MISTY1. For MISTY1 S9 is designed as a system of 9 quadratic equations in 54 terms. S7 is designed as a system of 7 cubic equations in 65 terms, but can also be defined by 11 quadratic equations in 93 terms. The quadratic representations are utilised and, along with a set of linear equations to define the linear layers, the whole of the block cipher can be described by 1824 equations in 1664 variables using 5848 linear and quadratic terms [74]. Similarly, the key schedule can be described by 432 equations in 528 variables using 1848 linear and quadratic terms. Two plaintext/ciphertext pairs are required to solve the system, implying a doubling of the cipher count, but not for the key schedule. In total [74] estimates that 4080 equations in 3856 variables using 13544 linear and quadratic terms are required. This gives the number of free terms as the number of terms minus the number of equations (see Sect. 2.2.3.17). For MISTY1 this is 9464 free terms. When compared with Khazad this seems large, but the count may decrease significantly if cubic representations are considered for S7 of MISTY1.

The main advantage of MISTY1 is its provable security against differential and linear cryptanalysis.

2.4.4 SAFER$_{++64}$

2.4.4.1 The design

SAFER$_{++64}$ [353] is a development from the existing SAFER family of ciphers and uses a combination of substitution and linear transformation to achieve confusion and diffusion, respectively. Recently, the 128-bit plaintext version has been adopted for use in the authentication scheme in Bluetooth, the wireless communication protocol. We consider here the legacy version which takes in 64-bit plaintext and 128-bit key, and outputs 64-bit ciphertext. The designers recommend this version be used with 8 rounds to achieve sufficient security. SAFER+ was a submission to the AES, and it is claimed that SAFER$_{++}$ is simpler and faster than SAFER+, and at least as strong, cryptographically. The main new feature in SAFER$_{++}$, as compared to all earlier versions of SAFER (including

SAFER+), is the use of a *4-point* Pseudo-Hadamard Transform (PHT), instead of the *2-point* PHT. SAFER$_{++}$ uses the 4-point PHT to achieve fast, rapid diffusion at low complexity. One 16-byte subkey is used with each round, along with one post-cipher *output transformation* which is a final 8-byte subkey *addition*. Another aspect of the cipher is its use of two incompatible group additions to achieve key addition, namely bitwise XOR (mod 2), and bytewise addition, mod 256. Moreover, the S-boxes are exponential and logarithmic functions, mod 257, which are combined with the two addition operations in such a way as to thwart potential homomorphisms. One round of the encryption is shown in Fig. 2.9, and an alternative view of the encryption round is given in Fig. 2.10. The decryption round is similar; however even if one does not consider the key schedule, decryption is not identical to encryption for SAFER$_{++64}$.

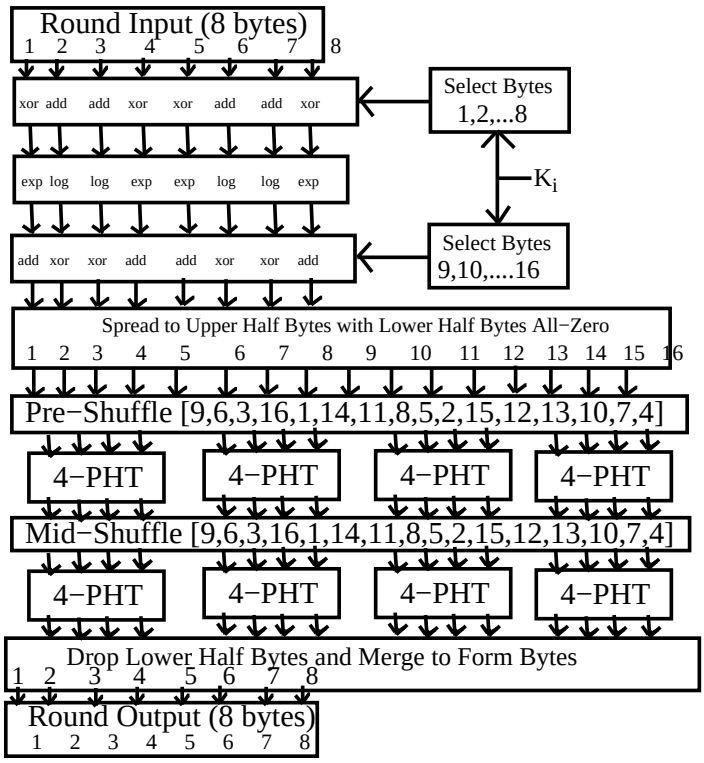

Fig. 2.9: Encryption Round for SAFER$_{++64}$

Key addition for a round is bytewise and uses two different (incompatible) group operations, where subkey bytes 1,4,5,8 are added using bitwise XOR, and subkey bytes 2,3,6,7 are added using bytewise addition, mod 256. After 8 rounds of Fig. 2.9 there is a final 8-byte key addition whose operations are identical to the first addition in each round. The decryption process follows the same structure

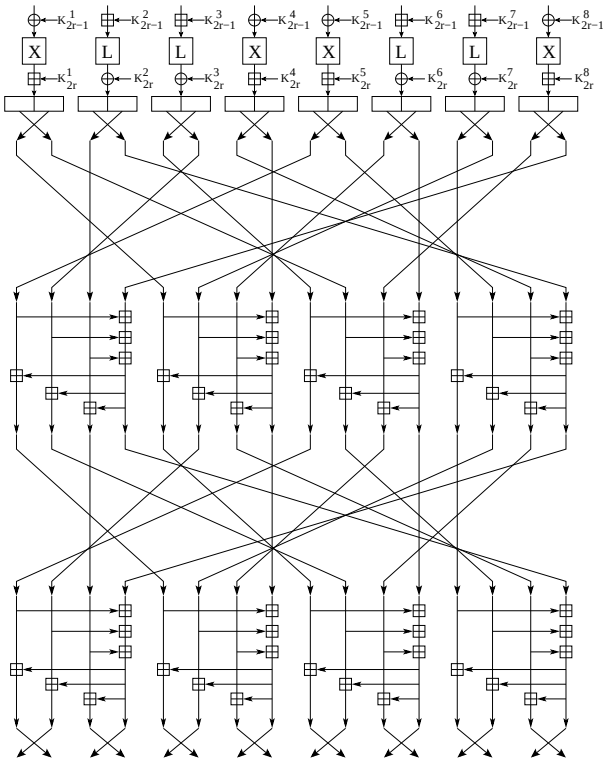

Fig. 2.10: Alternative View of Encryption Round for SAFER$_{++64}$

as encryption with the round keys used in reverse order, the PHT replaced with the inverse PHT, the shuffle replaced with the inverse shuffle, addition replaced with subtraction, and exp (log) replaced with log (exp). The 4-PHT matrix, H_4, which is used to achieve linear diffusion is as follows,

$$H_4 = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \mod 256$$

where each element of the matrix represents an 8-bit byte, and H_4 acts on a vector with 8-bit byte entries, mod 256. This PHT matrix can be implemented efficiently, as H_4 is a matrix of low weight. The shuffle permutation has also been chosen to maximise diffusion in conjunction with the PHT. H_4 is not an involution and the inverse of H_4, which is used for decryption to implement the inverse PHT, is as follows,

$$H_4 = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 4 \end{pmatrix} \mod 256$$

H_4^{-1} is also easy to implement, as it also has low weight. The advantage of the PHT method of diffusion is that a linear diffusion matrix, M, of dimensions 16×16 bytes is implemented by means of smaller, simpler 4×4 byte 4-PHT matrices. Moreover, the designers argue that because each row of M contains at least ten 1's, the diffusion of the cipher is wide and fast.

The nonlinear layer of the cipher is achieved by means of two S-boxes, exp and log. exp realises the function $y = 45^x \mod 257$ for bytes 1,4,5,8, and log realises the function $y = \log_{45}(x) \mod 257$ for bytes 2,3,6,7, where $\log_{45}(0) = 128$ by convention. The exp operation is juxtaposed with XOR, and these two operations do not admit a homomorphism, explicity $45^{x_1 \oplus x_2} \neq 45^{x_1} 45^{x_2}$. This incompatibility between exp and XOR enhances the security of the system. A similar incompatibility exists between log and add, which are also juxtaposed in the cipher. One reason for choosing S-boxes based on exp and log is that their associated ANF (Algebraic Normal Form - i.e. boolean expression) descriptions relating input and output look random, are of high degree, and contain many terms. This is a different philosophy than that used to design the S-boxes for MISTY1, where the ANF functions contain relatively few terms so as to simplify implementation complexity.

The key schedule for SAFER$_{++64}$ uses 9 16-byte bias words in order to randomize the produced subkeys so as to help to avoid weak keys. These bias words, B_j, are determined by,

$$B_{i,j} = 45^{(45^{17i+j} \mod 257)} \mod 257$$

where $B_{i,j}$ is the ith byte of B_j.

The odd-index 16-byte subkeys are generated using the method outlined in Fig. 2.11, and given in detail in Fig. 2.12. A similar strategy holds for the even-index subkeys.

For SAFER$_{++64}$, only subkeys $K_1, \ldots, K_9$ are used. Another interesting aspect of SAFER$_{++}$ is that it is possible to develop mini-versions of the cipher with, e.g. 4-bit nibble wordlengths, which still retain the main features of the cipher, and this enables the thorough testing of different attack strategies before extrapolating them to the larger, real cipher. Also one should note that the spreading of 64 bits to 128 bits, and the subsequent dropping of 128 bits down to 64 bits is an unusual feature of SAFER$_{++}$ which distinguishes it from many other block ciphers.

2.4.4.2 Security analysis

No security flaws have been found with SAFER$_{++64}$ [353] and it has many similarities to SAFER$_{++128}$. One weakness found in previous versions of the SAFER family by Knudsen was in the key-schedules [295, 298], but these weaknesses have been dealt with in SAFER$_{++64}$. Other pre-NESSIE attacks on the SAFER family include truncated differentials by Knudsen and Berson [304], and Murphy [374] identifies a potential algebraic weakness regarding the existence of invariant $\mathbb{Z}$-modules within the PHT layer. These modules and their cosets are not diffused by the PHT layer, and so provide a way to cope with diffusion in SAFER, regardless of the key schedule. One attack in [374] which used this property enabled a

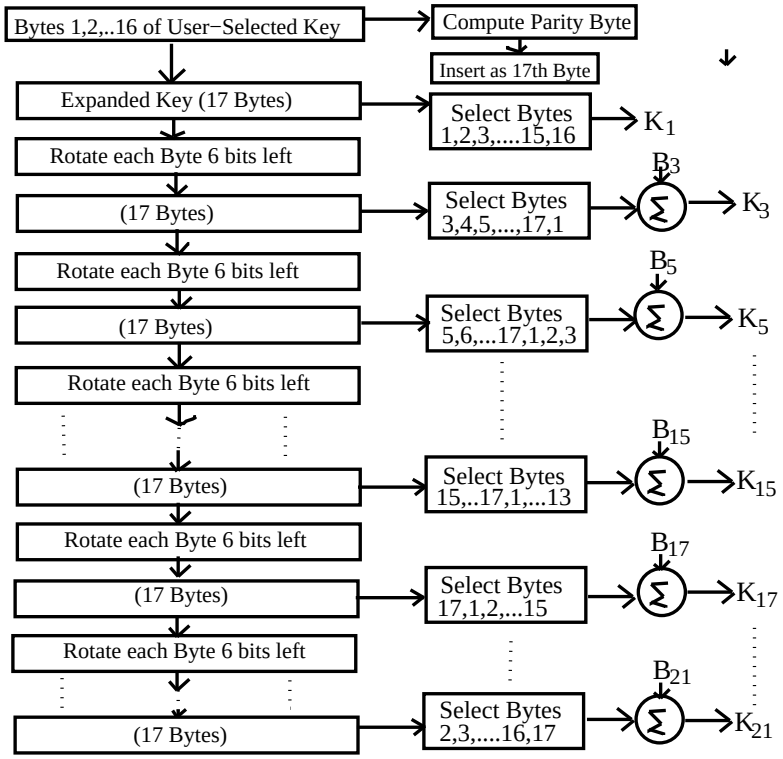

Fig. 2.11: Generation of Odd-Index Subkeys in the SAFER$_{++}$ Key Schedules. $\sum$ denotes bytewise summation, mod 256

projection of the message/ciphertext space onto a 4-byte $\mathbb{Z}$-submodule so that the probability of any message projection giving any ciphertext projection is independent of $\frac{1}{4}$ of the key bytes. This result, along with the results of [295] led to a change in the SAFER key schedule. The designers recommend 8 rounds to ensure security for the cipher and not fewer than 7 rounds, and they claim that one of the main reasons for the security of the cipher against differential and linear cryptanalysis is the high diffusion PHT layer. The designers conclude that SAFER$_{++}$ with six or more rounds is secure against differential cryptanalysis, and with two and a half or more rounds is secure against linear cryptanalysis. However, recently Nakahara *et al.* [382] applied techniques that were first used to more generally attack the SAFER family in [381], and [382] showed that three and a half rounds of SAFER$_{++64}$ can be attacked requiring 2^{33} known plaintexts. The reason for this discrepancy between the two and a half rounds claimed by the designers and the three to four rounds claimed by [382] is largely because the designers restricted themselves to homomorphic attacks, whereas [382] applies strictly non-homomorphic attacks where some key bits are assumed fixed. Therefore the results of [382] must be seen in the context of a weak-key class. Table 2.4 gives a summary of the complexity of linear attacks on SAFER$_{++}$ [382].

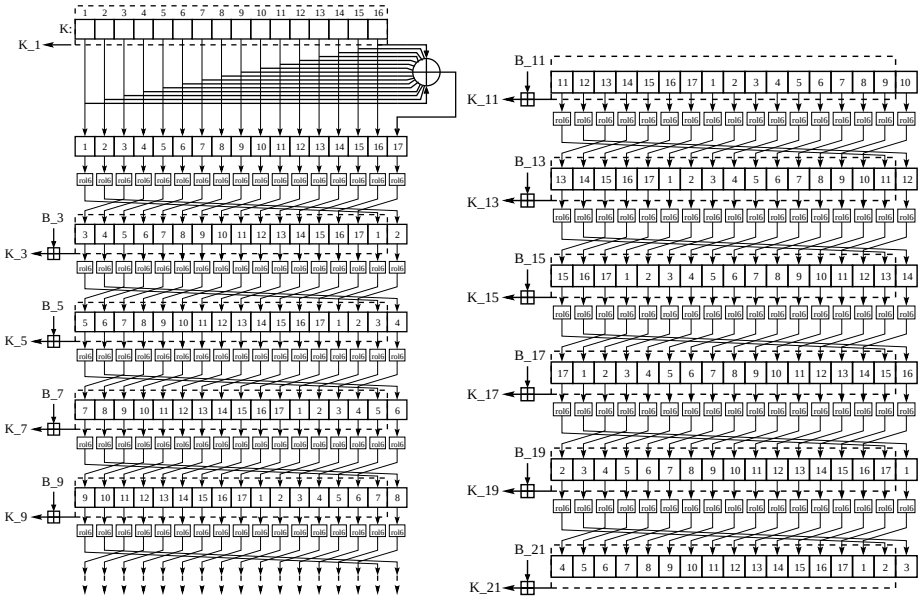

Fig. 2.12: Detailed Generation of Odd-Index Subkeys in the SAFER$_{++}$ Key Schedules

Table 2.4. Complexity of Linear Attacks on SAFER$_{++}$

# Rounds Attacked	Linear Relation	# Known Plaintexts	# Subkey Bits Explored	Attack Complexity	Fraction of Keys
2	(1)	2^5	37	2^{42} ♣	
3.5	(3)	2^{33}	88	2^{121} ♣	2^{-6}

♣ The attack applies to all key sizes defined for SAFER$_{++}$.

One important point to note with regard to the linear attack of [382] is that it identifies a surprisingly small byte and bit branch number for the PHT diffusion layer. Whereas the designers of SAFER$_{++}$ use the fact that the lowest row weight of the matrix M is 10, implying a high diffusion, a more detailed examination of the matrix reveals a byte branch number ≤ 7 and a bit branch number ≤ 5. Moreover the S-box defined by $y = 45^x \mod 257$ contains a linear relationship which holds with probability 1. In other words, we can write $y_0 = f(x_0, x_1, \ldots, x_6) + x_7$, where y_0 is one of the output bits of the S-box, and depends linearly on the input, x_7. This moderated diffusion combined with a high linear characteristic is what enables this linear attack on three to four rounds of SAFER$_{++}$.

It is also interesting to note that affine relationships exist between the bit outputs of the S-box, both for exp45 and log45 [54] (see Sect. 2.9.1).

2.4.5 Triple-DES

2.4.5.1 The design

One variant of Triple-DES which is in widespread use is called two-key Triple-DES. It takes a 64-bit plaintext and a $2 \times 56 = 112$-bit key. It occurs as a natural extension of the existing standard DES, where security has been increased, in particular the key input size, by repeating the cipher three times and the key twice. Note that double-DES is not an option due to a meet-in-the-middle attack which renders double-DES with no greater security than single DES. To allow for backwards-compatibility, two-key Triple-DES was suggested in the form encrypt-decrypt-encrypt although, in fact, the form encrypt-encrypt-encrypt can also revert to single DES encryption by using the all-zero key in the first two encryptions, and a single-DES key in the third encryption (the all-zero key being self-dual). There is also three-key Triple-DES (3DES), which takes 64-bit plaintexts and a 168-bit key, and this is also in widespread use. It is considered to be a lot more secure than two-key Triple-DES and can also be made backwards compatible with DES by making all three keys the same in encrypt-decrypt-encrypt mode. One should also mention DESX which also takes three keys but is relatively efficient compared to Triple-DES, requiring only a single DES encryption preceded by XOR with another key, and completed by XOR with a third key.

A round of DES is summarised in Fig. 2.13. DES is a Feistel cipher. L and R are the left and right splittings of the 64-bit plaintext, and C and D are the left and right splittings of the 56-bit key. The key is input linearly via XOR and there are 8 6-bit in, 4-bit out S-boxes which are applied in parallel to the 48-bit input to give 32-bit output.

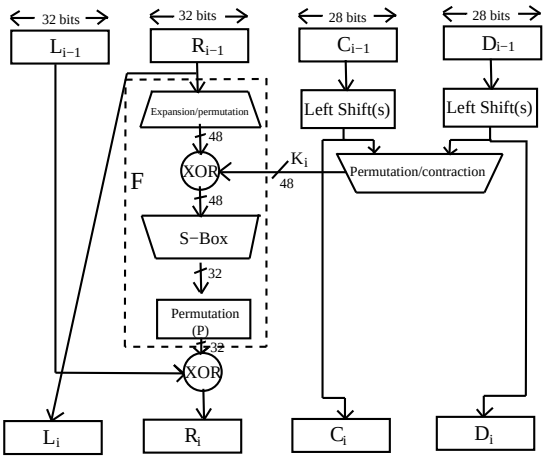

Fig. 2.13: A Round of DES

Essentially the DES standard recommends 16 rounds, and three-key Triple DES is simply a concatenation of three instances of DES, where a different key is input for each instance of DES to give a total key input of $3 \times 56 = 168$ bits.

2.4.5.2 Security analysis

The security of Triple-DES is significantly less than the 128 bits required for this cipher category and its performance on workstations is rather bad. However Triple-DES will still be considered as a benchmark for other algorithms. It has been shown by Merkle and Hellman in [367] that two-key triple encryption can be broken using 2^{56} chosen plaintexts, and 2^{112} single encryptions. The standard way to attack triple-DES is to use the meet-in-the-middle attack [366], requiring 3 plaintext/ciphertext pairs, and 2^{112} single encryptions. Advanced meet-in-the-middle attacks for two-key triple encryption have been proposed by Van Oorschot and Wiener in [506]. Two-key Triple-DES is generally considered weaker than three-key Triple-DES.

Clearly, any attack on DES is relevant to an attack on Triple-DES and in fact the best attacks on reduced variants of Triple DES are the attacks on DES. A well-known weakness of DES is the *complementation property*. Specifically, let p and k be plaintext and key inputs to DES, respectively. Then the complementation property is summarised as follows:

$$\mathrm{DES}_k(p) = \overline{\mathrm{DES}_{\overline{k}}(\overline{p})}$$

where $\overline{*}$ denotes the complement of the bit-string, $*$. However, this evident deviation from a random cipher does not comprise the cipher to any great extent. It has been shown that differential cryptanalysis can cover as many as 18 rounds of DES, and it is suspected that this may also be so for linear cryptanalysis. Moreover, due to the no-swap between rounds 16 and 17, it may be possible to gain one more round. It is also interesting to note that affine relationships exist between the bit outputs of the S-box, for the first seven S-boxes of DES [54] (see Sect. 2.9.1).

Kelsey *et al.* used related-key techniques [288] to attack three-key triple-DES, and Biham [52] encrypts the same plaintext 2^{28} times using Triple-DES under 2^{28} different keys, allowing the attacker to recover one of the 2^{28} keys using 2^{84} steps and 2^{84} single encryptions. In [340] a more efficient meet-in-the-middle attack is presented by Lucks which can break three-key triple DES with about 1.3×2^{104} single encryption steps, and 2^{32} known plaintext/ciphertext pairs. The attack works by saving single encryption steps and exploiting known and/or chosen plaintext/ciphertext pairs, the complementation property weakness of DES and a certain number of weak keys.

2.5 128-bit block ciphers considered during Phase II

The 128-bit block ciphers selected for phase II of NESSIE were Camellia, RC6, and SAFER$_{++128}$. We also continued to study the AES, in order to compare the

submitted ciphers with a current standard. None of the ciphers considered have been broken so the following security evaluation identifies weaknesses that occur in reduced-round versions of the ciphers, and weaknesses that may lead to more effective attacks in the future. We first describe each cipher in some detail, along with the most important attacks known on each cipher. Note that the algorithms given here are not complete specifications, but references are given to complete specifications which may be found on the NESSIE website. After discussing each cipher we summarise and compare some of the distinguishing features of the ciphers, identifying potential weaknesses and noting the best-known attacks, as shown in Tables 2.16 and 2.17 of Sect. 2.9.2.

2.5.1 Camellia

2.5.1.1 The design

Camellia [19] is an 18-round 128-bit block cipher which supports 128-, 192-, and 256-bit key lengths, with two layers of two 64-bit FL-blocks after the 6th and 12th rounds. It is a byte-oriented 18-round Feistel cipher with a particular emphasis on low-cost hardware applications, and is designed to be resistant to differential and linear cryptanalysis with linear bias and differential probabilities both $\leq 2^{-128}$. Camellia uses four 8×8-bit S-boxes with input and output affine transformations and logical operations. However there is no 32-bit integer addition, so as to avoid the possibility of a long critical path (longest inherent sequential computation) due to the carry propagation associated with addition. The diffusion layer uses a linear transformation based on a Maximum-Distance-Separable code with a branch number of 5 (activity on t input bytes to the linear transformation layer will diffuse to activity on at least $5-t$ output bytes from the linear transformation layer). Camellia also uses FL and FL^{-1} functions which are inserted every 6 rounds so as to enhance irregularity of the cipher. The FL functions are similar to those of MISTY, except that Camellia also uses a 1-bit rotation so as to make bytewise cryptanalysis harder. The entire structure of Camellia is given in Fig. 2.14. The FL and FL^{-1} functions are shown in Fig. 2.15.

The design of Camellia is based on E2 which was a previous block cipher by the same designers and was a submission to the AES. The main difference between E2 and Camellia is the adoption, for Camellia, of the 1-round SPN, not the 2-round SPN of E2, leading to an expected improvement in speed for Camellia. The design of the F-function of Camellia follows that of E2. The F-function transforms a 64-bit input, X_{64}, to a 64-bit output, Y_{64}, using a 64-bit subkey k_{64}, given by $Y_{64} = F(X_{64}, k_{64})$. More specifically, the F function is described by,

$$F : \mathbf{L} \times \mathbf{L} \rightarrow \mathbf{L}$$
$$(X_{64}, k_{64}) \rightarrow Y_{64} = P(S(X_{64} \oplus k_{64}))$$

where $\mathbf{L}$ denotes a vector-space of 64 bits, P is a byte-linear transformation, and S operates in parallel on 8-bit segments of the 64-bit input, with each 8-bit segment being subject to an 8×8 S-box transformation (one of 4 S-boxes, so that S is given by $s_1, s_2, s_3, s_4, s_2, s_3, s_4, s_1$). Each of the 8×8-bit S-boxes, s_1, s_2, s_3, s_4 is affine equivalent to x^{-1} over $\mathrm{GF}(2^8)$ — which is similar to the Rijndael design.

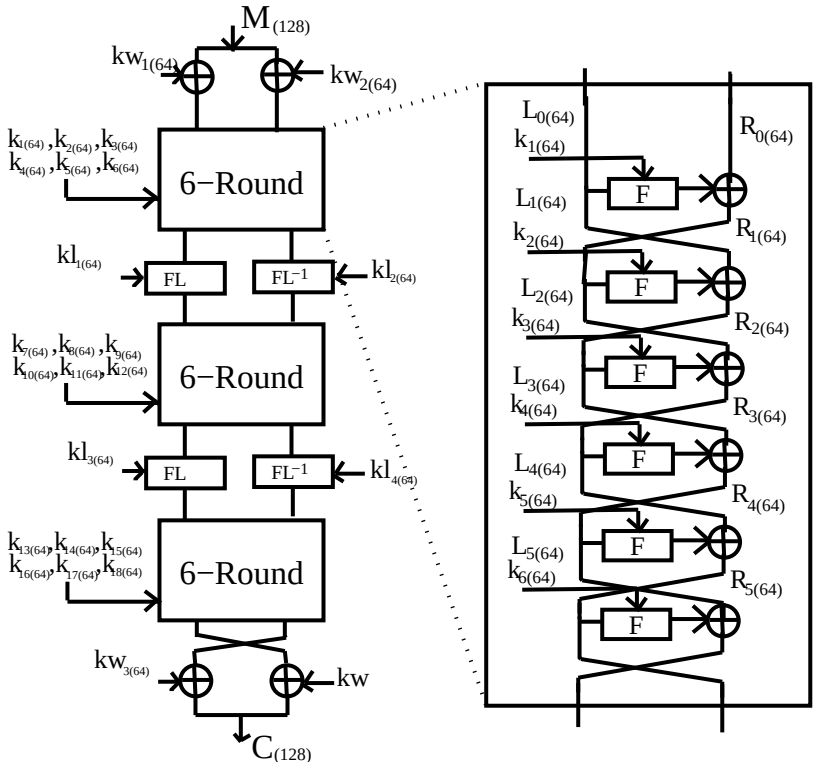

Fig. 2.14: Camellia

The P transformation is designed to have an optimal branch number and can be represented as follows,

$$\begin{pmatrix} z_8 \\ z_7 \\ \dots \\ z_1 \end{pmatrix} \rightarrow \begin{pmatrix} z'_8 \\ z'_7 \\ \dots \\ z'_1 \end{pmatrix} = P \begin{pmatrix} z_8 \\ z_7 \\ \dots \\ z_1 \end{pmatrix}$$

where,

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The key schedule of Camellia makes use of the F-function of the encryption module, and is the same for encryption and decryption. The user key is encrypted

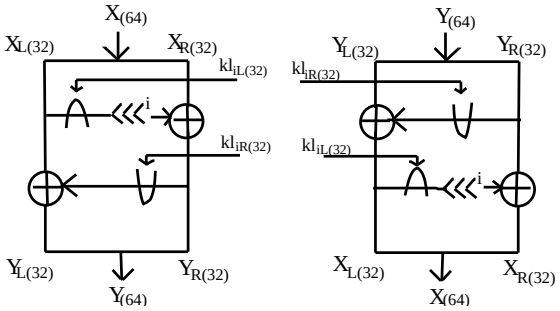

Fig. 2.15: FL and FL^{-1} Functions for Camellia

by means of the F-function using pre-fixed constants, where these constants, $\sum_i$, are defined as continuous values from the hexadecimal representation of the square root of the ith prime. The subkeys are then generated partly from rotated values of the user-input key, K (where $K = K_{L(128)}$, $K = K_{L(128)}||K_{RL(64)}$, or $K = K_{L(128)}||K_{R(128)}$, for a 128-bit, 192-bit, or 256-bit key, K, respectively), and partly from rotated values of the encrypted keys, K_A and K_B. Fig. 2.16 shows how to generate these encrypted keys.

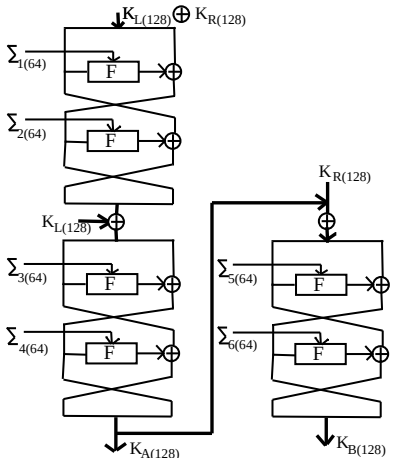

Fig. 2.16: Key Schedule for Camellia

We refer the reader to the documentation of Camellia [19] for a more detailed description.

2.5.1.2 Security analysis

No security flaws have been found for Camellia and it has an interesting design. The designers [20] claim that for Camellia no differential/linear characteristics exist with linear bias and differential probabilities $> 2^{-128}$ over 12 rounds and 2^{-132}

over 15 rounds. This claim is due to the use of four S-boxes which are affine transformations of x^{-1} over GF(2^8). This particular mathematical function ensures optimal differential and linear characteristics through the S-box (irrespective of the affine transformation) of 2^{-6}. Another reason for the strong resistance of the cipher to differential and linear cryptanalysis is the use of a linear transformation for the diffusion layer with a high branch number of 5. The affine transformations which modify the input and output to the S-boxes are designed to provide resistance to interpolation attacks by making each S-box less algebraic in character. The designers also claim that 10 rounds is indistinguishable from a random permutation with respect to truncated differential and linear cryptanalysis. As with all Feistel ciphers with bijective F-function, an impossible differential attack exists over 5 rounds of Camellia [296], but the designers have not found any other attacks of this type. The designers also claim security against interpolation, linear sum, and Square attacks. Iterated ciphers with identical rounds are susceptible to the Slide attack, and this is one reason why the FL functions are inserted in the cipher every 6 rounds, so as to introduce irregularity and resistance to Slide attacks.

The security of Camellia against the Square attack is discussed by Yeom *et al.* in [523]. A 4-round distinguisher allows for a Square attack (see Sect. 2.2.3.15), and four-round Camellia can be attacked by guessing a one byte subkey and using 2^{16} chosen plaintexts. This attack may be extended up to 9 rounds including the first FL/FL^{-1} layer by considering the key schedule. In [484] Shirai *et al.* discuss the security of Camellia against differential and linear attack, and the security is evaluated against the upper bounds of maximum differential characteristic probability (MDCP) and maximum linear characteristic probability (MLCP), calculated by determining the least numbers of active S-boxes, found by search. An evaluation method for truncated differential and linear paths is used to discard wrong paths. Using the above techniques, tighter upper bounds on MDCP and MLCP were found for reduced-round Camellia. Consequently, 10-round Camellia without FL/FL^{-1} has no differential and linear characteristic with probability higher than 2^{-128}. Shirai developed these attacks further in [483], proposing differential and linear attacks on Camellia without FL/FL^{-1} layers, and boomerang and rectangle attacks on Camellia with FL/FL^{-1} layers. The search complexity for the attacks of [483] is reduced by distinguishing between dependent and independent variables in the multi-round characteristics. Shirai obtains a differential attack on 11 rounds without FL/FL^{-1} layers using an 8-round characteristic, and a linear attack on 12 rounds without FL/FL^{-1} layers using a 9-round linear approximation. The boomerang and rectangle attacks are on 10-round Camellia with FL/FL^{-1} layers, and use a technique developed by Biham *et al.* [60, 61], where a cipher is described as $E_f \circ E_1 \circ E_0 \circ E_b$, such that the FL/FL^{-1} layer occurs between E_1 and E_0. Truncated and impossible differential cryptanalysis of Camellia (without FL/FL^{-1} functions) is described by Sugita *et al.* in [501], improving on the best known truncated and impossible differential cryptanalysis. A 9-round bytewise characteristic is shown that may lead to an attack on reduced-round Camellia without FL/FL^{-1} in a chosen plaintext scenario. A 7-round impossible differential is also shown by [501] on Camellia without FL/FL^{-1}

functions. However, the designers of Camellia suspect that the FL and FL^{-1} functions will make attacking Camellia using impossible differentials difficult, since the functions change differential paths depending on key values. A Square attack on Camellia is proposed by He and Qing in [248], requiring 2^{112} encryptions and 13×2^8 plaintexts, over 6 rounds. The designers propose an 18-round cipher, but claim that even a 10-round variant is secure. However, they do not describe attacks on reduced variants of Camellia. A 3-round iterative differential characteristic with probability 2^{-52} has been found by Biham *et al.* in [59], which can be iterated to further rounds. They also found 5 additional 7-round characteristics with probability 2^{-104}. A 9-round variant of Camellia (without FL layers) was attacked using 2^{105} chosen plaintexts. Also, a one-round truncated iterative differential was found which over 7 rounds has probability 2^{-112} (assuming no FL layer). This can be extended to 8 rounds with probability 2^{-112}. This differential has the added advantage that it passes through the FL/FL^{-1} layers with probability 2^{-8}. Linear cryptanalysis of Camellia did not produce any efficient results — the best linear approximation of the S-boxes being $\frac{1}{2} \pm \frac{1}{16}$. A higher-order differential attack on 10 rounds of 256-bit key Camellia without FL rounds is performed by Kawabata and Kaneko in [284]. This also leads to an attack on 9 rounds for a 192-bit key and an attack on 8 rounds for a 128-bit key.

A recent embedding by Murphy and Robshaw of the AES, Rijndael, in a larger block cipher, the Big Encryption System (BES) [375, 377] has led to questions regarding future potential attacks on the AES. Camellia uses the same x^{-1} function for the S-box as Rijndael, and hence is open to the same form of attack. However, Camellia also inserts FL and FL^{-1} layers every six rounds, and an initial estimate of the extra complexity needed to overcome these layers for a BES-style redescription is about 2^{16}. In short, if an attack using BES and a system of overdefined quadratic equations was ever successful on AES, then it might also be quite successful on Camellia. BES is discussed further in the section on Rijndael-128 of this report. As discussed in the security analyses for Khazad and MISTY1, Biryukov and De Cannière [74] compare minimal systems of multivariate polynomials which completely define certain block ciphers, including Camellia. As pointed out in [138], the Camellia (and Rijndael) S-box can be described by a system of 23 quadratic equations in 80 terms. Quadratic representations are utilised in [74] and, along with a set of linear equations to define the linear layers, the whole of the block cipher can be described by 5104 equations in 2816 variables using 14592 linear and quadratic terms. Similarly, the key schedule can be described by 1120 equations in 768 variables using 3328 linear and quadratic terms. In total [74] estimates that 6224 equations in 3584 variables using 17920 linear and quadratic terms are required. This gives the number of free terms as the number of terms minus the number of equations (See Sect. 2.2.3.17). For Camellia this is 11696 free terms. Roughly the same figures occur for Rijndael.

Finally, Fuller and Millan [208] recently observed that the bit-output functions of the S-box which is x^{-1} over $GF(2^8)$ are all affine transformations of the same function. This observation was given in relation to the Rijndael S-box, but Camellia uses essentially the same S-box. This observation of [208] suggests a

potential extra hardware saving for the Camellia S-box, although this may later also be seen as a security weakness (see Sect. 2.9.2).

2.5.2 RC6

2.5.2.1 The design

We consider here the version of RC6 [273] which takes 128-bit plaintext and key-lengths from 0 to 256 bytes, although the most useful variants may be versions with 16, 24, or 32-byte keys. 20 rounds are recommended. RC6 is quite a simple cipher to describe and it is a natural progression from the cipher RC5 which has undergone cryptanalysis for around 8 years without serious breaks or major security weaknesses being identified (although it is interesting to note that a key was recently found for 64-bit RC5 using a distributed internet exhaustive key search (see Sect. 2.2.3.1) involving over 300000 people over 5 years — the aim of a distributed internet search is to partition the exhaustive key search problem into many small subproblems which are then solved independently by participants in the search). However, RC5 was broken in a series of three papers which each improved the previous paper's result by a factor of about 1000 [305, 283, 76]. These attacks are generally based on the fact that the *rotation amounts* in RC5 do not depend on all the bits in a register. These attacks led to a redesign of RC5 into RC6 prior to submission of RC6 to the AES. The designers [273] claim that RC6 is an improvement on RC5, because of the introduction of fixed rotations and an extra quadratic function to enhance diffusion and resistance to differential and linear cryptanalysis. The key schedule is inherited from RC5, is quite involved and is considered strong. The cipher is fully-parameterised so that mini-versions (e.g. 4-bit, 8-bit, 16-bit) can be implemented and analysed, and this helps for the security analysis of the full 128-bit cipher. For a 128-bit block size, it is recommended to use a word size of $w = 32$ bits, and $r = 20$ rounds. RC6 uses a 32-bit multiplication, mod 2^w, to enhance security, and therefore benefits greatly from software/hardware implementations with optimised multiplication. A round in RC6 is a bit like a round in DES, where half of the data is updated by the other half, and then the two halves are swapped. In fact, [474] reconfigures RC6 as a Feistel-like cipher by swapping the B and C registers. The full encryption procedure is as follows:

Input: Plaintext in A, B, C, D
Output: Ciphertext in A, B, C, D
Key: $S[0, 1, \ldots, 2r + 3]$, r rounds
Procedure:

$B = B + S[0]$	$+$ is addition mod 2^w
$D = D + S[1]$	
for $i = 1$ to r do	
{	
$t = (B \times (2B + 1)) <<< \log_2(w)$	$\times$ is mult. mod 2^w
$u = (D \times (2D + 1)) <<< \log_2(w)$	$<<<$ is rotate left
$A = ((A \oplus t) <<< u) + S[2i]$	$\oplus$ is bitwise addition mod 2

$C = ((C \oplus u) <<< t) + S[2i + 1]$
$(A, B, C, D) = (B, C, D, A)$
}
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$

Decryption is similar to encryption, but not the same — for instance addition is replaced with subtraction, and rotate right operations are used.

Two constants are added into the key schedule. The constants are P_w and Q_w, which are binary expansions of $e - 2$ and $\phi - 1$, respectively, where e is the natural logarithm, and ϕ is the Golden Ratio. The key schedule is as follows:

Input: User-supplied b byte key preloaded into the c-word
Array $L[0, \ldots, c - 1]$
Number r of rounds
Output: w-bit round keys $S[0, \ldots, 2r + 3]$
Procedure:
$S[0] = P_w$
for $i = 1$ to $2r + 3$ do
$S[i] = S[i - 1] + Q_w$
$A = B = i = j = 0$
$v = 3 \times \max\{c, 2r + 4\}$
for $s = 1$ to v do
{
$A = S[i] = (S[i] + A + B) <<< 3$
$B = L[j] = (L[j] + A + B) <<< (A + B)$
$i = (i + 1) \mod (2r + 4)$
$j = (j + 1) \mod c$
}

2.5.2.2 Security analysis

No security flaws have been found for RC6 and it has resisted cryptanalysis during and after the AES process. It is specified over 20 rounds although the fact that 15 of the 20 rounds were broken meant that NIST did not consider that 20 rounds gave a sufficient security margin.

The designers performed extensive differential and linear cryptanalysis of RC6 in [121] and conclude that RC6 is highly resistant to both attacks. Table 2.5 shows non-exhaustive estimates for the numbers of plaintexts necessary to attack RC6 as quoted in [121], although some of the figures on the right-hand side of the table exceed 2^{128} which is the total number of possible plaintexts.

The two most obvious notions of difference for RC6 are XOR and subtraction, mod 2^w, and [121] claims that difference using subtraction gives a more effective attack. An aim of this type of attack will be to try to use differences that do not provide different rotation amounts, thereby minimising the avalanche effect. The purpose of the introduction of the quadratic function $f(x) = x(2x + 1)$ in RC6,

Table 2.5. Differential/Linear Cryptanalysis of RC6 [121]

	#Rnds				
attack	8	12	16	20	24
diff. crypt.	2^{56}	2^{117}	2^{190}	2^{238}	2^{299}
lin. crypt.	2^{47}	2^{83}	2^{119}	2^{155}	2^{191}

a feature not in RC5, is to increase dependency of the data-dependent rotations, and to speed-up diffusion. Both these effects improve the resistance of the cipher to differential and linear cryptanalysis. The most manageable differential and linear attacks use one-bit characteristics as, although multiple-bit characteristics do exist with higher probability, it is difficult to connect them up into a differential or linear trail. The most effective type of linear approximation appears to exploit approximation across the data-dependent rotations. The paper of [121] identifies such approximations of the form $A \cdot \Gamma_a = B \cdot \Gamma_b \oplus C \cdot \Gamma_c$ for the data-dependent rotation, $A = B <<< C$, where single-bit masks work best across the integer addition and the quadratic functions. Moreover, the best approximation across $y = (f(x) <<< 5)$ is $y[5] = x[0]$ with probability 1. In order to further protect against the theoretical risk of multiple linear approximations and linear hull attacks, the designers propose a minimum number of 20 rounds for RC6. Table 2.6 provides figures for the estimated number of plaintexts needed for a potential multiple linear approximation attack, with or without linear hulls [121].

Table 2.6. Multiple Linear Cryptanalysis of RC6 [121]

	#Rnds				
attack	8	12	16	20	24
basic linear attack	2^{62}	2^{102}	2^{142}	2^{182}	2^{222}
+ mult. linear approx.	2^{51}	2^{91}	2^{131}	2^{171}	2^{211}
+ mult. linear approx. + linear hulls	2^{47}	2^{83}	2^{119}	2^{155}	2^{191}

Jonsson and Kaliski construct 6-round characteristics for RC6 [273], so as to attack 8 rounds, and this leads to 2^{76} chosen plaintext pairs for differential cryptanalysis of 8 rounds, and 2^{60} known plaintexts for linear cryptanalysis of 8 rounds. Shimoyama *et al.* [474] develop further the multiple linear attack approach for RC6 with a 256-bit key, where they generalise the Piling-Up Lemma using a certain *Matrix Representation*. They achieve a 14-round key recovery attack using 2^{120} known plaintexts and 2^{186} round computations. They also achieve an 18-round key recovery attack on a weak key set of a fraction of 2^{-90} of the keys with 2^{127} known plaintexts, 2^{64} memory, and 2^{193} round computations.

As mentioned previously, the key schedule for RC6 is the same as that for RC5. No weak keys or related-key attacks have been found for RC5, perhaps owing to the key schedule being quite complicated — the design of the key schedule is somewhat incompatible with the encryption structure of RC5 or RC6.

One of the most effective attacks on RC6 is by Knudsen and Meier [306] showing that, by means of a chosen-plaintext attack, RC6 can be distinguished from a random permutation with up to 15 rounds, and for 1 in 2^{80} keys up to 17 rounds can be distinguished. Moreover, key-recovery attacks can be mounted on RC6 with up to 15 rounds faster than exhaustive search for the key. To do this, [306] considers two round iterations of a form quite different from [121]. Instead of exploiting bitwise linear approximations, input-output dependencies are considered by fixing the least-significant 5 bits in the first and third words of the input block, A and C. The correlations of the corresponding two 5-bit integer values at the output every two rounds later can then be effectively measured by χ^2 tests. This gives a considerable improvement over the basic linear attack. The attack of [306] has similarities to that considered by Baudron *et al.* [34], and by Gilbert *et al.* [218]. The approach of [306] is motivated by the fact that the least significant $\log(w) = 5$ bits in A and C are not changed by the XOR and data-dependent rotation, if both rotation amounts are zero. Small *negative* rotation events (e.g. $<<< 30$, or $<<< 31$) are also exploited in [306]. The paper [306] also analyses mini-versions of RC6 to verify the experimental evidence, and the 15-17 round attacks are extrapolated from experimental evidence computed on up to 6 rounds of RC6, where it is estimated that 2^{14} more plaintexts are needed in going from s to $s + 2$ rounds. Furthermore weak key classes are exploited for RC6 in [306], and these exist because RC6 uses addition mod 2^{32}, which introduces carry propagation into the cipher. The results of these χ^2 attacks on RC6 provide further evidence for the strength of RC6, as the results suggest that the χ^2 attacks tend to attain the same level of complexity as previous differential and linear attacks [102], and other attacks [289].

In summary, the attacks currently known on RC6 suggest that 20 rounds is secure, although the security margin may be somewhat narrow.

2.5.3 AES (Rijndael)

2.5.3.1 The design

We here consider 128-bit plaintext block Rijndael, with a 128, 192, or 256-bit key over 10, 12, or 14 rounds, respectively. Rijndael [150] has recently been selected as the Advanced Encryption Standard AES and has therefore been subject to intensive study in the last few years. Rijndael is a variant of the Square block cipher [147]. The cipher is non-Feistel, and emphasises a combination of optimal diffusion [151] with optimal nonlinearity for the S-box. The key is added linearly via XOR. Encryption is similar but not identical to decryption. In software, decryption has exactly the same speed as encryption, except on 8-bit machines when decryption is slightly slower. In hardware the speed of encryption and decryption operations is the same, but decryption requires slightly more hardware. A round of Rijndael can be written as follows,

```
Round(State,RoundKey)
{
.    ByteSub(State)
.    ShiftRow(State)
.    MixColumn(State)
.    AddRoundKey(State,RoundKey)
}
```

ByteSub is the optimally nonlinear 8×8-bit S-box operation, which is x^{-1} over GF(2^8), followed by an affine transformation. ShiftRow is a bytewise permutation over GF$(2^8)^4$, MixColumn is a 4-byte *bytewise* affine transform over GF$(2)^{32}$, in fact an MDS code, and AddRoundKey is the XOR of the key onto the output of the round. The diffusion layer is a linear transformation and comprises ShiftRow and MixColumn, and it can be shown that this diffusion has an optimum branch number of 5, and activates at least 25 bytes over 4 rounds.

The key schedule of 128-bit Rijndael over 10 rounds takes in a 128-bit key and generates $128 \times 11 = 1408$ round key bits in the form of 11 128-bit subkeys, one for each round, and one at the beginning [391]. The initial subkey is set to the key, and the remainder of the subkeys are generated iteratively using the following Key Expansion algorithm for a 128-bit key,

```
for i = 0 to 3
  W[i] = Key[i]
for j = 4 to 40 (in steps of 4)
{
  W[j] = W[j − 4]⊕ SubWord(Rotl(W[j − 1])) ⊕ Rcon[j/4]
  for i = 1 to 3
    W[i + j] = W[i + j − 4] ⊕ W[i + j − 1]
}
```

where 'Rotl' means rotate left, 'Rcon' means round constant, $W[0, 1, \ldots, 10]$ is an array of 32-bit subkeys, and SubWord() is a function that takes a four-byte input word and applies the AES S-box to each of the four input bytes to produce the output word. Note that the key-schedule uses the S-box of the enciphering process.

2.5.3.2 Security analysis

Rijndael has been selected by the NIST as the new AES. No security flaws have been found. It has been chosen for Phase II of NESSIE as a benchmark against which to evaluate other submissions.

Rijndael attracted much public attention after it became the AES [191, 219, 341, 287, 414, 66, 147, 311, 518, 359, 208, 192, 375, 377, 137, 138, 133, 124, 373, 307, 74]. However, this is also because the cipher is particularly elegant and easy to describe, using highly algebraic components — its simplicity invites analysis, and this was one of the philosophies on which Rijndael was based. The use of optimal nonlinearity followed by optimal diffusion using an MDS-based linear

transformation helps to give high resistance to differential and linear attacks for Rijndael.

One of the most successful attacks against Rijndael is the *Square attack* [150], which is a form of integral cryptanalysis [311] originally used by Daemen *et al.* to attack the Square block cipher [147]. The Square attack exploits the relatively slow avalanche of the sparse affine mapping (linear mixing along rows and columns of some matrix followed by subkey addition). It is a chosen plaintext attack of up to 6 rounds on key sizes 128, 192, and 256, which makes use of a distinguisher on 3 Rijndael rounds. For this distinguisher some input bytes are chosen to take all 256 values over 256 input chosen plaintexts so that one can predict the bytes which will take all 256 values (active bytes), the bytes which are constant (passive bytes), and the bytes which are balanced ($\oplus$ sum is zero) in future rounds. The Square attack requires 2^{32} plaintexts, 2^{72} cipher encryptions, and 2^{32} memory. Primarily, the Square attack follows the balance of certain data bytes as they progress through the cipher.

No attack is known on more than 7-8 rounds of Rijndael [191, 219, 341], the best attack being the collision attack by Gilbert and Minier which breaks up to 7 rounds [219] for key sizes 128, 192, and 256. This attack requires 2^{32} chosen plaintexts and, for key sizes 192 and 256, requires a time complexity of about 2^{140}. For key size 128 the complexity required is marginally less than exhaustive search. The attack makes use of a 4-round distinguisher which exploits, by means of the birthday paradox, the existence of collisions between some partial functions introduced by the cipher. The attack by Lucks [341] extends the Square attack to Rijndael variants with 192 and 256-bit keys, and achieves an attack on seven rounds by simply guessing the 16 bytes of the last round key and exploiting minor weaknesses in the key schedule. The attack requires 2^{32} chosen plaintexts and 2^{176} or 2^{192} encryptions for 192 and 256-bit keys, respectively. The attacks of Ferguson *et al.* [191] include a 6-round improved Square attack with time complexity about 2^{44} which is an improvement on the original 2^{72}, at the price of about $6 \cdot 2^{32}$ plaintexts. The improvement makes use of a *partial-sum* technique to reduce the workfactor over the original Square attack. There is also an improvement on the 7-round attack of [341] in [191] which requires 2^{155} or 2^{172} encryptions for 192 or 256-bit keys, respectively. There is also an alternative extension to 7 rounds by Ferguson *et al.* [191] that can break all key sizes with encryption complexity 2^{120} but which requires virtually the entire codebook of plaintexts ($2^{119} - 2^{128}$ plaintexts). Moreover, one may even break 8 rounds of Rijndael with $2^{119} - 2^{128}$ plaintexts, requiring 2^{188} or 2^{204} encryptions for 192 or 256-bit keys, respectively [191].

After two rounds, Rijndael provides full diffusion, i.e. every state bit depends on all state bits two rounds ago. This is due to the uniform structure of Rijndael, and the high diffusion. The designers [150] claim that no 4-round differential characteristic exists with probability greater than 2^{-150}, and no 4-round linear characteristic exists with a correlation greater than 2^{-75}. An analysis of the propagation of activity patterns in [150] leads to the conclusion that any linear or differential characteristic over 4 rounds must activate at least 25 bytes. Improved upper bounds are given by Keliher *et al.* on the maximum average linear

hull probability for Rijndael by noting that the linear hull effect for Rijndael is significant [287]. An upper bound on the probability is given, namely 2^{-75} for 7 rounds. In a subsequent paper the same authors improve on this result by taking into account more details of the linear characteristics of the Rijndael S-box. They improve their upper bound on maximum average linear hull probability to 2^{-92} for 9 rounds. Related work by Ohkuma *et al.* [414] upper bounds the maximum average differential and linear hull probabilities by 2^{-96} for 4 rounds of Rijndael. In [414] it is also shown how to represent Rijndael as a two-level nested Substitution Permutation Network (SPN) where each level uses an MDS layer for diffusion. There is also an impossible differential attack on 5 rounds by Biham and Keller, requiring $2^{29.5}$ plaintext-ciphertext pairs and 2^{31} time [66], and an extension to 6 rounds has been presented by Cheon *et al.* [118].

In [518] Wernsdorf shows that the round functions of Rijndael generate the alternating group over the set $\{0,1\}^{128}$, eliminating some potential weaknesses of Rijndael, e.g. non-trivial factor groups (the alternating group is a large, simple, primitive and $(2^{128}-2)$-transitive group).

With respect to the key schedule of Rijndael, Ferguson *et al.* [191] exploit weaknesses in the expanded key of Rijndael by proposing a related-key attack on 9 rounds which is a variant of the Square attack and use 256 related keys that differ in a single byte in the fourth round key. Plaintext differences are used to cancel out earlier round key differences, resulting in three bytes at the end of round 6 that sum to zero when taken over the 256 encryptions. Key bytes of the last three rounds are then guessed and used to compute backwards from the ciphertext to detect this property. This 9-round attack on Rijndael with 256-bit keys requires 2^{77} plaintexts under 256 related keys, and 2^{224} encryptions.

May *et al.* [359] provide a modified key-schedule for Rijndael with improved diffusion and nonlinearity, whilst keeping a reasonably fast speed for the key-schedule. More generally, the key schedule has a much slower diffusion than the cipher and contains relatively few nonlinear elements.

It has recently been observed by Fuller and Millan [208] that the output functions of the Rijndael S-box are all affine transformations of the same function. This observation suggests a potential extra hardware saving for the Rijndael S-box, although, perhaps more importantly, this may later also be seen as a security weakness (see Sect. 2.9.2). In other words, let b_i and b_j be two distinct output bit functions of the Rijndael S-box. Then we can always find a Boolean matrix, $\mathbf{A}$, and a Boolean vector, $\mathbf{B}$, such that,

$$b_i(x) = b_j(\mathbf{A}x + \mathbf{B})$$

This surprising result means that the Rijndael block cipher only uses *one* Boolean function from eight bits to one bit (used 128 times in each round). Since [208] was posted, Youssef and Tavares [525] have proved this result by making use of dual bases over GF(2^n) and trace functions, and showed that the result holds for any S-box based on a bijective monomial. They also extended the result to show that all *coordinate* (bit) functions of the Rijndael round function are equivalent under affine transformation of the input to the round function. Following on from [208], Biham [54] has shown that affine relationships exist between the output bits of

many S-boxes, not just the Rijndael S-box (see Sect. 2.9.2). Another simplified algebraic formulation for Rijndael was presented by Ferguson *et al.* in [192]. This paper argues that the security of Rijndael is based on a new hardness assumption for the solution of an algebraic formulation of the type derived in [192].

A recent redefinition of Rijndael has raised many questions regarding the security of the cipher. The *Big Encryption System* (BES) has been defined by Murphy and Robshaw in [375], where the AES can be regarded as being identical to the BES with a restricted message space and key space. Whereas the AES uses operations over GF(2^8) and GF(2), the BES only uses operations over GF(2^8) and this raises the question of algebraic attacks on AES. The key idea for BES is to map every GF(2^8) byte, a of AES to $(a^{2^0}, a^{2^1}, \ldots, a^{2^7})$ for BES, where this vector comprises a and its conjugates over GF(2^8). The central application of this mapping is to convert the GF(2) linear operation in the S-box of AES (an 8×8 binary matrix) to an 8×8 matrix mapping over GF(2^8) which linearly operates on a and its conjugates. This ensures that all operations of BES occur in GF(2^8), and one can write the ith round function of BES as:

$$\text{Round}_B(b\, k_i) = M_B(b^{(-1)}) + k_i$$

where b is the input to the round, M_B is a matrix with elements from GF(2^8) and k_i is the ith round key in BES. It should be noted that the key schedule can also be written completely over GF(2^8). M_B can always be converted to its Jordan Form, R_B, where $R_B = P_B^{-1} \cdot M_B \cdot P_B$ is a particularly well-structured representation for an AES round. The paper [375] identifies Related-Key and Differential attacks on BES which exhibit characteristics with probability one through a round, in spite of the high diffusion. But these attacks are not directly applicable to AES as they do not preserve the conjugacy relation which is necessary for the inverse mapping back to AES. One of the most interesting future lines of inquiry for BES is to combine it with the techniques of the type suggested by Courtois and Pieprzyk [137, 138, 133] for solving the type of multivariate quadratic systems that arise from block ciphers. A preliminary analysis [375, 377] of the complexity of an attack based on the estimates of Courtois and Pieprzyk [138] suggests an attack considerably faster than exhaustive key search. However, there are inaccuracies in these estimates [124, 377, 123] It has been noted by Knudsen and Raddum [307] that its mathematical elegance makes Rijndael more vulnerable to a devastating attack as there are no *random-looking* elements in the cipher.

As discussed in the security analyses for Khazad, MISTY1, and Camellia, Biryukov and De Cannière [74] compare minimal systems of multivariate polynomials which completely define certain block ciphers, including Rijndael-128. As pointed out in [138], the Rijndael S-box can be described by a system of 23 quadratic equations in 80 terms. It is estimated in [74] that the block cipher and key schedule can be described by 6296 equations in 3296 variables using 19296 linear and quadratic terms. For Rijndael-128 this means that there are 13000 free terms (see Sect. 2.2.3.17). Roughly the same figures occur for Camellia-128. Note that there are 37 quadratic equations in total for the AES S-box, whereas a randomly-chosen 8-bit S-box would expect to have zero quadratic equations

associated with it. However, it should be noted that this system of equations is far larger than the multivariate quadratic system provided with BES.

Recent work by Barkan and Biham [28] has developed the concept of the *Dual Cipher*. Specifically, let the ciphertext, C, be the result of encrypting the plaintext, P, using the cipher, E, where E is dependent on the secret key, k. We write this as $C = E_k(P)$. Then [28] has stated that, given any invertible functions, f, g, and h, E' is a dual cipher to E if

$$\forall P, k \qquad f(E_k(P)) = E'_{g(k)}(h(P)).$$

The dual cipher, E', is equivalent to the original cipher, E, in all aspects. One can therefore analyse and attack the dual cipher instead of the original cipher. Moreover, one can actually implement the dual cipher instead of the original cipher. For Rijndael, [28] demonstrates the existence of *Square-Dual* ciphers in the sense that every component of the original Rijndael cipher has been squared (or conjugated). Thus Rijndael has 7 dual ciphers of this form. Moreover, one can replace the irreducible polynomial used by Rijndael with another one, and this too leads to another set of dual ciphers — 240 in total. Also, the components of Rijndael can be replaced with their logs to give a set of *Log-Dual* ciphers, 128 in total. Finally, [28] shows that by straightforward modifications of Rijndael, one can create Self-Dual ciphers which exhibit cryptographic weakness although this does not imply weakness in Rijndael itself. It should be noted that this dual cipher analysis also holds for Khazad, and Anubis, and can also be applied to Camellia and SAFER$_{++}$.

2.5.4 SAFER$_{++128}$

2.5.4.1 The design

SAFER$_{++128}$ [353] is a development from the existing SAFER family of ciphers and uses a combination of substitution and linear transformation to achieve confusion and diffusion, respectively. We here consider the normal and high versions which take in 128-bit plaintexts and require 128-bit or 256-bit keys, respectively. The designers recommend 7 rounds for the 128-bit key version, and 10 rounds for the 256-bit key version.

Figure. 2.17 shows an encryption round for SAFER$_{++128}$. This figure should be compared with Fig. 2.9 for SAFER$_{++64}$. It is evident that, whereas SAFER$_{++64}$ requires zero-padding and merging of the round input, prior to and following the linear transformation, SAFER$_{++128}$ does not require this, as the round input is already of size 128 bits. For a more detailed discussion, please refer to the section commenting on SAFER$_{++64}$. Its design has many interesting properties.

2.5.4.2 Security Analysis

No security flaws have been found with SAFER$_{++128}$ [353] and it has many similarities to SAFER$_{++64}$. One weakness found in previous versions of the SAFER family by Knudsen was in the key-schedules [295, 298], but these weaknesses have been dealt with in SAFER$_{++128}$. Other pre-NESSIE attacks on the SAFER

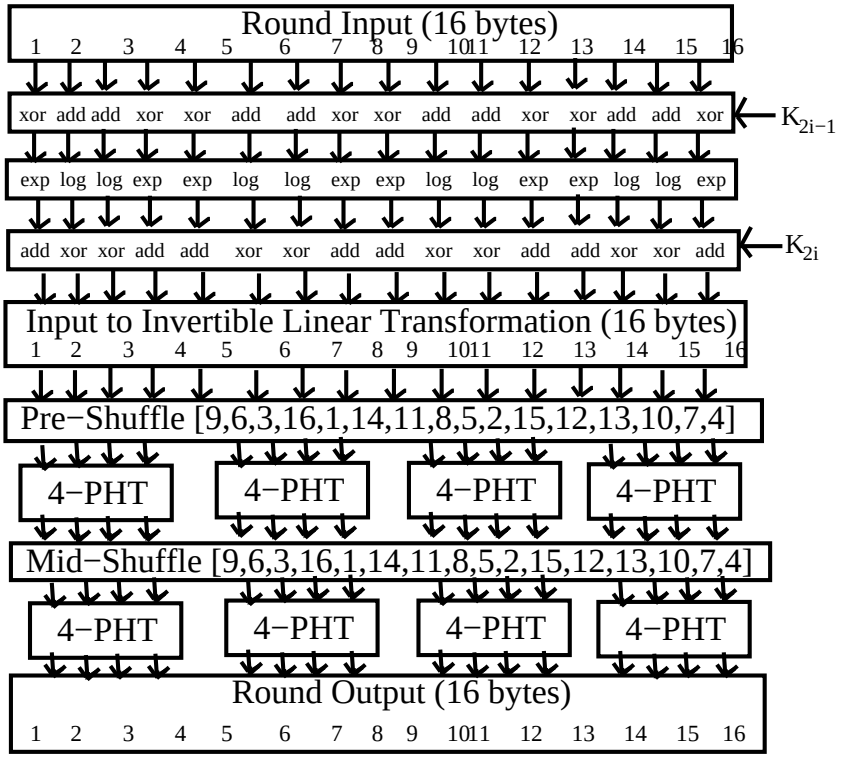

Fig. 2.17: Encryption Round for SAFER$_{++128}$

family include truncated differentials by Knudsen and Berson [304], and Murphy [374] identifies a potential algebraic weakness regarding the existence of invariant $\mathbb{Z}$-modules within the PHT layer. These modules and their cosets are not diffused by the PHT layer, and so provide a way to cope with diffusion in SAFER, regardless of the key schedule. One attack in [374] which used this property enabled a projection of the message/ciphertext space onto a 4-byte $\mathbb{Z}$-submodule so that the probability of any message projection giving any ciphertext projection is independent of $\frac{1}{4}$ of the key bytes. This result, along with the results of [295] led to a change in the SAFER key schedule. The designer claim that one of the main reasons for the security of the cipher against differential and linear cryptanalysis is the high diffusion PHT layer. The designers conclude that SAFER$_{++}$ with six or more rounds is secure against differential cryptanalysis, and with two and a half or more rounds is secure against linear cryptanalysis. However, recently Nakahara *et al.* [382] applied techniques that were first used to more generally attack the SAFER family in [381], and [382] showed that, for the 256-bit key version, up to four rounds can be attacked by linear cryptanalysis, with less effort than exhaustive search, requiring 2^{81} known plaintexts. The reason for this discrepancy between the two and a half rounds claimed by the designers and the three to four rounds claimed by [382] is largely because the designers restricted them-

selves to homomorphic attacks, whereas [382] applies strictly non-homomorphic attacks where some key bits are assumed fixed. Therefore the results of [382] must be seen in the context of a weak-key class. Table 2.7 gives a summary of the complexity of linear attacks on SAFER$_{++}$ [382]. One important point to note with regard to the linear attack of [382] is that it identifies a surprisingly small byte and bit branch number for the PHT diffusion layer. Whereas the designers of SAFER$_{++}$ use the fact that the lowest row weight of the matrix M is 10, implying a high diffusion, a more detailed examination of the matrix reveals a byte branch number ≤ 7 and a bit branch number ≤ 5. Moreover the S-box defined by $y = 45^x \mod 257$ contains a linear relationship which holds with probability 1. In other words, we can write $y_0 = f(x_0, x_1, \ldots, x_6) + x_7$, where y_0 is one of the output bits of the S-box, and depends linearly on the input, x_7. This moderated diffusion combined with a high linear characteristic is what enables this linear attack on three to four rounds of SAFER$_{++}$. Table 2.7 supplements Table 2.4 by stating the complexity of linear attacks on SAFER$_{++}$ that apply to 256-bit key input only.

Table 2.7. Complexity of Linear Attacks on SAFER$_{++}$

# Rounds Attacked	Linear Relation	# Known Plaintexts	# Subkey Bits Explored	Attack Complexity	Fraction of Keys
2	(1)	2^5	37	2^{42} ♣	
3.5	(3)	2^{33}	88	2^{121} ♣	2^{-6}
4	(4)	2^{81}	97	2^{178} ♠	2^{-13}
	(5)	2^{91}	76	2^{167} ♠	2^{-11}

♣ The attack applies to all key sizes defined for SAFER$_{++}$.
♠ This attack applies to 256-bit keys only.

It is also interesting to note that affine relationships exist between the bit outputs, both for exp45 and log45 [54] (see Sect. 2.9.1).

Impossible Differential and Square attacks have also been demonstrated on SAFER$_{++128}$ by Nakahara *et al.* [383], and Nakahara [379], respectively, over 2.75 to 3.25 rounds. Table 2.8 details these attacks along with further attacks on similar ciphers from the SAFER family.

SAFER$_{++128}$ is a Substitution-Permutation Network (SPN), and five-layer SPN's are susceptible to structural analysis leading to integral or multiset attacks. Piret [433] describes a (classical) integral distinguisher over 2 rounds of SAFER$_{++}$ (in the encryption direction). This allows a practical attack against 3 rounds of SAFER$_{++128}$, as well as attacks on 4 rounds of SAFER$_{++128}$ and SAFER$_{++256}$ (always without the last key addition layer), under the chosen-plaintext hypothesis. As a side result, Piret proves that the byte-branch number of the linear transform of SAFER$_{++}$ is precisely 5. Concrete figures for these attacks are given in Table 2.9.

Even stronger multiset attacks have recently been presented by Biryukov *et al.* [75]. The general method can be applied to any SPN network with incom-

Table 2.8. Attacks by Nakahara *et al.* for SAFER$_{++128}$ and Similar Designs

Cipher	Attack	#Rounds	Data	Memory	Time	Ref
SAFER$_{++128}$	Imp. Diff.	2.75	2^{64} CP	2^{97}	2^{60}	[383]
	Square	3.25	$2^{9.6}$ CP	$2^{9.6}$	2^{70}	[379]
	Linear	3.25	2^{81} KP	2^{81}	2^{103}	[382]
SAFER SK$_{128}$	Imp. Diff.	2.75	2^{39} CP	2^{58}	2^{64}	[383]
	Square	3.25	$2^{10.3}$ CP	$2^{10.3}$	2^{38}	[379]
	Linear	4.75	2^{63} KP	2^{63}	2^{90}	[381]
SAFER+$_{128}$	Imp. Diff.	2.75	2^{64} CP	2^{97}	2^{60}	[383]
	Square	3.25	$2^{9.6}$ CP	$2^{9.6}$	2^{70}	[379]
SAFER+$_{192}$	Linear	3.25	2^{100} KP	2^{100}	2^{137}	[382]
SAFER+$_{256}$	Linear	3.75	2^{81} KP	2^{81}	2^{176}	[382]

KP: known plaintext, CP: chosen plaintext

Table 2.9. Piret's Integral Attacks on SAFER$_{++128}$

Key Size	#Rounds	#Plaintexts	Time Compl.	Space Compl.
128	3	2^{16}	2^{16}	2^{16}
128	4	2^{64}	2^{112}	2^{64}
128	4	2^{64}	2^{120}	2^{16} (different tradeoff)
256	4	2^{64}	2^{144}	2^{64}

plete diffusion, and the method of [75] is also a collision attack, inspired by the attacks of Gilbert and Minier on Rijndael [219]. The multiset attacks of [75] can break up to 4.5 rounds of SAFER$_{++128}$ in 2^{48} chosen plaintexts and 2^{94} steps. Biryukov *et al.* [75] also present a Boomerang attack on SAFER$_{++128}$ which exploits the incomplete diffusion of SAFER++$_{128}$ and also certain special properties of the SAFER S-boxes. In this way they have constructed a 5 round attack on SAFER++$_{128}$ using 2^{75} chosen plaintexts/ adaptive chosen plaintexts and 2^{75} time complexity. The attack completely recovers the 128-bit secret key of the cipher and has a 86% probability of success. The attack can be extended to 5.5 rounds by guessing 46 bits of the secret key. The attacks of [75] are summarised in Table 2.10.

Table 2.10. Multiset and Boomerang Attacks on SAFER$_{++128}$ by Biryukov *et al.*

Attack	Key Size	#Rounds	Data	Workload	Memory
Multiset	128	4	2^{48}	2^{70}	2^{48}
Multiset	128	4.5	2^{48}	2^{94}	2^{48}
Boomerang	128	4	2^{41}	2^{41}	2^{41}
Boomerang	128	5	2^{75}	2^{75}	2^{48}
Boomerang	128	5.5	2^{121}	2^{121}	2^{48}

Workload expressed in equivalent number of encryptions.

2.6 Large block ciphers considered during Phase II

The large block ciphers selected for phase II of NESSIE were RC6, Rijndael, SHACAL-1, and SHACAL-2. None of these ciphers has been broken so the following security evaluation identifies weaknesses that occur in reduced-round versions of the ciphers, and identifies weaknesses that may lead to more effective attacks in the future. We first describe each cipher in some detail, along with the most important attacks known on each cipher. Note that the algorithms given here are not complete specifications, but references are given to complete specifications which may be found on the NESSIE website. After discussing each cipher we summarise and compare some of the distinguishing features of the ciphers, identifying potential weaknesses and noting the best-known attacks, as shown in Table 2.18 of Sect. 2.9.3.

2.6.1 RC6

2.6.1.1 The design

We consider here the version of RC6 [273] which takes 256-bit plaintexts and key-lengths from 128 to 256 bytes. For the 128-bit plaintext version the designers recommended 20 rounds, but for the 256-bit plaintext version, the recommended number of rounds is not specified. A detailed discussion of RC6 is given in the section on 128-bit block ciphers.

2.6.1.2 Security analysis

Until very recently, no security flaws have been found in RC6, and the 128-bit block variant of RC6 on which the 256-bit variant is based has been well studied.

Recently, Knudsen has detected correlations in 256-bit block RC6 using the χ^2-attack method [301]. From tests on RC6 with 256-bit blocks and three rounds together with other test results, [301] is able to extrapolate an estimated requirement for the number of plaintexts needed for this attack up to 25 rounds. It is estimated that for $3 + 2s$ rounds a χ^2 test would distinguish 256-bit block RC6 from random using 2^{16+20s} plaintexts. This constitutes a successful attack up to 25 rounds where it is expected that the χ^2 attack will require only 2^{236} plaintexts. The attack exploits the least significant five bits in the words A and C of the input of one round, and investigates the statistics of the 10-bit integer obtained by concatenating the least significant five bits in the words A'' and C'' every two rounds later. This is motivated by the fact that the least significant five bits in A and C are not changed by the XOR and data dependent rotation if both rotation amounts are zero. More generally, one can expect a bias for amounts smaller than five, and these strong biases can be iterated over many rounds in the same way as linear approximations.

2.6.2 AES Variant (Rijndael-256)

2.6.2.1 The design

We here consider a variant of the NIST AES standard, 256-bit plaintext block Rijndael, with a 256-bit key over 14 rounds. Whereas 128-bit Rijndael uses 16

8-bit S-boxes per round, 256-bit Rijndael uses 32 8-bit S-boxes per round. A detailed discussion of Rijndael is given in the section on 128-bit block ciphers.

2.6.2.2 Security analysis

No security flaws have been found, and the 128-bit block variant on which it is based was selected as the AES and has been well-studied. Although similar in structure to the 128-bit block Rijndael, the 256-bit block variant still warrants a separate analysis as the byte alignments of this variant are different from those of the 128-bit block variant.

2.6.3 SHACAL-1

2.6.3.1 The design

SHACAL [237] is a 160-bit (20-byte) block cipher using a 512-bit (64-byte) key that is based on the well-known Hash Function FIPS standard, SHA (or SHA-0). SHACAL-1 is based on SHA-1, a more recent FIPS standard, which is a minor modification of SHA-0 in the message expansion. It is considered to have very fast implementations. In hash function mode, SHA takes a 512-bit message with a 128-bit initial value. In encryption mode (block cipher), the message becomes the key, and the initial value is replaced by the plaintext. SHA mixes group operations, $+ \mod 2^{32}$ and XOR, with nonlinear logical functions. SHACAL-1 places the 160-bit plaintext in 5 concatenated 32-bit variables, A, B, C, D, E, and updates these five variables on each of 80 consecutive steps, so that the final ciphertext is contained in A, B, C, D, E after 80 steps. In the process, the 512-bit key is expanded to 2560 bits. SHACAL-1 is shown in Fig. 2.18, where ROL means *rotate left*, $+$ means addition mod 2^{32}, and f is a different linear or nonlinear function for steps $\lfloor i/20 \rfloor$.

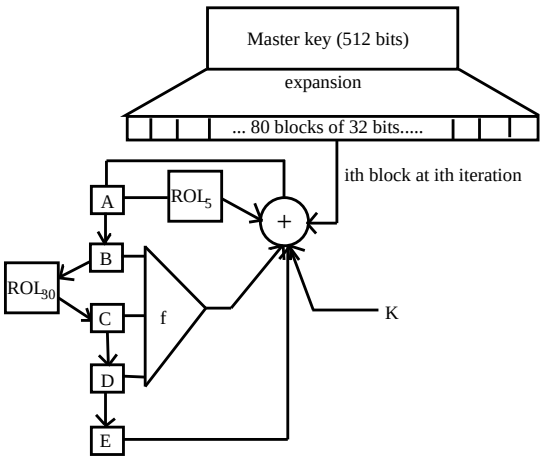

Fig. 2.18: Encryption for SHACAL-1

The encryption algorithm for SHACAL-1 is as follows.

Put the 160-bit plaintext in 32-bit variables $A\,B\,C\,D\,E$.
For 80 steps do
$A^{i+1} = W^i + \mathrm{ROL}_5(A^i) + f^i(B^i\,C^i\,D^i) + E^i + K^i$.
$B^{i+1} = A^i$.
$C^{i+1} = \mathrm{ROL}_{30}(B^i)$.
$D^{i+1} = C^i$.
$E^{i+1} = D^i$.

where the W^i are 32-bit step keys, the K^i are round-dependent constants, and $f_i(X\,Y\,Z)$ is one of three functions defined below, where the function chosen is dependent on the round.

$f_{if} = (X \text{ AND } Y) \text{ OR } (\bar{X} \text{ AND } Z)$	$0 \le i < 20$
$f_{xor} = (X \oplus Y \oplus Z)$	$20 \le i < 40, 60 \le i < 80$
$f_{maj} = ((X \text{ AND } Y) \text{ OR } (X \text{ AND } Z) \text{ OR } (Y \text{ AND } Z))$	$40 \le i < 60$

where $\oplus$ is bitwise XOR, AND and OR are both bitwise logical operations, and $\bar{*}$ is complement. Each set of 20 steps, $r \le i < r + 20$ constitutes a round of the cipher. In order for SHACAL-1 to be invertible the final addition of the initial value, which occurs in the hash mode of SHA-1, is omitted.

The message expansion is different for SHA-0 and SHA-1. For SHACAL-1 the key schedule (message expansion) is linear, it expands the 512-bit key (Master Key) to 2560 bits, and can be described as follows,

– The Master key is a concatenation of 16 32-bit words: $[W^0|W^1|\ldots|W^{15}]$.
– $W^i = \mathrm{ROL}_1(W^{i-3} \oplus W^{i-8} \oplus W^{i-14} \oplus W^{i-16})$, $16 \le i < 80$.

Keys shorter than 512 bits may be accomodated by padding the key input up to 512 bits.

2.6.3.2 Security analysis

No security flaws have been found in SHACAL-1. NESSIE considers that the security margins of SHACAL-1 are very large. It also has the interesting property of being able to share most of the code of the SHA-1 hash function. The best attack known on SHA-0 is that of Chabaud and Joux [117] who obtain collisions using 2^{61} encryptions by tracking perturbations through the hash function in combination with differential masks. However, it was found by the authors of [117] that they could not extend the attack to SHA-1 because SHA-1 interleaves bits in the message expansion so that it is not possible to split the expansion into 32 little expansions. The idea of the attack was to study the propagation of local perturbations in a linear variation of SHA-0 in order to discriminate between the role of the bare architecture and that of the elementary building blocks. The attack then looks for differential characteristic masks that can be added to the input word with non-trivial probability of keeping the output of the compression function unchanged. One first proposes a variant of SHA-0 called SHI1 which keeps all the rotations on blocks but replaces ADD with XOR, and makes the f_i functions XOR. Single bit errors or perturbations are then introduced to the input of SHI1 and the perturbation is traced through the cipher. These perturbations

are made to disappear by introducing five other perturbations. This allows an attack on SHI1 via differential masking. A second variant on SHA-0, SHI2 is then analysed, where SHI2 replaces ADD with XOR but this time keeps the nonlinear f_i. However we can still view f_i as acting like $\oplus$ with some probability, and the probability of a successful perturbation attack can be computed. A third variant of SHA-0, SHI3, is then analysed, where SHI3 uses ADD as in SHA-0, but uses XOR instead of the nonlinear f_i. In this case the addition mod 2^{32} causes the perturbations to spread out due to carry propagation. However one is still able to devise a perturbation attack on SHI3 with probability 2^{-44}. Finally SHA-0 itself is analysed by taking into account the analyses of SHI2 and SHI3, and this leads to a perturbation based attack on SHA-0 requiring 2^{61} plaintexts. It should be emphasised that, although SHA-1 and SHA-0 are so similar, this attack does not carry over to SHA-1 or, consequently, SHACAL-1.

Ad-hoc linear and differential cryptanalysis of SHA-1 by Handscuh *et al.* has suggested that such attacks will not be effective against SHACAL-1 [236, 237]. Both attacks are complicated by the integer addition and the f_i functions of SHACAL. It is noted [236, 237] that $Z = A + S$ can be written bitwise as $z_j = a_j + s_j + \sigma_{j-1}$ and $\sigma_j = a_j s_j + a_j \sigma_{j-1} + s_j \sigma_{j-1}$, where σ_{j-1} is the carry bit, and $\sigma_{-1} = 0$. This bitwise way of expressing addition is used extensively in the analysis of [236, 237]. Linear cryptanalysis mostly uses single-bit approximations as heavier linear approximations are difficult to connect together. The cyclic structure of SHACAL means that in all four rounds we can readily identify a family of linear approximations that always hold over four steps. A *perfect* (probability 1) linear approximation exists over both 4 and 7 steps. There is a 10-step linear approximation for rounds 2 and 4 which is valid over 40 steps with an estimated bias of 2^{-21}, and from these characteristics it is estimated that at least 2^{80} known plaintexts are required, although this cannot be considered a *break* as it is a very loose lower bound. For differential cryptanalysis there exists a 5-step characteristic over any 5 steps with probability 1, and it is conjectured that over 80 steps, the full cipher, the best differential characteristic has probability around 2^{-116}. It is emphasised in [236, 237] that their estimations are over-favourable to the cryptanalyst as it would be impossible to connect up all the constituent characteristics so as to achieve these biases. Neither is it expected that the more refined techniques involving linear hulls, multiple linear approximations, differentials, ... will make much difference.

van den Bogaert and Rijmen [504] search for optimal differential characteristics for reduced round SHACAL. The search is performed under the requirement that the Hamming Weight of every 32-bit word of the input is upper bounded by 2. It is found that there are two 10-step characteristics for f_{if} with probability 2^{-12} (this is a factor of 2 better than [237]), a 10-step characteristics for f_{xor} with best case probability 2^{-12}, and a 20-step characteristic for f_{if} and f_{maj} with probabilities 2^{-42} and 2^{-41} respectively (these figures agree with [237]).

Recently Saarinen [458, 460] has noted that a slide attack can be mounted on SHA-1 with about 2^{32} effort. This attack is described in detail in Chapter 4 on Hash Functions in this report, with respect to a security analysis of SHA-1. The analysis demonstrates an unexpected property of the compression function of

SHA-1, namely that the procedure for message expansion can be slid. However, it is not clear that this weakness can be exploited in the context of SHACAL-1. Also, in [460], Saarinen shows that the slide attack on SHA-1 points to a weakness in the key schedule of SHACAL-1, and this can be exploited in a related-key attack. Given access to two SHACAL-1 encryption oracles whose keys are "slid" (in the same way that the message expansion can be slid for the hash function) the cipher can be distinguished from a randomly chosen 160-bit permutation. This requires about 2^{128} chosen plaintexts. When certain properties hold for the (related) keys, the complexity can be further reduced to about 2^{96} chosen plaintexts. Differential cryptanalysis including boomerang attacks [290] and rectangle attacks [68] have also been applied to SHACAL-1. The best known attack works for 49 steps of the compression function with a data complexity of $2^{151.9}$ chosen plaintexts and a time complexity of $2^{508.5}$ [68].

2.6.4 SHACAL-2

2.6.4.1 The design

SHACAL-2 is based on SHA-2, which was introduced by NIST in 2000 [390, 393]. In spite of similarity in name to SHACAL-1, SHACAL-2 is a completely different function. It is a 256-bit block cipher with a 512-bit key, although it can also be configured to take 512-bit blocks. SHA-2 operates in a similar way to SHA-1 but with some notable differences.

The encryption algorithm for SHACAL-2 is as follows.

– Put the 256-bit plaintext in eight 32-bit variables $A\,B\,C\,D\,E\,F\,G\,H$.
– For 64 steps do
– $T_1 = H + \sum_1(E) + Ch(E\,F\,G) + K^i + W^i$.
– $T_2 = \sum_0(A) + Maj(A\,B\,C)$.
– $H^{i+1} = G^i$.
– $G^{i+1} = F^i$.
– $F^{i+1} = E^i$.
– $E^{i+1} = D^i + T_1$.
– $D^{i+1} = C^i$.
– $C^{i+1} = B^i$.
– $B^{i+1} = A^i$.
– $A^{i+1} = T_1 + T_2$.

where the W^i are 32-bit step keys, the K^i are constants, different in each step, and

$$\begin{array}{ll} Ch(X\,Y\,Z) & = (X \text{ AND } Y) \oplus (\bar{X} \text{ AND } Z) \\ Maj(X\,Y\,Z) & = (X \text{ AND } Y) \oplus (X \text{ AND } Z) \oplus (Y \text{ AND } Z) \\ \sum_0(X) & = S_2(X) \oplus S_{13}(X) \oplus S_{22}(X) \\ \sum_1(X) & = S_6(X) \oplus S_{11}(X) \oplus S_{25}(X) \end{array}$$

where S_i means right rotation by i bits.

In order for SHACAL-2 to be invertible the final addition of the initial value, which occurs in hash mode for SHA-2, is omitted.

For SHACAL-2 the key schedule (message expansion) expands the 512-bit key (Master Key) to 2048 bits, and is as follows,

– The Master Key is a concatenation of 16 32-bit words: $[W^0|W^1|\ldots|W^{15}]$.
– $W^i = \sigma_1(W^{i-2}) + W^{i-7} + \sigma_0(W^{i-15}) + W^{i-16} \quad 16 \leq i < 64$.

where σ_0 and σ_1 are defined as:

$$\begin{aligned} \sigma_0(x) &= S_7(x) \oplus S_{18}(x) \oplus R_3(x) \\ \sigma_1(x) &= S_{17}(x) \oplus S_{19}(x) \oplus R_{10}(x) \end{aligned}$$

where R_i means right shift by i bits. Keys shorter than 512 bits may be accomodated by padding the key input up to 512 bits.

2.6.4.2 Security analysis

No security flaws have been found in SHACAL-2. It also has the interesting property of being able to share most of the code of the SHA-2 hash function. Saarinen [460] has noted that the Slide attack on SHA-1 does not carry over to SHA-2, and hence does not consitute a threat for SHACAL-2. SHA-2 is a recently designed primitive, so more time is needed to perform a careful and thorough security evaluation of both SHA-2 and, consequently, SHACAL-2.

2.7 64-bit block ciphers not selected for Phase II

The 64-bit block ciphers not selected for phase II of NESSIE were CS-Cipher, Hierocrypt-L1, Nimbus, and Nush. After discussing each cipher briefly we summarise and compare some of the distinguishing features of the ciphers, identifying potential weaknesses and noting the best-known attacks, as shown in Table 2.19 of Sect. 2.9.4. Note that the algorithms given here are not complete specifications, but references are given to complete specifications which may be found on the NESSIE website.

2.7.1 CS-cipher

2.7.1.1 The design

CS-Cipher is a 64-bit block, 128-bit key SPN cipher over 8 rounds [198]. Each round starts with a subkey XOR, followed by a layer of four 16-bit non-linear mixing transformations, M, and a byte permutation which is based on the Fast Fourier Transform (FFT) graph. This is repeated twice more in each round, but with constants used instead of the subkey in the initial XOR. There is a final key XOR after the last round. Whereas many ciphers use separate nonlinear (confusion) layers and linear (diffusion) layers, CS-Cipher also uses a nonlinear diffusion primitive within the nonlinear layer. The encryption process is summarised as,

$$k^8 \oplus E(k^7 \oplus \ldots E(k^1 \oplus E(k^0 \oplus m)) \ldots)$$

where k^i are the subkeys, m is the plaintext, and E is the round encryption function. The key schedule generates 9 subkeys of 64 bits each, is Feistel, and is summarised by,

$$k^i = k^{i-2} \oplus F_{c_i}(k^{i-1}) \qquad F_{c_i}(x) = T(P(x \oplus c^i))$$

where T is transposition, P is permutation, and the c_i are constants. Further design details may be found in [198].

2.7.1.2 Security analysis

Two successive applications of an FFT graph have been shown to give very good diffusion properties. The M transformation implements a multi-permutation [467] which here means that fixing either of the two 8-bit inputs arbitrarily makes both 8-bit outputs permutations of each other. This makes E a mixing function so that, if we arbitrarily fix seven of the eight 8-bit inputs, all outputs become permutations of the remaining free input. This gives good diffusion. Also each byte in the output of one round depends on all eight input bytes to that round. The non-linear transformation of the encryption takes two bytes of input, and has the following property: if one takes 256 inputs that are constant in one byte and take on all values once in the other byte, then each byte value occurs once in each of the two output bytes. This nonlinear transformation includes reasonably nonlinear involutions, P, and the designers show that at least five P boxes must be active per round, and this implies a satisfactory resistance to differential or linear cryptanalysis.

In [511] the designers prove, by counting the number of active S-boxes, that a modified version of CS-cipher with all constants and round keys replaced by independent random values is secure against linear and differential cryptanalysis. The designers [511] claim that these results carry over to the real CS-cipher and that $5\frac{1}{3}$ rounds of CS-cipher is therefore secure against linear and differential cryptanalysis.

No weaknesses or attacks have been reported on CS-cipher.

2.7.2 Hierocrypt-L1

Attacks on Hierocrypt-L1 significantly reducing the security margin have been found that the submitters were not aware of [33].

2.7.2.1 The design

Hierocrypt-LI (HC-L1) is a 64-bit block, 128-bit key SPN block cipher over 6 rounds [412]. It is hierarchical in structure with an SPN structure itself built of smaller SPN structures. Each round consists of a layer with two parallel XS-boxes each operating on a 32-bit input, followed (except in the last round) by a linear diffusion layer with a 64-bit input based on a bytewise MDS matrix. An XS-box consists of an upper subkey mixing layer, an upper S-box layer, a linear MDS-based diffusion layer (with a 32-bit input), a lower subkey mixing layer, and

a lower S-box layer. The subkeys are XORed in, and the S-box used has 8-bit inputs and outputs. After the last round an output transformation introduces another subkey.

The key-schedule consists of two processes, namely the intermediate-key-generation and the round-key generation. The former generates intermediate keys out of the secret key, while the latter generates a round key out of each intermediate key. There are two ways to generate round keys; one is for a certain number of rounds close to the plaintext, and the other is for the remaining rounds nearer to the ciphertext. Most intermediate keys are used to generate one plaintext-side and one ciphertext-side round key. The schedule algorithms adopt a Feistel structure with a linear key scheduling in order to update intermediate keys, so that intermediate keys are supposed to be partially randomised in a non-linear manner.

For further details of the design refer to [412].

2.7.2.2 Security analysis

In the submission [412] the designers give a plausible argument that 6 rounds of Hierocrypt-L1 is secure against differential and linear cryptanalysis. They claim integral attacks for consistency work only up to 5 S-box layers (2.5 rounds) and that truncated differential attacks work only up to 5 rounds. During the 2nd NESSIE workshop the designers gave bounds not just on the best differential/linear characteristic, but also on the best differential and linear hull [414]. For 4-round Hierocrypt-L1, the upper bound on the probability of both is 2^{-48}.

Improved integral attacks for consistency on Hierocrypt-L1, for 6 or 7 S-box layers (3 or 3.5 rounds) have been found. This is not a real threat to the security of the cipher, but it is a better attack than the designers claimed exists. During the NESSIE assessment phase, an integral attack for consistency on 3.5 rounds was found by Barreto *et al.* [33]. Another such attack was also found by the designers. However, although a Gilbert-Minier distinguisher-type attack was successfully applied to 7 rounds of Rijndael by Gilbert and Minier [219], the alternation of upper and lower MDS diffusion layers effectively prohibits the construction of a 5-round Gilbert-Minier-type distinguisher on Hierocrypt.

It is also interesting to note that, as Hierocrypt uses essentially the same S-box as Rijndael, affine relationships exist between the bit outputs of the S-box [208, 54] (see Sect. 2.9.5).

Further to this, Furuya and Rijmen [210] discovered linear relationships between the master key and several of the round subkeys. These flaws are common to key scheduling of all members of the Hierocrypt family. The attack of [210] exploits the fact that all intermediate keys of the key schedule are used to generate round keys, whereas the randomising effect of a Feistel structure requires that only half the intermediate keys are used. This flaw results in simple linear relations between intermediate keys. Also it turns out that the right halves of certain intermediate keys can be calculated from the right half of the padded secret key. Finally, because the Hierocrypt key schedule is similar to that of DES, it exhibits significant deterministic iterative differentials.

A summary of the designers' known attack requirements compared to the attack requirements found by NESSIE is given in Table 2.11.

Table 2.11. Attack Requirements for Hierocrypt-3 and Hierocrypt-L1

Attack	#S-box Layers	#Chosen-Plaintexts	#Subkey-Guesses
HC-3			
Designers	5	2^{13}	2^{168}
NESSIE	6	6×2^{32}	2^{40}
NESSIE	7	22×2^{32}	2^{168}
HC-L1			
Designers	5	2^{32}	2^{72}
NESSIE	6	6×2^{32}	2^{40}
NESSIE	7	14×2^{32}	2^{104}

2.7.3 Nimbus

There is a very practical attack on Nimbus by Biham and Furman [63].

2.7.3.1 The design

Nimbus is a 64-bit block cipher with a key of at least 128 bits over 5 rounds [344]. It is not an SPN. Each round consists of a subkey XOR, multiplication by another subkey, mod 2^{64}, and then bit-reversal of the data word. An encryption round is given by,

$$Y_i = K_i^{\text{odd}} \cdot g(Y_{i-1} \oplus K_i)$$

where K_i and K_i^{odd} are subkeys (K_i^{odd} is always odd), $\oplus$ is XOR, g is the bit-reversal function, and $\cdot$ is multiplication mod 2^{64}. Y_0 is the plaintext, and Y_5 is the ciphertext.

The key schedule generates ten 64-bit subkeys, with two new subkeys used in each round. These ten subkeys are generated from the user input key, which is at least 128 bits, by means of nested Nimbus encryption operations on successive 64-bit blocks of the user input key, with the encryption key being a constant derived from π.

For further details of the design please refer to [344].

2.7.3.2 Security analysis

The designer claims that Nimbus is secure against differential and linear cryptanalysis, interpolation attacks, impossible differential attacks, saturation attacks and related key attacks, and also claims that there is no effective attack on more than 4 rounds. Two new iterative differentials for multiplication operations with probability about $\frac{1}{2}$ have been found. By applying one of these differentials to Nimbus, a 1-round iterative differential characteristic with probability $\frac{1}{2}$ can be obtained. Iterating this to the full 5-round cipher, a differential characteristic with probability 2^{-5} is obtained. This characteristic was used by Furman [209]

to devise an attack on full Nimbus using 256 chosen plaintexts and 2^{10} complexity. Recently, Borisov *et al.* [99] summarised the attack of [209] using the language of multiplicative differentials, redefining the differential pairs of [209] as (x, x^*) where $x^* = -x \mod 2^{63}$ but $x^* \neq -x \mod 2^{64}$, a property that survives multiplication by the relevant key bits.

2.7.4 Nush

2.7.4.1 The design

We here consider the version of Nush which is a 64-bit block cipher, with a 128, 192, or 256-bit key over 9 rounds [329]. 128-bit and 256-bit block sizes were also submitted to NESSIE. Each round consists of four iterations. In each iteration two of four variables are updated using a subkey, while the other two are changed in a non-linear manner. Then the four registers are cycled round (byte-wise) in a Feistel way. Nush does not use S-boxes and, to introduce nonlinearity, there are four different kinds of operations, using XOR, OR, AND, and bit rotations. Like IDEA, Nush depends on the mixing of non-commutative operations for confusion and diffusion. The cipher also has a pre-whitening step and a post-whitening step where a subkey is added via XOR.

The key schedule of Nush simply takes the user input key and partitions this key into different subkeys for use in the encryption algorithm. No nonlinearity is used in the key schedule. For further details of the design please refer to [329].

2.7.4.2 Security analysis

In the submission [329], the designers mention resistance against differential and linear cryptanalysis, weak key and related key attacks as well as other attacks, but do not include any details of their analysis.

A linear attack with complexity less than that of exhaustive key search for the different variants of NUSH has been reported by Wenling and Dengguo [516]. NESSIE has confirmed that the linear approximation used in the attack is correct. This approximation is effective over the full cipher and holds with probability $\frac{1}{4}$ or $\frac{3}{4}$ depending on whether AND or OR is chosen in the iteration. Specifically, the linear approximation is of the form,

$$A_i[0] \oplus B_i[0] \oplus D_i[0] \simeq A_{i-1}[0] \oplus B_{i-1}[0] \oplus D_{i-1}[0]$$

where A, B, C, D are 16-bit partitions of the input/output block. But NESSIE has found some flaws in the further analysis. NESSIE has devised attacks on NUSH with 64-bit or 128-bit block size based on this linear approximation. These attacks are slightly faster than exhaustive search (by a factor of 2) for all key sizes. Furthermore, removing the first two rounds leaves all nine variants of NUSH vulnerable to a linear attack, suggesting a very limited security margin for the cipher.

2.8 128-bit block ciphers not selected for Phase II

The 128-bit block ciphers not selected for phase II of NESSIE were Anubis, Grand-Cru, Hierocrypt-3, Noekeon, Nush, Q, and SC2000. After discussing each cipher briefly we summarise and compare some of the distinguishing features of the ciphers, identifying potential weaknesses and noting the best-known attacks, as shown in Tables 2.20 and 2.21 of Sect. 2.9.5. Note that the algorithms given here are not complete specifications, but references are given to complete specifications which may be found on the NESSIE website.

2.8.1 Anubis

No security flaws have been found in the tweaked version of Anubis, and it is very similar to Rijndael.

2.8.1.1 The design

Anubis is a 128-bit SPN block cipher which accepts keys of length $32N$ bits (a minimum of 128 bits) and uses $8 + N$ rounds depending on the key size — a minimum of 12 rounds [29]. Each round consists of a subkey addition, 16 S-boxes (8-bit to 8-bit) and a linear transformation (presented as a matrix transpose operation). As in Khazad, all round components were chosen to be involutions in order to guarantee that encryption and decryption are identical but with the order of the subkeys reversed. Confusion and diffusion layers are kept separate, with the diffusion layer realised as a matrix transposition followed by a linear transformation designed to be an MDS code. In the original submission the S-boxes were randomly generated to avoid any internal structure. This tended to have a high cost in hardware, hence the tweaked submission used an S-box which could be decomposed into 3 layers of 4×4 mini-S-boxes, the same S-box as for Khazad (see Fig. 2.4 for Khazad), which has a complexity estimated to be one fifth of that for Rijndael [32].

The key schedule of Anubis is complicated and appears to be quite strong. It expands the cipher key into a series of round subkeys and uses a two-stage key-evolution and key-selection function. The schedule makes use of the encryption S-box combined with permutation, linear transformations based on MDS codes, and the addition of constants. For further details of the design refer to [29].

2.8.1.2 Security analysis

The designers claim [29] that no 4-round differential characteristic with probability higher than 2^{-125} exists and that no 4-round linear approximation with bias of more than $2^{-57.5}$ exists. They claim that related key attacks, interpolation attacks and boomerang attacks are infeasible, that truncated differential attacks only work up to 6 rounds and that saturation attacks work only up to 6 rounds. Such a saturation attack requires 6×2^{32} chosen-plaintexts, 2^{24} bits of storage (in addition to the memory required for storing the plaintext-ciphertext pairs) and time equivalent to 6×2^{48} S-box lookups. Extending the attack to 7 rounds requires almost the entire code book, 2^{64} bits of additional storage and analysis time equivalent to about 2^{120} encryptions. For 8 rounds, 2^{104} storage bits and

analysis time of 2^{204} encryptions is required (with the same data complexity). The designers claim [29] that the Gilbert-Minier attack can break 7 rounds using 2^{32} chosen -plaintexts and about 2^{140} S-box lookups. The designers claim [29] that the best impossible differential attack is on 5 rounds and uses $2^{29.5}$ chosen-plaintexts and 2^{31} analysis time.
Other than a small error in the calculation of linear approximation biases (the maximal bias is 17/256 instead of 13/256) that seems to have no effect on the overall security, no weaknesses or attacks have been reported. Apart from this mistake,it seems that the rest of the claims are correct, and that even the linear weakness cannot be exploited.

A summary of the known attacks according to the designers is given in Table 2.12.

Table 2.12. Designers Claims of Security — Known Attacks

Attack	Rounds	Key Size	Complexity		
			Data	Memory	Time
Saturation	6	all	6.2^{32} CP	2^{24} bits	6.2^{48}
	7	all	$2^{128} - 2^{119}$ CP	2^{64} bits	2^{104}
	8	> 204	$2^{128} - 2^{119}$ CP	2^{104} bits	2^{204}
Gilbert-Minier	7	> 140	2^{32}		2^{140}
Imp. Diff.	5	all	$2^{29.5}$		2^{31}

2.8.2 Grand Cru

2.8.2.1 The design

Grand Cru is a 128-bit block SPN block cipher over 10 rounds, requiring a key of at least 128 bits [100]. Grand Cru can be viewed as an enhanced version of Rijndael [150]. Rijndael encryption for a 128-bit key, K^0, can be described by,

$$\sigma_{K_{10}^0} \circ \pi \circ \gamma \circ \sigma_{K_9^0} \circ \prod_{i=8}^{0} (\theta \circ \pi \circ \gamma \circ \sigma_{K_i^0})$$

where σ is round key addition, γ is nonlinear substitution, π is a byte permutation, and θ is a linear transformation on a subset of the bytes. Grand Cru adds three keyed operations to give a four layered cipher (comprising four *subciphers*):

$$\psi_{K_1^3} \circ \nu^{-1} \circ \sigma_{K_{10}^0} \circ \beta_{K_9^2} \circ \pi_{K_9^1} \circ \gamma \circ \sigma_{K_9^0} \circ \prod_{i=8}^{0} (\beta_{K_i^2} \circ \theta \circ \pi_{K_i^1} \circ \gamma \circ \sigma_{K_i^0}) \circ \nu \circ \psi_{K_0^3}$$

where π_{K^1} is now a keyed permutation, where the round subkey K_i^1 can take on $(4!)^5$ possible values, β_{K^2} is a keyed byte-wise rotation, where K_i^2 can take on 2^{48} values, and two outer round key additions, ψ_{K^3}, are appended, using addition mod 2^8. ν is an extra diffusion layer. Note that the S-box is identical to that used in [150], as is the 4×4 matrix over GF(2^8) described by θ.

The cipher requires four 128-bit keys to derive the subkeys for the different keyed operations. When the user-selected key is shorter than 512 bits, these four keys are derived using a one-way function. All round subkeys are derived using the Rijndael key schedule. Further details of the design can be found in [100].

2.8.2.2 Security analysis

Grand Cru is a cipher that implements *multiple layered security.* The idea behind this is to mix several ciphers in such a way that if all but one of them are broken, one is still left with a secure cipher. The designer shows that introducing the keyed operations not found in Rijndael does not reduce the security compared to Rijndael [100]. There is also a short analysis of the different ciphers that emerge when all but one set of the subkeys are known or chosen. It should be noted that Rijndael is very close to being one of these ciphers. Hence the designer claims that any attack that breaks Grand Cru also breaks Rijndael. Only in the case of weak keys for the permutation subcipher, π, and the rotation subcipher, β, may Grand Cru be weaker than Rijndael. The effectiveness of the different subciphers can be examined by assuming that the other subcipher keys are known or chosen. The designer shows that for the permutation subcipher there is a meet-in-the-middle attack requiring 2^{110} operations and storage (faster than exhaustive search).

No attacks or weaknesses have been reported by NESSIE on Grand Cru.

2.8.3 Hierocrypt-3

Attacks on Hierocrypt-3 significantly reducing the security margin have been found by Barreto *et al.* that the submitters were not aware of [33].

2.8.3.1 The design

Hierocrypt-3 (HC-3) is a 128-bit block SPN cipher taking 128-bit, 192-bit, or 256-bit keys, and operating over 6, 7, or 8 rounds, depending on the key size [412]. Like HC-L1, HC-3 has a hierarchical structure. At the highest level, an HC-3 round consists of, in order:

- A layer of four simultaneous applications of 32×32-bit keyed substitution boxes (XS-boxes).
- A diffusion layer consisting of a bytewise linear transform defined by the MDS_H matrix.

Within each round a similar structure exists. A 32-bit XS-box consists of, in order:

- An upper subkey mixing layer which XORs 32-bit input data with four subkey bytes.
- An upper (key-independent and nonlinear) S-box layer composed of the parallel application of four 8×8-bit S-boxes.
- A diffusion layer consisting of a bytewise linear transform defined by the MDS_L matrix.
- A lower subkey mixing layer.
- A lower S-box layer.

The output transformation is composed of an XS-box layer followed by an XOR layer with the last 128-bit subkey. The key schedule for Hierocrypt-3 follows the same algorithm as for Hierocrypt-L1, which is discussed under 64-bit block ciphers. More details of the design can be found in [412].

2.8.3.2 Security analysis

In the submission [412], the designers show that Hierocrypt-3 is secure against differential and linear cryptanalysis using conservative estimates. They also studied integral (Square) attacks for consistency and claim they work only up to 4 S-box layers (2 rounds) for a 128-bit key and up to 5 S-box layers (2.5 rounds) for a 192-bit or 256-bit key. They claim that 5 rounds is secure against truncated differentials. During the 2nd NESSIE workshop the designers gave bounds not just on the best differential and linear characteristics, but also on the best differential and linear hull [414]. For 4-round Hierocrypt-3, the upper bound on the probability of either is 2^{-96}.

During the NESSIE assessment phase, an integral attack for consistency was found on 7 S-box layers (3.5 rounds) by Barreto *et al.* [33]. Another such attack was also found by the designers. The attack requirements are summarised in Table 2.11 which is located in the subsection related to Hierocrypt-L1, Sect. 2.7.2, along with further comments regarding the security of Hierocrypt-3.

2.8.4 Noekeon

Both key schedules of Noekeon were found, by Knudsen and Raddum [308], to be susceptible to related key attacks.

2.8.4.1 The design

Noekeon is a 128-bit block, 128-bit key SPN cipher over 16 rounds [149]. Each round operates on four 32-bit words, a_0, a_1, a_2, a_3 and starts with the addition of a round constant to a_0. Then a_0 and a_2 are XORed together to make the word w, and two copies of w are made. One of the copies is rotated 8 bits to the left, and the other is rotated 8 bits to the right. These rotated copies are then XORed back onto w, and w is XORed onto a_1 and a_3. After this the 128-bit working key (see below) is added to the four words. Then a_1 and a_3 are used to create a temporary w in the same way as described above, and this word is XORed onto a_0 and a_2. The words a_1, a_2 and a_3 are then rotated 1, 5, and 2 bits, respectively, to the left. Then, for all 32 positions in a word, the bits that are in the same position in the different words are passed through a 4-bit S-box (32 parallel S-boxes in total). Finally, the round ends with the words a_1, a_2, a_3 being rotated 1, 5 and 2 bits, respectively, to the right. After the last round, the linear operations described before the rotations of a_1, a_2, a_3 are repeated one more time. The encryption and decryption routines are very similar.

Noekeon has two key schedules, one for applications where related-key attacks are not considered dangerous and one for applications where related-key attacks can be mounted. The stronger key schedule consists of taking the user selected key and encrypting it once with the all zero key. The ciphertext is then used as

the so-called working key. The simpler key schedule simply uses the user selected key as the working key.

More details of the design can be found in [149].

2.8.4.2 Security analysis

The designers [149] claim resistance against linear and differential cryptanalysis. For linear attacks they claim that no 4-round linear characteristic with correlation coefficient higher than 2^{-24} exists. For differential attacks, the designers give a plausible argument that no 4-round differential with probability higher than 2^{-48} exists. The bounds are sufficient to conclude resistance against both attacks. The designers' reasoning that Noekeon is not vulnerable to attacks based on truncated differentials and to the interpolation attack also seems to be correct. Note, however, that the constituent functions of the S-box include the identity function on four of the sixteen different inputs.

In [308] Knudsen and Raddum show that there exist many related keys for which plaintexts of certain differences result in ciphertexts of certain differences with high probabilities independent of the key schedule used. It is also shown in [308] that for six of seven S-boxes which satisfy the design criteria of the Noekeon designers, the resulting block ciphers are vulnerable to either a differential attack, a linear attack or both. It is concluded that Noekeon is not designed according to an optimal diffusion strategy [308].

2.8.5 Nush

2.8.5.1 The design

This version of Nush is a 128-bit block cipher, with a 128, 192, or 256-bit key over 9 rounds [329]. 64-bit and 256-bit block sizes were also submitted to NESSIE.

More details of the design can be found in [329].

2.8.5.2 Security analysis

Nush has a very limited security margin [398]. Details of the algorithm and security analysis can be found in Sect. 2.7.4 which describes the 64-bit version of Nush.

2.8.6 Q

There are attacks on Q by Biham *et al.* and Keliher *et al.* both faster than exhaustive search [65, 286].

2.8.6.1 The design

Q is a 128-bit block cipher with a key size of 128, 192, or 256 bits over 8 rounds for 'low security' and over 9 rounds for 'high security' [360]. It was designed to be faster than Serpent and also to be immune to differential and linear cryptanalysis. The data is divided into sixteen 8-bit words (a 4×4 matrix of bytes). At the beginning of each round, a subkey word is XORed in, then an 8×8 S-box is applied 16 times (for each of the 16 data bytes) and an additional subkey XOR

is performed. A 4×4 S-box is then applied 32 times, the round key is XORed in, a byte permutation is performed, and the 4×4 S-box is again used 32 times. After the last round, there is additional subkey XOR, an 8×8 S-box layer, a subkey XOR and an additional subkey XOR (post-whitening). Two of the three subkeys used in each round are the same, and are equal for all rounds and also for the last half round.

The key schedule applies an operation similar to encryption to the low order half of a 256-bit key (which may be a zero-padded 128-bit key. It includes an XOR with a round counter and an XOR with a constant derived from the Golden Ratio.

More details of the design can be found in [360].

2.8.6.2 Security analysis

The designers have claimed [360] that there are no differentials with probability larger than 2^{-120} after 7 rounds, and that there is no differential attack on the full Q. An equivalent claim is made regarding linear cryptanalysis. The designers also claim security against key-related attacks, slide attacks, Davies-Murphy attacks, boomerang attacks, approximation attacks and impossible differential attacks.

During the NESSIE assessment phase, a differential with higher probability than the upper limit claimed by the designers was found by Biham and Furman and used to break the full Q. These results were presented at FSE 2001 [64]. A paper performing linear cryptanalysis of the full Q, with a significantly better attack than this differential attack, was presented by Keliher *et al.* at the 2nd NESSIE workshop [286]. The differential attack on full Q with 128-bit keys requires 2^{105} chosen plaintexts and has a time complexity of 2^{77} encryptions. The best attack on the full Q with larger key sizes requires 2^{125} chosen ciphertexts, and has a time complexity of 2^{96} for 192-bit keys, and 2^{128} for 256-bit keys. Table 2.13 summarizes these results.

Table 2.13. Data/Time Complexity of Attacks on Q for Different Key Sizes

Key Size (bits)	Round Number	Chosen Plaintexts	Complexity (encryptions)
128	8	2^{105}	2^{77}
192	9	2^{125}	2^{96}
256	9	2^{125}	2^{128}

2.8.7 SC2000

2.8.7.1 The design

SC2000 is a 128-bit block cipher taking a 128, 192, or 256-bit key over 6.5 or 7.5 rounds [475]. It is a mixture of a Feistel cipher and an SPN. The round function of SC2000 consists of a layer of 32 parallel 4-bit S-boxes followed by two rounds of a Feistel network. The round keys are XORed with the cipher block before and after the application of the S-boxes. The last half round consists of key additions

and S-box look-ups. The F-function in the Feistel network consists of a layer of four 6-bit S-boxes and eight 5-bit S-boxes, multiplication by a fixed 32×32 bit matrix and a final mixing of the words with each other using the AND operation with a constant and XOR. This produces two words of output from the function in the Feistel network. This two-round Feistel structure is the last operation on one round of SC2000. The key schedule is a complex transformation of the key selected by the user, in such a way that every 32-bit word of the round keys depends on the whole key.

More details of the design can be found in [475].

2.8.7.2 Security analysis

The designers [475] have studied the efficiency of the key avalanche and conclude that the complexity is that of exhaustive search if one tries to bypass one round in the beginning and end by guessing some of the key bits. A differential attack on 4.5 rounds has been reported by the designers. This attack finds 28 bits in the first and last round key. The key schedule in SC2000 appears to be very strong. The knowledge of one round key doesn't seem to leak any information about any of the other round keys or the key selected by the user, so the key schedule prevents the attacker from searching exhaustively for the remaining key bits. However, the success of this attack raises some questions about the design of SC2000. In [476] the designers present differential characteristics with higher probabilities than those found in their submission to NESSIE.

Raddum and Knudsen [449] and Dunkelman and Keller [177] both report attacks on SC2000 when the number of rounds is reduced to 3.5 or 4.5 from the original 6.5. In [449] two different 3.5-round differential characteristics with probabilities 2^{-106} and 2^{-107} are given. These characteristics have higher probabilities than those reported in [380]. The characteristics can be used to extract up to 32 bits of the first and last round keys in a 4.5-round variant of SC2000. In [177] distinguishers for 2.5 and 3 rounds are found and used to attack a 3.5 round variant of the cipher. These results on SC2000 were presented at the 2nd NESSIE workshop.

It is also interesting to note that affine relationships exist between the bit outputs of the S-box, for all three S-boxes, S4, S5, and S6 [54] (see Sect. 2.9.5).

2.9 Comparison of studied block ciphers

2.9.1 64-bit block ciphers considered during Phase II

Table 2.14 highlights the best attacks known on the block ciphers considered in this section. It identifies some unusual features of each cipher and some of its potential weaknesses, and gives a list of the best attacks, including the number of rounds broken, data and time complexities, and references for the attacks. MISTY2 and KASUMI are not part of NESSIE but are included for comparison with MISTY1.

Table 2.15 shows affine bit-relations between output bits of the S-boxes, i.e. the number of equations of the form,

$$b_i(x) = b_j(\mathbf{A}x + \mathbf{B}) + c$$

for the vector of S-box input bits, x, a Boolean matrix $\mathbf{A}$, a Boolean vector $\mathbf{B}$, and a constant, $c \in \{0, 1\}$. The submatrices of $\mathbf{A}$ which are permutation or rotation matrices are also enumerated [54].

2.9.2 128-bit block ciphers considered during Phase II

Table 2.16 highlights the best attacks known on the block ciphers considered in this section. It identifies some unusual features of each cipher and some of its potential weaknesses, and gives a list of the best attacks, including the number of rounds broken, data and time complexities, and references for the attacks.

Table 2.17 shows affine bit-relations between output bits of the S-boxes, i.e. the number of equations of the form,

$$b_i(x) = b_j(\mathbf{A}x + \mathbf{B}) + c$$

for the vector of S-box input bits, x, a Boolean matrix $\mathbf{A}$, a Boolean vector $\mathbf{B}$, and a constant, $c \in \{0, 1\}$. The submatrices of $\mathbf{A}$ which are permutation or rotation matrices are also enumerated [54].

2.9.3 Large block ciphers considered during Phase II

Table 2.18 highlights the best attacks known on the block ciphers considered in this section. It identifies some unusual features of each cipher and some of its potential weaknesses, and gives a list of the best attacks, including the number of rounds broken, data and time complexities, and references for the attacks.

2.9.4 64-bit block ciphers not selected for Phase II

Table 2.19 highlights the best attacks known on the block ciphers considered in this section, identifying the number of rounds over which they operate, some unusual features of the cipher, some potential weaknesses of the cipher, and a list of the best attacks, including the number of rounds broken, data and time complexities, and references for the attacks.

2.9.5 128-bit block ciphers not selected for Phase II

Table 2.20 highlights the best attacks known on the block ciphers considered in this section, identifying the number of rounds over which they operate, some unusual features of the cipher, some potential weaknesses of the cipher, and a list of the best attacks, including the number of rounds broken, data and time complexities, and references for the attacks.

Table 2.14. A summary of each 64-bit block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
IDEA	8.5	Juxtaposition of 3 dissimilar algebraic groups, $+ \mod 2^{16}$, $\times \mod 2^{16}+1$ and $\oplus \mod 2$. No S-boxes Encryption $\simeq$ Decryption Widely studied for over a decade — few security flaws	Homomorphisms from $\times \mod 2^{16}+1$ to $+ \mod 2^{16}$ or $\oplus \mod 2$ (partial) Linear Key Schedule (weak) Relatively large weak-key classes	2	Diff	2^{42}	2^{10}	[362]
				2	Square	23	2^{64}	[158]
				2.5	Square	2^{58}	3.2^{16}	[380]
				2.5	Square	2^{79}	2^{48}	[380]
				3	Diff-Lin	$3 \cdot 2^{42}$	2^{29}	[77]
				3.5	Miss-in-the-middle	2^{53}	238.5	[58]
				3.5	Square	2^{82}	2^{34}	[158]
				4	Miss-in-the-middle	2^{70}	2^{38}	[58]
				4	Diff-Lin (related-key)	38	38.3	[244]
				4	Square	2^{34}	2^{114}	[158]
				4.5	Miss-in-the-middle	2^{112}	2^{64}	[58]
				4.5	Boomerang 2^{101} weak keys	2^{18}	2^{18}	[77]
				5	Boomerang 2^{95} weak keys	4	4	[77]
				8.5	Boomerang 2^{53} weak keys	4	4	[77]
				8.5	Boomerang 2^{64} weak keys	2^{16}	2^{16}	[77]
Khazad	8	All components involutions Efficient Diffusion Very high diffusion branch by use of linear transformation (MDS code) S-box does not depend on simple mathematical function. Encryption = Decryption	Slightly weak S-box nonlinearity	3	Square	2^{16}	2^{8}	[30]
				3	Imp. Diff.	2^{64}	2^{13}	[66]
				4	Square	2^{80}	2^{9}	[30]

Table 2.14. (continued) A summary of each 64-bit block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
MISTY1	8	Entire algorithm built from recursive components. Hard to analyse. Additional FL layers. S-boxes have irregular 7/9 splitting. Simple ANF for S-boxes S-boxes have optimal nonlinearity. Key schedule uses S-boxes Provable security against diff/lin. crypt. Encryption $\simeq$ Decryption	Low algebraic degree of S-boxes. Simple key schedule. Hardware/software penalty through 7/9 S-box splitting. Generalised linear approx. Characteristics not optimal. Complicated.	4	Slicing/Diff (FL)	$2^{81.6}$	2^{27}	[319]
				4	Collision (FL)	2^{89}	2^{20}	[318]
				4	(FL)	$2^{90.4}$	2^{23}	-
				4	(FL)	2^{62}	2^{38}	-
				4	(FL)	2^{89}	2^{20}	-
				4	(FL)	2^{76}	2^{28}	-
				4	(FL)	2^{48}	2^{34}	-
				5	High Diff (No FL)	2^{17}	$11 \cdot 2^7$	[502]
				5	High Diff (No FL)	2^{38}	2^6	[500]
				6	Diff (No FL)	2^{106}	2^{39}	[318]
				6	(No FL)	2^{61}	2^{54}	-
				6	Integral (FL)	2^{71}	2^{34}	[311]
MISTY2 (not part of NESSIE)	-			5	High Diff (No FL)	2^{39}	2^7	[500]
				5	Diff (FL)	2^{62}	2^{38}	[318]
				5	Collision (FL)	2^{76}	2^{28}	[318]
				6	Integral (FL)	2^{71}	2^{34}	[311]
KASUMI (not part of NESSIE)	-			4	Diff (No FL)	2^{22}	2^{10}	[503]
				5	Square (FL)	2^{80}	2^{38}	3GPP
				5	Lin. (FL)	2^{95}	2^{58}	3GPP
				6	Rel Key (FL)	2^{112}	$3 \cdot 2^{17}$	[91]
				6	Imp Diff (Rnds 2-7,FL)	2^{100}	2^{55}	[318]

Table 2.14. (continued) A summary of each 64-bit block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks Rnds	Technique	Time	Data	Ref.
SAFER$_{++64}$	8	4-point PHT for diffusion Two incompatible group operations, $\oplus$ and +, mod 256. S-boxes use exp and log. Key addition uses both $\oplus$ and + mod 256. Mini versions possible for analysis. Use of bias words for key schedule	Non-homomorphic linear approx. Rather small bit and byte branch numbers for diffusion layer. S-box using 45^x exhibits linear relationship with prob. 1	2 3	Lin Lin weak key frac: 2^{-6}	2^{42} 2^{121}	2^{5} 2^{33}	[382] [382]
Triple-DES (three key)	48	DES analysed for many years with no significant security flaws. Extension of DES. Backwards compatible with DES.	S-boxes are not optimally nonlinear. Lin. Crypt. Inefficient.	 8 8 9 10	2^{28} related-keys Improved m.i.t.m. Diff-Lin Diff-Lin Diff-Lin Diff-Lin Related-key	2^{84} $1.3 \cdot 2^{104}$ $2^{14.8}$ $2^{29.17}$ 2^{50}	2^{28} 2^{32} $2^{14.8}$ $2^{15.75}$ 2^{20}	[52] [340] [327] [62] [62] [62] [288]
(two key)					Meet-in-the-middle Chosen plaintext	2^{112} 2^{56}	3 2^{56}	[366] [367]

Table 2.21 shows affine bit-relations between output bits of the S-boxes, i.e. the number of equations of the form, $b_i(x) = b_j(\mathbf{A}x + \mathbf{B}) + c$ for the vector of S-box input bits, x, a Boolean matrix $\mathbf{A}$, a Boolean vector $\mathbf{B}$, and a constant, $c \in \{0, 1\}$. The submatrices of $\mathbf{A}$ which are permutation or rotation matrices are also enumerated [54].

Table 2.15. Affine relations for the S-boxes of some Phase II 64-bit block ciphers

			Sbox			Inverse Sbox		
Cipher	Sbox	Size	# Aff. Eqns	Perm	Rot	Total	Perm	Rot
Khazad-tweak		8×8	-	-	-	-	-	-
Khazad-orig		8×8	-	-	-	-	-	-
MISTY1	S7	7×7	> 2000	-	-	> 2000	-	-
	S9	9×9	> 100000	20	32	720	-	-
Safer++	exp45	8×8	256	-	-	(log45) 7	-	-
DES	S1	6×4	1	-	-	not invertible		
	S2	6×4	3	-	-	not invertible		
	S3	6×4	4	-	-	not invertible		
	S4	6×4	28	-	-	not invertible		
	S5	6×4	3	-	-	not invertible		
	S6	6×4	3	-	-	not invertible		
	S7	6×4	6	1	-	not invertible		
	S8	6×4	-	-	-	not invertible		

Table 2.16. A summary of each 128-bit block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
Camellia	18	Byte-Oriented, Feistel	x^{-1} S-box function	5	Imp Diff			[57]
		Use of x^{-1} for S-box	potentially open to	6	Square (FL)		2^{16}	[523]
		FL/FL^{-1} layers	algebraic attacks e.g.	6	Square (No FL)	2^{112}	$2^{11.7}$	[248]
		with 1-bit rot.	quadratic expressions,	7	Imp Diff (No FL)			[501]
			and affine relationship	8	Trunc Diff (No FL)	$2^{55.6}$	$2^{83.6}$	[330]
			between S-box o/ps	9	Square+key (FL)	2^{202}	2^{60}	[523]
			Identical rounds lead	9	Diff Crypt (No FL)		2^{105}	[59]
			to Slide attacks	9	Boomerang (FL)	2^{170}	2^{124}	[483]
				10	Rectangle (FL)	2^{241}	2^{127}	[483]
				11	Diff (No FL)	2^{232}	2^{104}	[483]
				11	High Diff (No FL)	2^{255}	2^{21}	[243]
				11	High Diff (FL)	2^{256}	2^{93}	[243]
				12	Lin (No FL)	2^{247}	2^{119}	[483]
RC6	20	Very simple description	Vulnerable via approx.	16	Diff. Crypt.		2^{128}	[121]
		Progression from RC5	across data-dep. rots.	16	Lin. Crypt.		2^{119}	[121]
		Data-dependent rots.	Small safety margin	8	Lin. Crypt.		2^{47}	[273]
		Quadratic function for		14	mult. lin. crypt.	2^{186}	2^{120}	[474]
		diffusion.		18	mult. lin. crypt.	2^{193}	2^{127}	[474]
		Strong, complex key-			with 2^{-90} weak key fraction			
		schedule.		15	χ^2 (stat.) attack			[306]
		Mini-versions possible.			by fixing 5 lsbs in A,C			
		Uses 32-bit mult. mod 2^{32}		17	χ^2 (stat.) attack			[306]
		Feistel-like			for 2^{-80} keys			
		Encryption $\neq$ Decryption			Extrapolated			
					experimentally			

Table 2.16. (continued) A summary of each 128-bit block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
Rijndael	10,12,14	Very high diffusion Use of x^{-1} for S-box Encryption $\neq$ Decryption Diffusion via MDS trans. Highly elegant Provable resistance to Diff/Lin. Crypt. Implicit Nested SPN structure	Small safety margin Potentially open to algebraic attacks e.g. quadratic expressions, and affine relationship between S-box o/ps Unnecessarily weak key-schedule. Description using BES Too elegant to be random?	5	Imp Diff	2^{31}	$2^{29.5}$	[66]
				6	Square	2^{72}	2^{32}	[150]
				7	Collision (192,256)	2^{140}	2^{32}	[219]
				7	Collision (128-bit key)	2^{128}	2^{32}	[219]
				7	Square (192-bit key)	2^{176}	2^{32}	[341]
				7	Square (256-bit key)	2^{192}	2^{32}	[341]
				6	Square	2^{44}	2^{32}	[191]
				7	Square (192-bit key)	2^{155}	2^{32}	[191]
				7	Square (256-bit key)	2^{172}	2^{32}	[191]
				7	Square	2^{120}	2^{128}	[191]
				8	Square (192-bit key)	2^{188}	$2^{119} - 2^{128}$	[191]
				8	Square (256-bit key)	2^{204}	$2^{119} - 2^{128}$	[191]
				9	Related-key/Square 256 related keys (256-bit keys)	2^{224}	2^{77}	[191]
SAFER$_{++128}$	8	4-point PHT for diffusion Two incompatible group operations, $\oplus$ and $+$, mod 256. S-boxes use exp and log. Key addition uses both $\oplus$ and $+$ mod 256. Mini versions possible for analysis. Use of bias words for key schedule	Non-homomorphic linear approx. Rather small bit and byte branch numbers for diffusion layer. S-box using 45^x exhibits linear relationship with prob. 1 Decryption diffusion weak	2.75	Imp. Diff.	2^{60}	2^{64}	[383]
				3.25	Square	2^{70}	$2^{9.6}$	[379]
				3.25	Lin. Crypt.	2^{103}	2^{81}	[382]
				4	(256-bit key) weak key frac: 2^{-13}	2^{178}	2^{81}	[382]
					Lin (256-bit key) weak key frac: 2^{-11}	2^{167}	2^{91}	[382]
				4	Integral (128-bit key)	2^{112}	2^{64}	[433]
				4	Integral (128-bit key) (less memory)	2^{120}	2^{64}	[433]
				4	Integral (256-bit key)	2^{144}	2^{64}	[433]
				4	Boomerang (128-bit key)	2^{41}	2^{41}	[75]
				5	Boomerang (128-bit key)	2^{75}	2^{75}	[75]
				5.5	Boomerang (128-bit key)	2^{121}	2^{121}	[75]

Table 2.17. Affine relations for the S-boxes of some Phase II 128-bit block ciphers

			Sbox			Inverse Sbox		
Cipher	Sbox	Size	# Aff. Eqns	Perm	Rot	Total	Perm	Rot
Camellia		8×8	504	-	-	504	-	-
AES-Rijndael		8×8	504	-	-	504	-	-
Safer++	exp45	8×8	256	-	-	(log45) 7	-	-

Table 2.18. A summary of each large block cipher selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
RC6	20	Very simple description Progression from RC5 Data-dependent rots. Quadratic function for diffusion. Strong, complex key-schedule. Mini-versions possible. Uses 32-bit mult. mod 2^{32} Feistel-like Encryption $\neq$ Decryption	Vulnerable via approx. across data-dep. rots. Small safety margin	$3+2s$	χ^2 (stat.) attack Extrapolated experimentally		2^{16+20s}	[301]
Rijndael	14	Wide-Trail Strategy Use of x^{-1} for S-box Encryption $\neq$ Decryption Diffusion via MDS trans. Highly elegant Provable resistance to Diff/Lin. Crypt. Implicit Nested SPN structure	Small safety margin Potentially open to algebraic attacks e.g. quadratic expressions, and affine relationship between S-box o/ps Unnecessarily weak key-schedule. Description using BES Too elegant to be random?					
SHACAL-1	80 (steps)	SHA well studied. Derived from hash fn. uses + mod 2^{32}, AND, OR, and data rots.	Potentially weak key-schedule (slide property)	41 47 49	Diff. Amp. Boomerang Rectangle	2^{491} $2^{508.4}$ $2^{508.5}$	2^{141} $2^{158.5}$ $2^{151.9}$	[290] [290] [68]
SHACAL-2	64 (steps)	Derived from hash fn. uses + mod 2^{32}, AND, OR, and data rots.						

Table 2.19. A summary of each 64-bit block cipher not selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
CS-Cipher	8	SPN, nonlinear diff FFT-based Diffusion	-	5.5	Lin/Diff	-	-	[198]
Hierocrypt-L1	6	Hierarchical SPN MDS Diffusion Feistel key schedule	Weak key-schedule	2.5 3 3.5 5	Integral Integral Integral Trunc Diff Key-schedule	2^{72} 2^{40} 2^{104}	2^{32} 6×2^{32} 14×2^{32}	[412] [412] [33] [412] [210]
Nimbus	5	Not SPN, Mult, mod 2^{64}	High prob. Diff.	5	Diff	2^{10}	2^{8}	[63]
Nush	9	Four iterations/round Feistel-like, No S-box	High lin. bias	-	Lin	2^{K-1}	-	[516]

Table 2.20. A summary of 128-bit block ciphers not selected for Phase II

Cipher	Rnds	Unusual Features	Potential Weaknesses	Best Attacks				
				Rnds	Technique	Time	Data	Ref.
Anubis	$8+N$	Like Khazad/Rijndael		5	Imp Diff	2^{31}	$2^{29.5}$	[29]
		Uses involutions		6	Saturation (all keys)	6×2^{48}	6×2^{32}	[29]
		Diffusion uses		7	Saturation (all keys)	2^{120}	2^{128}	[29]
		matrix transposition		7	Gilbert-Minier (key-size > 140)	2^{140}	2^{32}	[29]
				8	Saturation (key size > 204)	2^{204}	2^{128}	[29]
Grand Cru	10	Enhanced Rijndael						
		Multi-layered security						
Hierocrypt-3	6-8	Hierarchical SPN	Weak key-schedule	2.5	Integral	2^{168}	2^{13}	[412]
		MDS Diffusion		3	Integral	2^{40}	6×2^{32}	[412]
		Feistel key schedule		3.5	Integral	2^{168}	22×2^{32}	[33]
Noekeon	16	four 32-bit words	S-box includes		Diff/Lin			[308]
		Two key schedules	identity funcs					
			many related keys					
Nush		Four iterations/round	High lin. bias	-	Lin	2^{K-1}	-	[516]
		Feistel-like, No S-box						
Q	8	4×4-bit data blocks	High Diff/Lin prob.	8	Diff			[64]
				8	Lin (128-bit key)	2^{77}	2^{105}	[286]
				9	Lin (192-bit key)	2^{96}	2^{125}	[286]
				9	Lin (256-bit key)	2^{128}	2^{125}	[286]
SC2000	6.5 - 7.5	Feistel/SPN	High Diff Prob.	3.5	Diff			[177]
		4×4, 5×5, 6×6 S-boxes		4.5	Diff			[475]
		Strong key-schedule		4.5	Diff			[449]

Table 2.21. Affine relations for the S-boxes of some 128-bit block ciphers not selected for Phase II

			Sbox			Inverse Sbox		
Cipher	Sbox	Size	# Aff. Eqns	Perm	Rot	Total	Perm	Rot
Hierocrypt		8×8	504	-	-	504	-	-
SC2000	S4	4×4	> 1000	7	-	> 1000	7	-
	S5	5×5	> 1000		-	> 3000	8	-
	S6	6×6	210		-	210	8	-

Changes from version 1.0 to version 2.0 of the document

— Typos have been corrected.
— Qualitative statements on performance and selection criteria have been removed.
— Some of the tables at the end of the chapter rotated and split so as to be enlarged but still fit on the page.
§2.2.2 Added a few paragraphs to more formally define a block cipher, and to describe distinguishing attacks on block ciphers when viewed as pseudo-random functions.
§2.2.3.17 Included amendment by Courtois to say that the security PROBABLY does not grow exponentially with the number of rounds. Also included Bart Preneel and Lars Knudsen's view that less-than-exponential growth is probably not possible.
§2.3.11 ISO standard now includes Khazad and CAST-128, as pointed out by Barreto.
§2.5.1.2 Mention is made of Shirai's paper on Camellia from Munich, 2002, and some of his results on Camellia added to the table at the end.
§2.5.3.2 Reference made to 6-round impossible differential attack on Rijndael by Cheon *et al.* . Pointed out by Raphael Chung-Wei Phan.
§2.5.3.2 Correction to text wrt Rijndael and the alternating group, as suggested by Wernsdorf.
§2.2.3.14 A few comments added on the Slide attack.
§2.5.4.2 Text from the SAFER$_{++64}$ security analysis section which is also relevant for the SAFER$_{++128}$ security analysis section has been copied across, largely verbatim, to the SAFER$_{++128}$ security analysis section.
§2.5.4.2 Added Alex Biryukov's SASAS comment about SAFER$_{++128}$ to the Security Analysis section.
§2.5.4.2 Added a table to the SAFER$_{++128}$ security analysis section with Jorge Nakahara's attack results.

3. Stream ciphers

3.1 Introduction

A stream cipher is an algorithm for encrypting a sequence of elements or characters from a plaintext alphabet, usually the binary alphabet $\{0, 1\}$. Stream ciphers are commonly classified as being *synchronous* or *self-synchronising*. In a *synchronous* stream cipher the keystream is generated independently of the plaintext and ciphertext, so the keystream depends only on the key. In contrast, the keystream of a *self-synchronising* stream cipher depends on the key and a fixed amount of the previously generated ciphertext. Most stream ciphers can be classified as *additive* stream ciphers. An additive stream cipher is a synchronous cipher in which the ciphertext is the XOR of the plaintext and the keystream. None of the submissions is a self-synchronising stream cipher, therefore only synchronous stream ciphers are considered.

In specific applications, stream ciphers are more appropriate than block ciphers:

- Stream ciphers are generally faster than block ciphers, especially in hardware.
- Stream ciphers have less hardware complexity.
- Stream ciphers process the plaintext character by character, so no buffering is required to accumulate a full plaintext block (unlike block ciphers).
- Synchronous stream ciphers have no error propagation.

Most stream ciphers are based on simple devices that are easy to implement and run efficiently. A common example of such a device is the *linear feedback shift register* (LFSR) [457]. Such simple devices produce predictable output given some previous output. Thus, the output of such devices is typically used as the input to a function that produces the keystream. Keystreams can also be produced by using certain modes of operation of a block cipher.

Many of the common uses of stream ciphers require frequent key reinitialization or rekeying. A full definition of a stream cipher intended for such uses should give details of how the cipher should be rekeyed. The original NESSIE call for primitives did not require stream ciphers to be accompanied by a rekeying schedule. Only the two SOBER stream ciphers provided a rekeying schedule with the original schedule, though rekeying schedules have subsequently been provided for the other submissions.

[0] Coordinator for this chapter: SAG — Marcus Schafheutle, Stefan Pyka

3.2 Security requirements

The techniques used to analyse stream ciphers use mathematical and statistical properties of the generator or approximations to it. In particular, the keystream generator should produce a memoryless balanced sequence of bits, which are modelled as a sequence of independent identically distributed Bernoulli random variables with parameter 0.5 (fair coin tosses). Stream cipher analysis is essentially concerned with analysing the keystream generator to find deviations from this statistical model. It is customary when analysing stream ciphers to consider known plaintext attacks. This essentially means assuming that a large amount of keystream is known. The statistical deviations are exploited to give methods for attacking the stream cipher based on the known keystream. Such methods are usually classified in one of the following three ways:

1. *Distinguishing Attack.*
   A method for distinguishing output from the keystream generator from a 'random' sequence of the same length.
2. *Prediction.*
   A method for predicting output from the keystream generator more accurately than guessing
3. *Key Recovery.*
   A method for recovering the key from the output of the keystream generator.

Key recovery is clearly the most powerful of these three methods as it enables both prediction and a distinguishing attack. Prediction also clearly enables a distinguishing attack. However, a distinguishing attack can be thought of as a type of prediction. This is because a distinguishing attack makes statements that certain sequences from a keystream generator are more or less likely to occur than they would if produced 'at random', thus making predictions about the keystream sequence that are more accurate than guessing.

3.2.1 Classification of attacks

Stream ciphers tend not to use iterated functions in the same way as block ciphers, so the classification of attack techniques is more difficult. However, the most common techniques are discussed below.

Exhaustive key search

This attack is the most general type of attack that can be applied to any stream cipher. Given a keystream sequence generated by an unknown key, an attacker simply tries all possible keys and checks whether the generated keystream matches the given keystream sequence. For exhaustive key search for stream ciphers there exist very efficient time-memory tradeoff techniques as described in [78]. In a time-memory tradeoff attack, some key-output relations are precomputed and stored in memory. In the real-time phase of the attack, the given output data is searched until a stored output pattern is found. The attacker has then found the corresponding key. The time complexity for exhaustive key search is split into a

time and a memory complexity. The name 'time-memory tradeoff' results from this idea.

Periodic and Statistical Attacks

If the period of a keystream generator is too small, then the keystream will repeat itself, so enabling easy prediction. The period must be large enough to ensure that the keystream is not repeated. More generally, if the keystream deviates in an obvious way from the memoryless Bernoulli distribution discussed above, then there is an obvious prediction technique available.

Linear Complexity

The linear complexity of a sequence is the length of the shortest LFSR that can produce that sequence. The linear complexity of a sequence is easily calculated using the Berlekamp-Massey algorithm [352]. If this linear complexity is too small, then an attacker can reproduce the sequence on an LFSR.

Maximum Order Complexity

The Maximum Order Complexity (MOC) Test determines the length of the shortest possibly non-linear feedback shift register which can produce the given bit sequence For the MOC profile, this is done for the first 1,2,3.. bits of the sequence.

Correlation Attacks

Correlation attacks are the most important general attacks on LFSR-based stream ciphers. In a correlation attack, the output of a keystream generator is correlated in some manner with the output of a much simpler device, such as a component LFSR of the generator. This correlation can sometimes be exploited to determine the key. The first ideas were described by Thomas Siegenthaler [490]. Meier and Staffelbach [363] and others subsequently improved these ideas by developing fast correlation attacks.

Higher Order Correlation attacks

Many stream ciphers are built of a linear sequence generator and a non-linear output function f. Correlation attacks try to find a linear approximation of f. Equivalenty higher order approximations are possible. With the aid of the approximations an overdefined system of multivariate equations can be defined. The XL method [137] can be adapted to solve these equations. This kind of attack is not well examined, but a more detailed description can be found in [130].

Divide-and-Conquer Attacks

In such attacks a portion of the key (or of the internal state) is guessed. The constraints now placed on the keystream may allow the determination of the remainder of the key faster than searching this remainder exhaustively.

Rekeying Attacks

There are many applications in which a stream cipher is frequently rekeyed. It is sometimes possible to exploit this rekeying in order to find the key.

Sidechannel attacks

Using sidechannel attacks one exploits the behaviour of primitives in use, for example different performances or different power consumption for different keys or internal states. For a complete description of sidechannel attacks see the annex.

3.2.2 Assessment process

The stream cipher submissions were assessed with reference to the above generic common stream cipher attack techniques. Furthermore, some stream cipher submissions were analysed using techniques specific to that primitive.

Statistical testing is a vital part of stream cipher analysis in order to provide assurance that the generator possesses the required statistical properties. The NESSIE toolbox provides extensive tools for stream cipher and block cipher testing. A list of these tools used in stream cipher assessment is given below, and further details can be found in [400]. It is emphasized that properties like large period, large linear complexity and a good statistical behaviour are necessary but not sufficient conditions for a stream cipher to be considered cryptographically secure.

3.2.2.1 The NESSIE statistical toolbox for stream ciphers

The NESSIE toolbox provides several tests in order to assess the statistical properties of output sequences from the keystream generator of a stream cipher:

- Collision Test
  The collision test splits up the bit sequence into blocks of a fixed size. A collision occurs if the same block appears more than once. The test statistically evaluates the number of collisions.
- Constant Runs Test
  For the constant runs test, the sequence of bits is subdivided into runs, that is maximal disjoint subsequences of consecutive 0s and 1s. The frequencies of these runs of the various lengths are evaluated statistically.
- Correlation Test
  The correlation test statistically evaluates the correlation between a sequence and shifts of the sequence.
- Coupon Collector's Test
  The coupon collector's test splits up the bit sequence into blocks of a fixed size. The test statistically evaluates the number of blocks required until all possible blocks have appeared. The coupon test is also applied to cyclic shifts of the original sequence.
- Dyadic Complexity Test
  The dyadic complexity test is an implementation of the complexity measure suggested by Goretzky and Klapper [291] for sequences of bits. This measure is cryptologically relevant because feedback shift registers with carry, also described in [291], have low dyadic complexity.
- The Fast Spectral Test
  The fast spectral test applies the fast Walsh transform to the bit sequence. It

uses two values derived from the transform to assess the randomness of the sequence.

- Frequency Test
  The frequency test splits up the bit sequence into blocks of a fixed size. The frequencies of these blocks are evaluated statistically.
- Gap Test
  The gap test splits up the bit sequence into blocks of a fixed size. The blocks are interpreted as binary representations of numbers, to give a sequence of numbers. A gap is a maximal subsequence containing no members in a certain numerical range, and the lengths of gaps are evaluated statistically. This test is also applied to cyclic shifts of the original sequence.
- Linear Complexity Test
  The linear complexity test uses the Berlekamp–Massey algorithm to determine the length of the shortest linear feedback shift register which can produce the given bit sequence. The linear complexity profile is also evaluated.
- Maximum Order Complexity Test
  The maximum order complexity test determines the length of the shortest possibly nonlinear feedback shift register which can produce the given bit sequence. For the maximum order complexity profile, this is done for the first 1,2,3... bits of the sequence.
- Overlapping m-tuple Test
  The overlapping m-tuple test splits up the bit sequence into overlapping subsequences of length m. The frequency of these (dependent) subsequences is evaluated statistically. This test is also applied to cyclic shifts of the original sequence.
- Percolation Test
  The percolation test is the simulation of a forest fire. The bit sequence to be tested determines where trees are standing in the simulated forest. The test evaluates statistically how fast a fire propagates in the simulated forest.
- Poker Test
  The poker test splits up the bit sequence into groups of k successive blocks of a fixed size, known as (poker) hands. The poker test statistically evaluates the frequencies of these hands. This test is also applied to cyclic shifts of the original sequence.
- Rank Test
  In the rank test, the bits of the sequence to test are used to fill square matrices. The bits are treated as elements of the field GF(2), and the ranks of the matrices are evaluated statistically.
- Run Test
  The run test splits up the bit sequence into blocks of a fixed size. These blocks are interpreted as binary representations of numbers, to give a sequence of numbers. A run is a maximal subsequence of strictly increasing numbers, and the lengths of runs are evaluated statistically.
- Universal Maurer Test
  The universal Maurer test splits up the bit sequence into disjoint subsequences of bits. The test statistically evaluates the distances between identical subse-

quences. The test result of the Maurer test is closely related to the entropy of the bit sequence.
– Ziv-Lempel Complexity Test
The Ziv-Lempel complexity test is based on a measure of the rate at which new patterns occur in the sequence.

These tests were applied to all of the stream cipher submissions, but no submission exhibited anomalous behaviour. Detailed results of the statistical tests are available as NESSIE public reports.

Some block cipher tests were also used for stream cipher analysis. As an example, the following tests were applied to analyse the key loading (initialization vector loading) process used in the stream ciphers SNOW, SOBER-t16 and SOBER-t32:

– Dependence Test
The dependence test evaluates the dependence matrix and the distance matrix of a function. Furthermore, the degree of completeness, the degree of avalanche effect and the degree of strict avalanche criterion of the function are computed.
– Linear Factors Test
The linear factors test is used to find out whether there are any linear combinations of output bits which, for all keys and plaintexts, are independent of one or more key or plaintext bits. Such a linear combination is called a linear factor.

These tests detected linearity properties in the key loading of SOBER-t32 [166].

3.3 Overview of the common designs

In this section we give an overview of stream cipher design techniques which are most commonly used today in practice. Because of the broad field of stream cipher design, only a brief overview is presented in order to give a rough classification of the stream cipher submissions.

3.3.1 Stream ciphers based on feedback shift registers

Feedback shift registers, in particular LFSRs, are widely used as building blocks for stream ciphers. LFSRs produce sequences having large periods and good statistical properties, they are well-suited for hardware implementations and there are mathematical techniques to analyse them. Unfortunately, the output sequence of an LFSR is linear and so is easily predictable. When LFSRs are used as components for keystream generators, it is very important that the output sequence does not inherit linearity properties from the output sequences of the component LFSRs. Some methodologies with this objective are briefly discussed in the next sections.

The submitted ciphers SNOW, LILI-128, SOBER-t16 and SOBER-t32 are all based on LFSRs.

Nonlinear combination generators

A nonlinear combination generator uses a number of LFSRs. The keystream is generated as a nonlinear function f of the outputs of these LFSRs.

Nonlinear filter generators

In this construction, the keystream is generated as a nonlinear function f of the stages of a single LFSR.

Clock-controlled generators

An irregularly clocked LFSR does not exhibit the same linearity properties as one that is regularly clocked. Thus a common technique is to use one LFSR sequence to control the clocking of another LFSR. A more general form of clock-control is the irregular decimation of the output sequence of one device by another device. For each generated sequence bit from the first device, the second device decides whether the generated bit should be used as a keystream bit or discarded.

The stream cipher submissions LILI-128, SOBER-t16 and SOBER-t32 use LFSRs, nonlinear filters and irregular clocking as main components.

3.3.2 Stream ciphers based on block ciphers

Some modes of operation of block ciphers can be used to generate a keystream sequence, such as the Output Feedback (OFB) Mode, the Cipher Feedback (CFB) Mode and the Counter Mode (CTR). The BMGL stream cipher submission to NESSIE is in reality a block cipher mode of operation. Note that stream ciphers based on block cipher modes of operation can potentially be attacked by cryptanalysis of the underlying block cipher. There are also generic distinguishing attacks on block ciphers in OFB and Counter Mode. For a block cipher with block size b, $2^{b/2}$ blocks of keystream are sufficient to distinguish the keystream from a truly random sequence. This is achieved by looking for repeated occurrences of blocks, which are not possible when the stream is generated by a block cipher in OFB or Counter Mode (unless the sequence has started to repeat itself).

3.3.3 Pseudorandom number generators based on modular arithmetic

The security of these generators is based on the presumed intractability of an underlying number-theoretic problem. Popular examples of this class are the RSA generator and the Blum-Blum-Shub generator [89]. The required modular arithmetic makes these generators extremely slow compared to other keystream generators, so such generators are primarily used as pseudorandom number generators.

3.3.4 Other stream ciphers

Stream ciphers based on LFSRs are well-suited for hardware. Some recent stream ciphers have been designed particularly for efficient software implementation,

and are not based on LFSRs. Examples include the stream ciphers RC4 [451], SEAL [453] and SCREAM [235], and the NESSIE stream cipher submission LEVIATHAN.

3.3.5 Current standards

At present a standard for dedicated stream ciphers, such as the AES standard for block ciphers, does not exist. One probable reason is that most stream ciphers in use are either secret or proprietary designs.

The standard ISO 10116 (2nd edition) specifies modes of operation for 64-bit block ciphers that give keystream generators, in particular the Cipher Feedback Mode and the Output Feedback Mode. Several other standards (ANSI X9.52, FIPS 81) specify modes of operation for the DES and triple-DES cipher. Recently, the National Institute of Standards and Technology (NIST) held two public workshops on block cipher modes of operation. Some of the proposed modes are suitable for stream cipher design, for example the Counter Mode and the Key Feedback Mode (BMGL is based on this mode). In the NIST publication SP800-30A, modes of operation that could be used as keystream generators, such as Cipher Feedback Mode, Output Feedback Mode and Counter Mode are recommended. However, as described in section 3.3.2, block ciphers in OFB and Counter Mode might be vulnerable to distinguishing attacks. When speaking of OFB mode, one usually considers the OFB mode with full feedback in contrast to the OFB mode with r-bit ($r < n$) feedback. With full feedback the complete output word from the last encryption step is used for feedback, while with r-bit feedback only r bits are used for feedback. In the first case the feedback function can be treated as a random permutation with expected cycle length of about 2^{n-1}, while in the second case the feedback function is a random function with expected cycle length $2^{n/2}$. This is the reason why OFB is used in full feedback mode, and why it is vulnerable to distinguishing attacks. On the other hand there is a simple way to remove this drawback. If the block size is twice as big as the key size, then one needs as many output words as the key space size to mount a distinguishing attack, which is not better than brute force. So if one uses a 256-bit block cipher with a 128-bit key, then this configuration provides adequate security in the normal category.

3.4 Stream cipher primitives considered during Phase II

3.4.1 BMGL

BMGL is a stream cipher designed by Johan Håstad and Mats Näslund [241] and submitted to the NESSIE project. BMGL provides key sizes as for Rijndael (128-bit, 192-bit and 256-bit). Furthermore, the tweaked version of BMGL allows rekeying with an 128-bit initialization vector.

3.4.1.1 Design

The construction of BMGL is based on so called *hardcore functions* for one-way permutations and on the possibility to construct pseudo random number generators with the aid of them. Given a one-way function f, a set of binary functions $\{b_r\}$ is called a family of *hardcore functions*, if for any r the output $b_r(x)$ cannot be distinguished computationally from a random bit.

Given the seeds x_0 and r and the one-way permutation f, one can construct a random number generator g by $x_{i+1} = f(x_i)$ and $g(x_0, r) = b_r(x_1), b_r(x_2), \ldots$. One can show, that if there is an efficient algorithm D that distinguishes with non-negligible advantage $g(x, r)$ from a random string (with given r), then there is an efficient algorithm P that given r, $f(x)$ predicts $b_r(x)$ with non-negligible advantage [90]. Furthermore Goldreich and Levin [225] have shown, that in this case, there is an efficient algorithm B, that inverts $f(x)$ on random x with non-negligible probability. They also could prove, that for every one-way function f, *hardcore functions* exist. The set $\{b_r\}$ can be defined by the inner product $b_r(x) := x \cdot r \bmod 2$ of n-bit strings x and r. This also holds for extending the inner product on matrix products, in order to generate several bits at once. Let M_m^n be the set of all $m \times n$-matrices and let $R \in M_m^n$. Then one can define hardcore functions

$$B_R^m(x) = \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ r_{21} & r_{22} & \ldots & r_{2n} \\ \ldots & \ldots & & \ldots \\ r_{m1} & r_{m2} & \ldots & r_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}.$$

This results in the basic construction of BMGL:

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function and let n, m and λ be integers. Then the generator $BMGL_{n,m,\lambda m}(f)$ is defined as follows: The input of the generator is $x_0 \in \{0,1\}^n$ and $R \in M_m^n$. Let $x_i = f(x_{i-1})$. Then the output of the generator is defined by $\{B_R^m(x_i)\}_{i=1}^{\lambda}$.

In the BMGL submission for the NESSIE project the authors have defined the function f as the block cipher Rijndael. The construction of BMGL is based on theoretical results for one-way functions. Therefore the security of BMGL with Rijndael as one-way function has to be related to these theoretical results. The authors show, that if Rijndael is secure, then their construction of BMGL is secure.

In the BMGL submission Rijndael is used in a new mode of operation called *key feedback mode* (KFB). The idea is not to feedback the ciphertext as new plaintext input in the next iteration step, but as a new key. In some sense it is a "dual" to OFB mode. The plaintext and the random matrix R may be publicly known and need not be a part of the secret key.

3.4.1.2 Security analysis

The security of the construction results from the assumption that the one-way function used in the BMGL generator does not lose its one-wayness property

even if the function is iterated many times. Based on previous papers about the reduction of the security of generators of pseudorandom bits to the existence of one-way functions [225, 240], the designers show formally in the model of exact security that a non-trival attack on BMGL gives a black-box reduction to an attack on the underlying iterated one-way function, i.e. Rijndael. The analysis allows us to quantify the loss of security. The security and correctness of the overall construction has been verified several times before in the case of one-way permutations [225, 240, 223, 336, 222]. All proofs given in the submission were carefully checked.

In [242] the submitters generalize the generator in order to allow keystream synchronization with random access properties. Furthermore, they present a sketch for a security proof based on the assumption that the iterated Rijndael mapping is hard to invert even if an attacker has a number of extra plaintext-ciphertext pairs.

We note that BMGL is in reality a mode of use of a block cipher, in this case Rijndael. This means that certain "generic attacks" on modes of use are applicable as noted by Babbage [23]. The type of generic attacks described there is a time-memory tradeoff for stream ciphers, where the internal state size is not much bigger than the key space size. Babbage recommends that the internal state size of the key stream generator should be at least two times bigger than the key space size to prevent time-memory tradeoffs. For the stream cipher BMGL this is not the case.

3.4.2 SNOW

SNOW is a synchronous stream cipher designed by Patrick Ekdahl and Thomas Johansson [180] and submitted to the NESSIE project. It uses a 128-bit or 256-bit key and has an internal memory of 576 bits. The tweaked version of SNOW allows rekeying with a 64-bit initialization vector.

3.4.2.1 The design

SNOW consists of an LFSR and a Finite State Machine, the states of which are words in $GF(2^{32})$. We denote addition in $GF(2^{32})$ by the symbol $\oplus$, addition modulo 2^{32} by the symbol $+$, and the ith bit of the element x in the field by $x[i]$. SNOW uses an LFSR of length 16 defined by the recurrence relation

$$s_{t+16} = \alpha(s_t \oplus s_{t+3} \oplus s_{t+9})$$

where $\alpha \in GF(2^{32})$ is given in the NESSIE submission. The Finite State Machine consists of two registers whose values at time t we will denote by a_t and b_t respectively. Let

$$\begin{aligned} a_{t+1} &= a_t \oplus R(f_t + b_t) \\ b_{t+1} &= S(a_t) \\ f_t &= (s_{t+15} + a_t) \oplus b_t \\ z_t &= f_t \oplus s_t, \end{aligned}$$

where R denotes a 7-bit left (towards the most significant bit) rotation and S is a 32-bit to 32-bit S-box given in the NESSIE submission. The sequence (z_t) is used as the keystream. There is a key initialisation process described in the submission.

3.4.2.2 Security analysis

A distinguishing attack. Coppersmith, Halevi and Jutla [125] have observed that if $\sigma_t = s_{t+15}[15] \oplus s_{t+15}[16] \oplus s_{t+16}[22] \oplus s_{t+16}[23] \oplus f_t[15] \oplus f_{t+1}[23]$ then σ_t has bias $\epsilon = 2^{-8.3}$.

For a sequence (v_t) and polynomial $q(X) = \sum_{i=0}^{N} c_i X^i$, let $T_q^j(v_t)$ denote $\sum_{i=0}^{N} c_i v_{j+i}$. They also show that there are about 2^{100} weight six polynomials which are divisible by the linear feedback polynomial given above. If p is such a polynomial, then $T_p^j(s_t) = 0$ for each j. For each such p, $T_p^j(\sigma_t) = T_p^j(f_t[15]) \oplus T_p^j(f_t[23])$, so we can obtain $T_p^j(\sigma_t)$ since we know the f_t. However, $T_p^j(\sigma_t)$ has bias $\epsilon^6 = 2^{-49.8}$ since p has weight 6. As there are about 2^{100} such polynomials p, we can find 2^{100} such $T_p^j(\sigma_t)$, each with a bias of $2^{-49.8}$.

The mean of the sum of these 2^{100} values is $2^{99} + 2^{75}$ and the standard deviation approximately $2^{49} + 2^{24}$. The mean of a sum of 2^{100} 'random' values would be 2^{99} and the standard deviation 2^{49}. We can assume that the distributions of these sums are normal, so most (about 95%) realisations of these sums lie within two standard deviations of the mean. Thus it is clear we can distinguish the two distributions. This leads to a distinguishing attack on SNOW requiring 2^{95} observed bits of keystream and workload about 2^{100}.

Guess and determine attacks. Hawkes and Rose [246] present two attacks on SNOW, the first requiring 2^{64} observed bits of keystream with workload 2^{256} (so no faster than exhaustive key search) and the second 2^{224} observed bits of keystream with workload 2^{95}.

For the basic attack we fix t and assume that $b_t = S(a_t \oplus \mathbf{1})$ and $b_{t+14} = S(a_{t+14} \oplus \mathbf{1})$ where $\mathbf{1} = 2^{32} - 1$. We guess s_t, s_{t+1}, s_{t+2}, s_{t+3}, a_t and a_{t+14} (192 bits in total) and can then determine the shift register state based on these assumptions and test if this shift register state is correct by comparing its output with the keystream. If no guesses give the correct keystream for this value of t, we repeat this with another value of t. As the probability that these assumptions hold is 2^{-64}, we expect to have to try about 2^{64} values of t. The details of how to determine the shift register state from each guess are given in [246]. This gives an attack requiring 2^{64} observed bits of keystream with workload 2^{256}.

We can improve the workload of this attack by also assuming that a_{t+1} is either 0 or $\mathbf{1}$ as well as $b_t = S(a_t \oplus \mathbf{1})$ and $b_{t+14} = S(a_{t+14} \oplus \mathbf{1})$ as before. We guess s_t, s_{t+1}, a_t, a_{t+14} and whether a_{t+3} is 0 or $\mathbf{1}$ (129 bits in total) and can then determine the shift register state and test whether this state is correct. The probability that these assumptions are correct is 2^{-95}, so we expect to have to repeat this for about 2^{95} values of t. As before, the details of how to determine the shift register state from each guess are given in [246]. This gives us an attack requiring 2^{224} observed bits of keystream with workload 2^{95}.

3.4.3 SOBER-t16

SOBER-t16 is a synchronous stream cipher designed by Philip Hawkes and Greg Rose [245] and submitted to the NESSIE project. SOBER-t16 uses a 128-bit key and has an internal memory of 272 bits. SOBER-t16 allows rekeying with an initialization vector. Hawkes and Rose also submitted to NESSIE SOBER-t32, a similar stream cipher but with a 256-bit key (see Section 3.4.4).

3.4.3.1 Description of SOBER-t16

SOBER-t16 is based on an LSFR of length 17 over the field $GF(2^{16})$. This LFSR is 'stuttered' as described below. We represent elements of this field with 2^{16} elements by 16-bit binary vectors corresponding to polynomials modulo the irreducible polynomial

$$x^{16} + x^{14} + x^7 + x^6 + x^4 + x^2 + x + 1.$$

We denote addition in $GF(2^{16})$ by the symbol $\oplus$, addition modulo 2^{16} by the symbol $+$ and the ith bit of the element x in the field by $x[i]$.

The linear feedback shift register. SOBER-t16 uses an LFSR of length 17 over $GF(2^{16})$ given by the recurrence relation

$$s_{t+17} = \alpha s_{t+15} \oplus s_{t+4} \oplus \beta s_t$$

where $\alpha =$ `0XE382` and $\beta =$ `0X67C3`.

The nonlinear filter. If the shift register output is given by the sequence (s_t), then the nonlinear filter (NLF) used by SOBER-t16 is given by

$$v_t = ((f(s_t + s_{t+16}) + s_{t+1} + s_{t+6}) \oplus K) + s_{t+13}$$

where

$$f(a) = S(\bar{a}) \oplus (a - \bar{a})$$

and $\bar{a}$ denotes the 8 most significant bits of a, so $a - \bar{a}$ is the 8 least significant bits of a. The value K is a 16-bit key-dependent constant initialised by the key loading and S is an 8-bit to 16-bit substitution box specified in the submission.

The stuttering. This pair of shift register and nonlinear filter are stuttered as follows. The first output word v_1 of the nonlinear filter is the first 'stutter control word' and is partitioned into 8 pairs of bits. Starting with the least significant pair, these pairs determine the stuttering according to the table below, where $C =$ `0X6996`, and $\sim C$ is the bitwise complement of C. When all the pairs have been used, the shift register is clocked and the output of the nonlinear filter becomes the next stutter control word. This procedure is repeated until sufficient keystream has been produced.

00	Clock LFSR.
01	Clock LFSR. Output XOR of C with NLF output to the keystream. Clock LFSR.
10	Clock LFSR twice. Output the NLF output to the keystream.
11	Clock LFSR. Output XOR of $\sim C$ with NLF output to the keystream.

Key loading and rekeying. The 17 states $r_1, \ldots, r_{17}$ of the shift register are set to the first 17 Fibonacci numbers. The key loading uses two operations: the '`Include`' operation and the '`Diffuse`' operation. The `Include` operation consists of adding a given word to r_{15} modulo 2^{16}. The `Diffuse` operation consists of clocking the shift register and replacing r_4 with the XOR of r_4 with the nonlinear filter output. To load the key, the `Include` operation is applied with each key word in turn, with each `Include` being followed by the `Diffuse` operation. When this has been completed, the `Include` operation is then performed using the key length as input and the `Diffuse` operation is applied 17 times. The shift register is then clocked and K set to the nonlinear filter output.

It is also possible to rekey the cipher by using a public 'frame key'. This is done by loading the frame key after the secret key in the same manner.

3.4.3.2 Observations on SOBER-t16

We first consider a simplified version of SOBER-t16 in which the stuttering is not used. Ekdahl and Johansson [181] have found a distinguishing attack on this unstuttered SOBER-t16, which we now briefly describe. Let $w_t = v_t \oplus s_t \oplus s_{t+1} \oplus s_{t+6} \oplus s_{t+16} \oplus s_{t+13}$ and $W_t = w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t$. Then $W_t = z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t$ so we can obtain the sequence (W_t) from the keystream. The distribution of w_t (for fixed K) can be estimated by simulation and is non-uniform for all the values of K for which this has been done. The error in the distribution obtained this way can also be estimated. We can then use this distribution to calculate the distribution of W_t (for fixed K). The Neyman-Pearson lemma then tells us that (in the worst case scenario) we need 2^{92} keystream words to distinguish between keystream from the cipher and a random keystream with probability of error 2^{-32}. The computational complexity of this distinguishing attack is 2^{92}.

Ekdahl and Johansson [181] have also found a distinguishing attack on SOBER-t16 (including stuttering). Let $\mathcal{E}$ be the event that if $z_n = C_o \oplus v_t$ then $z_{n+2} = C_1 \oplus v_{t+4}$, $z_{n+7} = C_2 \oplus v_{t+15}$ and $z_{n+8} = C_3 \oplus v_{t+17}$, where $C_1, C_2, C_3 \in \{C, \sim C, 0\}$. We can calculate the most probable position in the keystream for each of $(v_t, v_{t+4}, v_{t+15}, v_{t+17})$ so we can show that $\mathcal{E}$ happens with probability $2^{-5.5}$. Let $W'_t = W_t \oplus C_3 \oplus \alpha C_2 \oplus C_1 \oplus \beta C_0$. If the event $\mathcal{E}$ happens, then $W'_t = z_{n+8} \oplus \alpha z_{n+7} \oplus z_{n+2} \oplus \beta z_n$ so we can calculate the sequence (W'_t) from the keystream. If we assume that the distribution is uniform when $\mathcal{E}$ does not occur, then we can calculate the distribution of W'_t. The Neyman-Pearson lemma then tells us that we need 2^{111} keystream words to distinguish between keystream from the cipher and a random keystream with probability of error 2^{-32}. The computational complexity of this distinguishing attack is 2^{111}.

Pyka [444] has independently found a bias in each bit of w_t (other than the 0th, 1st, 2nd and 4th bits) when $K = 0$. For $K \neq 0$ the correlations change their values depending on the bits of K. He also pointed out that for unstuttered SOBER-t16, the distribution of w_t for a given K can be calculated precisely, since the bias of $f(s_t + s_{t+16}) \oplus s_t \oplus s_{t+16}$ can be determined by looking at all the possibilities for s_t and s_{t+16}. The bias of each bit can then be determined by looking at the probabilities of the carry values.

Besides those attacks, SOBER-t16 is vulnerable to timing attacks and power attacks due to its irregular decimation [245].

3.4.3.3 Comments from the submitters

The submitters have made some comments on the relevance of these results about SOBER-t16 and about the results about SOBER-t32, discussed below. These comments are briefly summarised in Section 3.4.4.3, in the discussion of SOBER-t32.

3.4.3.4 Conclusions

There is a distinguishing attack on SOBER-t16, as well as a much faster one on the unstuttered version of SOBER-t16. The nonlinear filter also exhibits significant biases. Furthermore, SOBER-t16 is vulnerable to timing and power attacks.

3.4.4 SOBER-t32

SOBER-t32 is a synchronous stream cipher designed by Philip Hawkes and Greg Rose [245] and submitted to the NESSIE project. SOBER-t32 uses a 256-bit key and has an internal memory of 544 bits. SOBER-t32 allows rekeying with an initialization vector. Hawkes and Rose also submitted SOBER-t16 to NESSIE, a similar stream cipher but with a 128-bit key (see Section 3.4.3).

3.4.4.1 Description of SOBER-t32

SOBER-t32 is based on an LSFR of length 17 over the field $GF(2^{32})$. This LFSR is 'stuttered' as described below. We represent elements of this field with 2^{32} elements by 32-bit binary vectors corresponding to polynomials modulo the irreducible polynomial

$$x^{32} + (x^{24} + x^{16} + x^8 + 1)(x^6 + x^5 + x^2 + 1).$$

We denote addition in $GF(2^{32})$ by the symbol $\oplus$, addition modulo 2^{32} by the symbol $+$ and the ith bit of the element x in the field by $x[i]$.

The linear feedback shift register. SOBER-t32 uses an LFSR of length 17 over $GF(2^{32})$ given by the recurrence relation

$$s_{t+17} = s_{t+15} \oplus s_{t+4} \oplus \alpha s_t$$

where $\alpha =$ `0XC2DB2AA3`.

The nonlinear filter. If the shift register output is given by the sequence (s_t), then the nonlinear filter (NLF) used by SOBER-t32 is given by

$$v_t = ((f(s_t + s_{t+16}) + s_{t+1} + s_{t+6}) \oplus K) + s_{t+13}$$

where

$$f(a) = S(\bar{a}) \oplus (a - \bar{a})$$

and $\bar{a}$ denotes the 8 most significant bits of a, so $a - \bar{a}$ is the 24 least significant bits of a. The value K is a 32-bit key-dependent constant initialised by the key loading and S is an 8-bit to 32-bit substitution box specified in the submission.

The stuttering. This pair of shift register and nonlinear filter are stuttered as follows. The first output word v_1 of the nonlinear filter is the first 'stutter control word' and is partitioned into 16 pairs of bits. Starting with the least significant pair, these pairs determine the stuttering according to the table below, where $C =$ `0X6996C53A`, and $\sim C$ is the bitwise complement of C. When all the pairs have been used, the shift register is clocked and the output of the nonlinear filter becomes the next stutter control word. This procedure is repeated until sufficient keystream has been produced.

00	Clock LFSR.
01	Clock LFSR. Output XOR of C with NLF output to the keystream. Clock LFSR.
10	Clock LFSR twice. Output the NLF output to the keystream.
11	Clock LFSR. Output XOR of $\sim C$ with NLF output to the keystream.

Key loading and rekeying. The 17 states $r_1, \ldots, r_{17}$ of the shift register are set to the first 17 Fibonacci numbers. The key loading uses two operations: the '`Include`' operation and the '`Diffuse`' operation. The `Include` operation consists of adding a given word to r_{15} modulo 2^{32}. The `Diffuse` operation consists of clocking the shift register and replacing r_4 with the XOR of r_4 with the nonlinear filter output. To load the key, the `Include` operation is applied with each key word in turn, with each `Include` being followed by the `Diffuse` operation. When this has been completed, the `Include` operation is then performed using the key length as input and the `Diffuse` operation is applied 17 times. The shift register is then clocked and K set to the nonlinear filter output. It is also possible to rekey the cipher by using a public 'frame key'. This is done by loading the frame key after the secret key in the same manner.

3.4.4.2 Observations on SOBER-t32

We first consider a simplified version of SOBER-t32 in which the stuttering is not used. Ekdahl and Johansson [181] have found a distinguishing attack on this unstuttered version of SOBER-t32, which we now briefly describe. Let $w_t = v_t \oplus s_t \oplus s_{t+1} \oplus s_{t+6} \oplus s_{t+16} \oplus s_{t+13}$ (as for SOBER-t16). Although we cannot estimate the distribution of w_t, we can estimate the distribution of $w_t[i] \oplus w_t[i-1]$ for each i by simulation. It can be shown that $s_{t+\tau_5} \oplus s_{t+\tau_4} \oplus s_{t+\tau_3} \oplus s_{t+\tau_2} \oplus s_{t+\tau_1} \oplus s_t = 0$ where $\tau_1 = 11$, $\tau_2 = 13$, $\tau_3 = 4 \cdot 2^{32} - 4$, $\tau_4 = 15 \cdot 2^{32} - 4$, $\tau_5 = 7 \cdot 2^{32} - 4$. Thus if we let $Z_t = w_{t+\tau_5} \oplus w_{t+\tau_4} \oplus w_{t+\tau_3} \oplus w_{t+\tau_2} \oplus w_{t+\tau_1} \oplus w_t$ then $Z_t = z_{t+\tau_5} \oplus z_{t+\tau_4} \oplus z_{t+\tau_3} \oplus z_{t+\tau_2} \oplus z_{t+\tau_1} \oplus z_t$ so the sequence (Z_t) can be calculated from the keystream. Since we know the distribution of $w_t[i] \oplus w_t[i-1]$ for each value of i, we can calculate the distribution of $Z_t[i] \oplus Z_t[i-1]$. For certain values of i this has bias $2^{-40.5}$. The Neyman-Pearson lemma then tells us that (in the worst case scenario) we need $2^{86.5}$ keystream words to distinguish between keystream from the cipher and a random keystream with probability of error 2^{-32}. The computational complexity of this distinguishing attack is $2^{86.5}$.

De Cannière, Lano, Preneel and Vandewalle [156] enhanced the attack described in [181] by adapting the attack on SOBER-t16 so that it works for SOBER-t32. This attack results in a distinguishing attack on full SOBER-t32 of complexity 2^{153}.

The designers of SOBER-t32 describe a guess-and-determine attack on unstuttered SOBER-t32 with complexity 2^{320} [245]. This attack is extended by de Cannière to improve the complexity to 2^{304} [155]. The unstuttered version of SOBER-t32 may also be analysed by noting that the 8 most significant bits of $\bar{a}$ determine the 8 most significant bits of $f(a)$. Babbage and Lano [24] also describe a guess-and-determine attack based on this observation on unstuttered SOBER-t32 with complexity 2^{244} and also indicate that their approach might be adapted to SOBER-t32 by using timing information to find an unstuttered part of the keystream that their approach requires.

The observation that the 8 most significant bits of $f(a)$ are determined solely by the 8 most significant bits of $\bar{a}$ is also the basis of some comments about the nonlinear filter function. This results in poor diffusion properties for the nonlinear filter, as the only means of diffusion of the 24 least significant bits of the input is by carry propagation. However, long carry chains only occur with low probability. In particular Dichtl and Schafheutle [166] specify a sum of bits from the initial state of the shift register that remains constant with very high probability when the last key bit is inverted. They have also identified other such sums that remain constant for inversion of each of the 11 least significant key bits of the last key word and for each of 8 of the 9 least significant bits of the second to last key word. For more significant positions, carry propagation seems to be sufficiently high to destroy such linearity properties. This property of the key loading could be destroyed by increasing the number of `Diffuse` steps. Similarly, they show that there is a correlation between the initial states derived from different frame keys but the same key. In particular, they give a sum of bits of the initial state that does not change with very high probability if the least significant bit of the last key frame word is inverted. As the frame key is a binary counter this is particularly relevant.

Besides those attacks, SOBER-t32 is vulnerable to timing attacks and power attacks owing to its irregular decimation [245].

3.4.4.3 Comments from the submitters

The submitters have made some comments on the relevance of these results, which also apply to SOBER-t16. These comments are briefly summarised here.

- There is a distinguishing attack on the unstuttered version of SOBER-t32. The submitters in their comments state that irregular stuttering is an essential part of the security of SOBER. However, the NESSIE project believes that the original SOBER submissions clearly state the SOBER stream ciphers are secure without the stuttering.
- The submitters question the rejection of a cipher based solely on distinguishing attacks based on large known plaintext, which is faster than exhaustive key search [247]. The submitters argue that there are many stream ciphers in use for which distinguishing attacks exist (such as a block cipher used in a standard mode that gives a stream cipher) and that the distinguishing attacks presented for the SOBER ciphers do not yield any information about the state, so do not translate into key recovery or prediction attacks.

3.4.4.4 NESSIE response to the comments from the submitters

- Distinguishing attacks on full SOBER-t16 and SOBER-t32 have been found since the comments from the submitters were received.
- Distinguishing attacks do not currently give a method for determining the keystream for SOBER-t16 and SOBER-t32. However, the NESSIE consortium believes that the SOBER distinguishing attacks mean that a recommendation is inadvisable.
- The SOBER design raises other security issues not discussed by the submitters.

3.4.4.5 SOBER-t32 Conclusions

There are distinguishing and guess-and-determine attacks on the unstuttered version of SOBER-t32, and recently distinguishing attacks for full SOBER-t32 have been found. For the guess-and-determine attack there are reasons for believing that it could be extended to SOBER-t32. The nonlinear filter exhibits poor diffusion and has significant biases. Furthermore, SOBER-t32 is vulnerable to timing and power attacks.

3.5 Stream cipher primitives not selected for Phase II

3.5.1 LEVIATHAN

Leviathan is a synchronous additive stream cipher submitted by Cisco and designed so that it can efficiently find arbitrary locations in the keystream. Though Leviathan may work with arbitrary output size and an arbitrary number of key bytes, owing to standardization the key should be either 128 bits or 256 bits and the output size should be 32 bits. It was submitted to NESSIE, but not considered during Phase II of the project.

3.5.1.1 Design

The key scheduling:
Let $K[i]$ be the ith byte of the key of length m bytes, and let $j \in \{0, 1, 2, 3\}$. We define a sequence (k_r) of integers modulo 256 and a sequence (π_r) of byte permutations (i.e. permutations of the set $\{0, \ldots, 255\}$) as follows.

Let π_0 be the identity permutation, and $k_0 = j$. For $r \neq 257$ let

$$k_{r+1} = k_r + K[i \bmod m] + \pi_r(r \bmod 256) \bmod 256$$

If $r = 257$, define $k_{257} = j + k_1 + K[i \bmod m] + \pi_{256}(0 \bmod 256)$. The permutations (π_r) are updated as follows:

$$\pi_{r+1}(i) = \begin{cases} \pi_r(k_{r+1}) & \text{if} \quad i = r \bmod 256 \\ \pi_r(r) & \text{if} \quad i = k_{r+1} \\ \pi_r(i) & \text{else} \end{cases}$$

We then let $\sigma_j = \pi_{512}$. So to define σ_j, we are repeating a loop twice in which we take a permutation and then swap certain pairs of images of elements under the permutation to obtain a new permutation.

Finally we define byte permutations $S0 = \sigma_3$, $S1 = \sigma_2\sigma_3$, $S2 = \sigma_1\sigma_2\sigma_3$ and $S3 = \sigma_0\sigma_1\sigma_2\sigma_3$.

The key stream generation:
Leviathan is defined by a set of binary tree structures of height 16. Each node of each tree is associated with a triple of words (each of four bytes) $z|y|x$. The triple at the root of the jth tree is $1|0|j/2^{16}$. Key-dependent functions a and b map the triple s at a node to a triple at each of its two descendants, so that its lefthand descendant is $a(s)$ while its righthand descendant is $b(s)$, where $a(s)$ in the kth level is chosen, if the kth bit of j is zero. Otherwise $b(s)$ is chosen.

We apply a function c to the triple at each leaf of each tree to give a word and use the words thus obtained as the keystream.

The functions a, b and c are defined as follows:

$$\begin{aligned} f(z|y|x) &= 2z|SRSRy|LSLSx \\ g(z|y|x) &= 2z+1|LSLSy|SRSR(\bar{x}) \\ d(z|y|x) &= z|x+y+z|2x+y+z \\ c(z|y|x) &= x \oplus y \\ a &= f \circ d \\ b &= g \circ d \end{aligned}$$

where L and R denote rotation left and right respectively by one byte, $\bar{x}$ denotes the complement of x and S denotes a key-dependent S-box given below.

The permutation S is defined as follows. Let the word y consist of the bytes y_0, y_1, y_2 and y_3. Then the image of y under S is defined by

$$\begin{aligned} y_0 &\mapsto S0(y_0) \\ y_1 &\mapsto y_1 \oplus S1(y_0) \\ y_2 &\mapsto y_2 \oplus S2(y_0) \\ y_3 &\mapsto y_3 \oplus S3(y_0) \end{aligned}$$

where the byte permutations $S0$, $S1$, $S2$ and $S3$ are defined in the key scheduling.

3.5.1.2 Security analysis

There is a distinguishing attack on LEVIATHAN faster than exhaustive key search [143]. It shows, that the probability of a collision in 64 bit output words is doubled compared to a random function. Hence LEVIATHAN was not selected for further study in phase II.

3.5.2 LILI-128

LILI-128 is a synchronous stream cipher designed by Ed Dawson and submitted to the NESSIE project. It uses a 128-bit key and an internal memory of 128 bits. LILI-128 was not considered during Phase II of the project.

3.5.2.1 Design

LILI-128 consists of two components, one used for clock control and the other for data generation. Each component consists of an LFSR ($LFSR_c$ and $LFSR_d$) and a function f (f_c and f_d) tapping the LFSRs. LILI-128 can be viewed as a clock-controlled nonlinear filter generator.

The key scheduling:
During key scheduling the 128 key bits are loaded directly into the LFSRs. The first 39 key bits are loaded into $LFSR_c$, and the last 89 key bits into $LFSR_d$. Neither $LFSR_c$ nor $LFSR_d$ may be zero.

The clock control component:
The clock control component consists of the regularly clocked $LFSR_c$ of length 39 and the function f_c. The feedback poynomial of $LFSR_c$ is

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1$$

and it produces a maximum length sequence.

$LFSR_c$ is clocked once, then the function f_c takes the contents of the stages 12 and 20 as input and produces an output integer by

$$c = f_c(x_{12}, x_{20}) = 2x_{12} + x_{20} + 1.$$

The data generation component:
$LFSR_d$ of length 89 produces a maximum length sequence. Its feedback polynomial is

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

After the clock control component has produced the integer c, $LFSR_d$ is clocked c times. The nonlinear function f_d defined by a truth table takes the content of 10 stages of $LFSR_d$ as input and calculates an output bit. This output bit is taken as the new keystream bit.

3.5.2.2 Security analysis

The key of LILI-128 can be recovered faster than exhaustive key by several kinds of time-memory tradoff attacks [22, 459] or a fast correlation attack [270]. The last attack has a complexity around 2^{71} bit operations assuming a received sequence of length around 2^{30} bits and a precomputation phase of complexity 2^{79} table lookups. Hence LILI-128 was not selected for further study in phase II.

3.5.3 RC4

RC4 is the de facto standard for stream ciphers (see [451] for a detailed description). The second output byte of RC4 can be easily distinguished from a random one [349]. Although this can be easily overcome, the keystream still can be distinguished from a random output [197] and the key schedule has severe weaknesses [196]. There is also no form of rekeying defined for RC4. Therefore RC4 was not considered by NESSIE.

Changes from version 1.0 to version 2.0 of the document

—	Typos have been corrected.
§3.2.1	Description of time-memory-tradoff for exhaustive key search attack added.
§3.2.1	Reference changed for description of efficient time-memory tradeoffs.
§3.2.1	Description of test for nonlinear complexity added.
§3.3.4	References added for RC4, SCREAM and SEAL.
§3.3.5	Desription of different types of OFB mode and its relationship to distinguishing attacks added.
§3.3.5	ISO number changed.
§3.4.1.2	Explanation of "generic attack" added.
§3.4.2.2	Notation for distinguishing attack changed.
§3.4.4.1	Recurrence relation changed.
§3.4.4.4	Response added.
§3.5.1	Description for LEVIATHAN added.
§3.5.2	Description for LILI-128 added.

4. Hash functions

4.1 Introduction

Hash functions are used as a building block in various cryptographic applications. The most important uses are in the protection of information authentication and as a tool for digital signature schemes (see Chapter 7). A hash function is a function that maps an input of arbitrary length into a fixed number of output bits, the *hash value*. In order to be useful for cryptographic applications, a hash function needs to satisfy some additional requirements. Hash functions can be further divided into one-way hash functions and collision-resistant hash functions. We informally give the conditions we require of these different types:

- **A one-way hash function** should be *preimage* and *second preimage* resistant, that is it should be 'hard' to find a message with a given hash (preimage) or that hashes to the same value as a given message (second preimage).
- **A collision-resistant hash function** is a one-way hash function for which it is 'hard' to find two distinct messages that hash to the same value.

In some applications additional properties may be required for a hash function, for example pseudo-randomness of the generated output. Note that, in contrast to other cryptographic primitives, the computation of a hash function does *not* depend on any secret information.

Before presenting a detailed analysis of the hash functions studied by the NESSIE project, we first discuss the security requirements and give an overview of common designs and current standards. For a more comprehensive overview of cryptographic primitives for information authentication (including hash functions) we refer to the treatment by Preneel in [440].

4.2 Security requirements

In this section we give practical and formal definitions for one-way and collision-resistant hash functions and describe the general model of an iterated hash function. We then discuss different types of attacks on hash functions and describe the assessment process followed by the project.

[0] Coordinator for this chapter: KUL — Bart Van Rompay

4.2.1 Security model

The following informal definitions for one-way and collision-resistant hash functions were given by Preneel in [440].

A **one-way hash function** is a function h satisfying the following conditions:

1. The argument X can be of arbitrary length and the result $h(X)$ has a fixed length of n bits.
2. The hash function must be one-way in the sense that given a Y in the image of h, it is 'hard' to find a message X such that $h(X) = Y$ (preimage-resistant) and given X and $h(X)$ it is 'hard' to find a message $X' \neq X$ such that $h(X') = h(X)$ (second preimage-resistant).

A **collision-resistant hash function** is a function h satisfying the following conditions:

1. The argument X can be of arbitrary length and the result $h(X)$ has a fixed length of n bits.
2. The hash function must be one-way, i.e., preimage-resistant and second preimage-resistant.
3. The hash function must be collision-resistant: this means that it is 'hard' to find two distinct messages that hash to the same result (i.e., find X and X' $(X' \neq X)$ such that $h(X) = h(X')$).

Note that if an attacker can find a second preimage, he can also find a collision. Therefore the second preimage condition in this definition is redundant. However, preimage-resistance is not always implied by collision resistance.

Most hash functions are iterated constructions, in the sense that they are based on a compression function with fixed size inputs; they process every message block in a similar way. The input X is padded by an unambiguous padding rule to a multiple of the block size. Typically this also includes adding the total length in bits of the input. The padded input is then divided into t blocks denoted X_1 through X_t. The hash function involves a *compression function* f and a *chaining variable* H_i between stage $i-1$ and stage i:

$$\begin{aligned} H_0 &= IV \ , \\ H_i &= f(H_{i-1}, X_i) \ , \ 1 \leq i \leq t \ , \\ h(X) &= g(H_t) \ . \end{aligned}$$

Here IV denotes the Initial Value (a constant which should be defined as part of the description of the hash function) and g denotes the (optional) *output transformation*. A collision-resistant compression function can be extended to a collision-resistant hash function taking arbitrary length inputs. For a detailed discussion on the relation between a compression function and a hash function that is built from it, we refer to [440].

4.2.1.1 Formal definitions

Before discussing security aspects we first give more precise definitions of a hash function and its cryptographic properties. The following formal definitions for a hash function, a compression function and an iterated hash construction are similar to those given by Black *et al.* in [82].

Definition 4.1. *A* **hash function** *is a function $h : \mathcal{D} \rightarrow \mathcal{R}$ where the domain $\mathcal{D} = \{0,1\}^*$ and the range $\mathcal{R} = \{0,1\}^n$ for some $n \geq 1$.*

Definition 4.2. *A* **compression function** *is a function $f : \mathcal{D} \rightarrow \mathcal{R}$ where $\mathcal{D} = \{0,1\}^a \times \{0,1\}^b$ and $\mathcal{R} = \{0,1\}^n$ for some $a, b, n \geq 1$ with $a + b \geq n$.*

Definition 4.3. *The* **iterated hash** *of the compression function $f : (\{0,1\}^n \times \{0,1\}^b) \rightarrow \{0,1\}^n$ is the hash function $h : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ defined by $h(X_1 \ldots X_t) = H_t$ where $H_i = f(H_{i-1}, X_i)$ for $1 \leq i \leq t$ (set $H_0 = IV$).*

The following definitions for (second) preimage-resistance and for collision-resistance were given by Brown in [108]. These definitions are somewhat simplified because we consider a fixed hash function rather than a family of hash functions. Therefore our assumptions, particularly the assumption about collision-resistance, are stronger than the usual assumptions for families of hash functions.

Definition 4.4 (Preimage-resistance). *A hash function $h : \{0,1\}^* \rightarrow \mathcal{R}$ is preimage-resistant of strength (t, ϵ) if there exists no probabilistic algorithm I_h that accepts input $Y \in_R \mathcal{R}$ and outputs a value $X \in \{0,1\}^*$ in running time at most t, where $h(X) = Y$ with probability at least ϵ, assessed over the random choices of both Y and I_h.*

Definition 4.5 (Second preimage-resistance). *Let $\mathcal{S}$ be a finite subset of $\{0,1\}^*$. A hash function $h : \{0,1\}^* \rightarrow \mathcal{R}$ is second preimage-resistant of strength $(t, \epsilon, \mathcal{S})$ if there exists no probabilistic algorithm S_h that accepts input $X \in_R \mathcal{S}$ and outputs a value $X' \in \{0,1\}^*$ in running time at most t, where $X' \neq X$ and $h(X') = h(X)$ with probability at least ϵ, assessed over the random choices of both X and S_h.*

Note: One can define a stronger notion of (second) preimage resistance, where the value $Y \in \mathcal{R}$ (or the value $X \in \mathcal{S}$) is a fixed point rather than a random point, and where one maximises over all such points.

Definition 4.6 (Collision-resistance). *A hash function $h : \{0,1\}^* \rightarrow \mathcal{R}$ is collision-resistant of strength (t, ϵ) if no probabilistic algorithm C_h is known that outputs values $X, X' \in \{0,1\}^*$ in running time at most t, where $X' \neq X$ and $h(X) = h(X')$ with probability at least ϵ, assessed over the random choices of C_h.*

Note: Since $\{0,1\}^*$ is infinite and $\mathcal{R}$ is finite, collisions for h do exist. And since C_h has no input, there exists a very efficient algorithm, namely one that immediately outputs (X, X') for some fixed collision. Nevertheless, for a good hash function with $\mathcal{R} = \{0,1\}^n$, the best collision-finding algorithm *known* with high success probability should have a running time of about $2^{n/2}$ (see Sect. 4.2.2).

4.2.2 Classification of attacks

For one-way and collision-resistant hash functions the computation of the function does not involve any secret information. Hence, it is not meaningful to distinguish attacks depending on the information available to an attacker. The goal of the attacker is to find a *preimage* or *second preimage* for the hash function, or to generate a *collision*. Collision-resistance is not required in all applications (the most important case where it is required is in conjunction with digital signatures), which is the reason for the separate category of one-way hash functions. Also it can be noted that some attacks find a preimage or collision only for random messages, while others leave the attacker enough freedom to choose (part of) the messages.

Note that the security of a hash function can be examined through analysis of its compression function. A collision-resistant compression function can be extended to a collision-resistant hash function taking arbitrary length inputs. On the other hand, attacks on the compression function do not necessarily mean attacks on the hash function: an attack that finds preimages or collisions for f with chosen H_{i-1} leads to an attack on h, but an attack that finds preimages or collisions for f with a random H_{i-1} does not yield an attack on h (unless when the IV can be changed). Another example is an attack that finds collisions for the compression function where the initial chaining value is not the same for both messages ($H_{i-1} \neq H'_{i-1}$). Attacks on the compression function which cannot be extended are seen as certificational weaknesses of the hash function involved.

We now briefly describe the best known attacks on hash functions as in [440].

Random (Second) Preimage Attack

This attack can be applied to any hash function and depends only on the size n in bits of the hash result. An attacker simply selects a random message and hopes that a given hash result occurs. If the hash function has the required "random" behaviour, his probability of success equals $1/2^n$. This attack can be carried out off-line and in parallel. It is expected that a (second) preimage can be found in approximately 2^n operations.

Birthday Attack

This attack can be applied to any hash function and depends only on the size n in bits of the hash result. The birthday paradox states that for a group of 23 people, the probability that at least two people have a common birthday exceeds 1/2. Intuitively one expects that the probability is much lower. However, the probability is determined by the number of pairs of people in such a group and this number equals $23 \cdot 22/2 = 253$. This can be exploited to find collisions for a hash function in the following way: an adversary generates r_1 variations on a bogus message and r_2 variations on a genuine message. The expected number of collisions equals $r_1 \cdot r_2/2^n$. The probability of finding a bogus message and a genuine message that hash to the same result is given by $1 - \exp(-r_1 \cdot r_2/2^n)$, which is about 63% when $r = r_1 = r_2 = 2^{\frac{n}{2}}$. Finding the collision does not

require r^2 operations: after sorting the data, which requires $O(r \log r)$ operations, comparison is easy. One can reduce the memory requirements for collision search by translating the problem to the detection of a cycle in an iterated mapping.

Meet-in-the-Middle Attack

This attack — as well as the attacks described below — depends on some properties of the compression function. It is a variation on the birthday attack, but instead of comparing the hash result, one compares intermediate chaining variables. If the attack can be applied to a hash function, it enables an adversary to construct a (second) preimage, which is not possible for a simple birthday attack. The opponent generates r_1 variations on the first part of a bogus message and r_2 variations on the last part. Starting from the initial value and going backwards from the hash result, the probability for a matching intermediate variable is again given by $1 - \exp(-r_1 \cdot r_2 / 2^n)$. The cycle finding algorithm can be used to perform a meet-in-the-middle attack with negligible storage.

Correcting Block Attack

This attack consists of substituting all blocks of the message except for one or more blocks. For a (second) preimage attack, one chooses an arbitrary message X and finds one or more correcting blocks Y such that $h(X\|Y)$ takes a certain value. For a collision attack, one starts with two arbitrary messages X and X' and appends one or more correcting blocks denoted with Y and Y', such that the extended messages $X\|Y$ and $X'\|Y'$ have the same hash result.

Fixed Point Attack

The idea of this attack is to look for a H_{i-1} and X_i such that $f(H_{i-1}, X_i) = H_{i-1}$. If the chaining variable is equal to H_{i-1}, it is possible to insert an arbitrary number of blocks equal to X_i without modifying the hash result. Producing collisions or a second preimage with this attack is only possible if the chaining variable can be made equal to H_{i-1}: this is the case if IV can be chosen equal to a specific value, or if a large number of fixed points can be constructed (i.e., if one can find an X_i for a significant fraction of H_{i-1}'s). Of course this attack can be extended to fixed points that occur after more than one iteration.

Differential Attacks

Differential cryptanalysis [70] is a powerful tool for the analysis of not only block ciphers (see Chapter 2) but also of hash functions. This attack method examines input differences to a compression function and the corresponding output differences. A collision is obtained if the output difference is zero.

Analytical Weaknesses

A large number of attacks have been based on blocking the diffusion of the data input to the hash function: this means that changes have no effect or can be cancelled out easily in a next stage. Among the most remarkable attacks in this class are the attacks developed by Dobbertin on the MDx-family [170, 171]. They combine conventional optimisation techniques (simulated annealing, genetic algorithms) with conventional cryptanalytic techniques. Another property of these attacks is that differences are only controlled in certain points of the algorithms; this in contrast to differential cryptanalysis, where typically all intermediate differences are controlled to a certain extent.

Side-Channel Attacks

Side-channel attacks are a major threat for all implementations of cryptographic algorithms. Hash functions are only vulnerable to this type of attack when part of the input to the function is secret, for example when a hash function is used in a construction for a message authentication code (MAC) as described in Chapter 5, or when it is used in a key derivation function (KDF) as described in Chapter 6. In such a case side-channel attacks on hash functions are similar as for other symmetric primitives. For a detailed discussion on side-channel attacks and countermeasures that can be applied to protect an implementation we refer to Annex A.

4.2.3 Assessment process

The hash function submissions were assessed with reference to the above generic common hash function attacks and by specific attacks when appropriate. Statistical analysis was carried out for various input lengths. Hash functions were submitted to the dependence test and linear factors test described in Sect. 2.2.4. If the NESSIE submissions were based on compression functions, these were also submitted to the two tests. Furthermore the stream cipher tests described in Sect. 2.2.4 were also applied on the hashes both in output feedback and counter mode. None of the hash functions tested exhibited any anomalous behaviour.

4.3 Overview of the common designs

In this section we give an overview of those hash functions which are most commonly used in practice today. Most known hash functions are iterated constructions that are based on a compression function with fixed size input. Sometimes, this compression function is based on a block cipher or on modular arithmetic; in other cases it is built from scratch specifically for the purpose of hashing. We conclude the section with a summary of the procedures which have been undertaken by several organisations for the standardisation of hash functions.

4.3.1 Hash functions based on block ciphers

In systems where a block cipher implementation is already available, a hash function can be obtained at little extra cost (in design, evaluation and implementation) by constructing it from the underlying block cipher. A disadvantage is that these hash functions are usually slower than the dedicated proposals (especially when the block cipher used has a slow key schedule). Furthermore, the use of a block cipher in a different context may reveal weaknesses which are not relevant to encryption (it is still an open problem which requirements for a block cipher are sufficient in order to produce a secure hash function).

Most block cipher based hash functions belong to one of two classes: those producing a hash value of single block length and those producing a hash value of double block length. In case the AES (Advanced Encryption Standard, see Chapter 2) is used as underlying block cipher, the resulting output lengths of these hash function classes are 128 and 256 bits respectively. Another relevant characteristic is the rate of the hash function, which is equal to the number of blocks that are hashed for each computation of the encryption function.

There are several *single block length hash functions* of rate 1 with provable black-box security (assuming the underlying block cipher satisfies the required randomness properties). Two dual constructions are those attributed to Matyas-Meyer-Oseas and Davies-Meyer. In each step of the iteration the previous value of the chaining variable (H_{i-1}) and the message block to be processed (X_i) serve as key and plaintext of the encryption function (or vice versa), and there is an extra feed-forward with the purpose of making the compression function uninvertible. For a block cipher $E_K(X)$ (with key K and plaintext X) the compression function of the two constructions is as follows:

- Matyas-Meyer-Oseas: $H_i = E_{g(H_{i-1})}(X_i) \oplus X_i$,
- Davies-Meyer: $H_i = E_{X_i}(H_{i-1}) \oplus H_{i-1}$.

Here g is a function that maps H_{i-1} to a key suitable for E. The Miyaguchi-Preneel hashing scheme is a variation on Matyas-Meyer-Oseas, where the only difference is that the output H_{i-1} from the previous stage is also XORed to that of the current stage.

- Miyaguchi-Preneel: $H_i = E_{g(H_{i-1})}(X_i) \oplus X_i \oplus H_{i-1}$.

The most common *double block length hash functions* are MDC-2 and MDC-4 with a rate of respectively 1/2 and 1/4 (all proposed schemes with rate 1 have been broken). MDC-2 requires two parallel block encryptions in each iteration step, and the compression function of MDC-4 has two sequential executions of the MDC-2 compression function. The level of security of these two schemes is less than suggested by the output size.

The Matyas-Meyer-Oseas scheme and MDC-2 are included in ISO/IEC 10118-2 [258], a standard for hash functions using an (unspecified) block cipher. The NESSIE submission Whirlpool (see Sect. 4.4.1) is based on the Miyaguchi-Preneel hashing mode of an internal block cipher.

4.3.2 Hash functions based on modular arithmetic

A hash function can also use modular arithmetic as the basis of its compression function, allowing the re-use of existing implementations of modular arithmetic (such as in public-key systems). The biggest disadvantage is the low speed of these constructions (especially compared to dedicated hash functions). Many of the proposals which use modular arithmetic have been broken. The experience with these attacks has led to the design of the MASH-1 and MASH-2 hash functions.

The compression function of MASH-1 is based on a modular squaring operation, where the modulus M is chosen as a composite of sufficient bitlength m (making it infeasible to factor M). The message block input X_i is first expanded with redundancy bits to a block Y_i of double length. The block Y_i is then added bitwise to the previous value of the chaining variable H_{i-1} (both Y_i and H_{i-1} have bitlength n, chosen as the largest multiple of 16 less than m). After the squaring operation a feed-forward is applied with H_{i-1}. This results in the following equation for the compression function:

$$H_i = ((((H_{i-1} \oplus Y_i) \vee A)^2 \mod M) \dashv n) \oplus H_{i-1} \ ,$$

where A is a constant forcing the four most significant bits to 1, prior to squaring; and $\dashv n$ denotes truncation of the result of the squaring operation to the n least significant bits.

The security of this construction depends in part on the difficulty of extracting modular roots (for a composite of unknown factorisation). The redundancy bits are vital as well, resulting in the following security level: matching a given hash value requires $2^{n/2}$ operations; finding a collision requires $n \times 2^{n/4}$ operations.

MASH-2 is a variant of MASH-1, the only difference is that it uses a modular exponentiation with exponent $e = 2^8 + 1$ (instead of modular squaring with $e = 2$). Both MASH-1 and MASH-2 are included in ISO/IEC 10118-4 [260], a standard for hash functions using modular arithmetic. This standard also defines an additional output transformation that must be used to reduce the length of the hash value to $n/2$ bits or less.

4.3.3 Dedicated hash functions

Dedicated hash functions are functions specifically designed for the purpose of hashing, with optimised performance in mind. The hash functions of this class which have received the most attention are those based on the design of the MD4 algorithm. While MD4, which dates from 1990, has been broken with respect to collision-resistance, the algorithms derived from it have benefited from the experience gained with the cryptanalysis of this type of hash functions.

The MD4-based algorithms generally divide the message block X_i and the chaining variable H_{i-1} in words of 32 bits (or 64 bits for some of the new proposals). The compression function updates the chaining variable to a new value H_i, making use of the current message block. The computation is divided into a number of sequential steps, where each step updates the value of one register —

containing one word of the chaining variable — applying one message word (and depending on the other registers). The compression function can also be divided into a number of rounds, where in general each round uses all words from the message block exactly once.

As an example we describe the step operation of MD4, which is of the following form:

$$A = (A + f(B, C, D) + X[j] + y)^{<<s[j]} \ .$$

Here (A, B, C, D) are the registers containing the four 32-bit words of the chaining variable. The notation + means addition modulo 2^{32} and $(\cdot)^{<<s}$ means bitwise rotation (circular shift) over s positions to the left. f is a non-linear function working at bit level, $X[j]$ is the message word applied in step j, y an additive constant and $s[j]$ a rotation constant. The function f and constant y differ in different rounds (MD4 has three rounds using the multiplexer, majority and exclusive-or functions), as does the ordering in which the message words are applied throughout the steps of one round.

The step operations are reversible (one can calculate backwards easily), but at the end of the compression function there is a feed-forward operation which adds the initial value of the registers to their final value in order to compute the chaining variable H_i. This makes the compression function uninvertible. The chaining variable derived by the last application of the compression function (which works on a message block with padding and length information) serves as the result of the hash function.

Several algorithms have been derived from MD4, applying different ideas in order to increase the security against preimage and collision attacks. These ideas include the use of more rounds, slightly different step operations and longer chaining variables (which also means a longer hash result). In particular, the SHA family[1] uses a procedure for expansion of the message block in order to calculate a different word for every step (instead of just reordering the message words between different rounds). The RIPEMD family uses two parallel lines of computation consisting of modified versions of MD4. In Table 4.1 we summarise the main differences between the different algorithms.

Table 4.1. Summary of selected MD4-based hash functions. All sizes are given in bits. Note that for SHA-2 there is no clear distinction in rounds.

Algorithm	output size	block size	word size	rounds × steps per round
MD4	128	512	32	3×16
MD5	128	512	32	4×16
RIPEMD-128	128	512	32	4×16 twice (in parallel)
RIPEMD-160	160	512	32	5×16 twice (in parallel)
SHA-1	160	512	32	4×20
SHA-2/256	256	512	32	1×64
SHA-2/384	384	1024	64	1×80
SHA-2/512	512	1024	64	1×80

[1] A detailed description of these hash functions is given in Sect. 4.4.2 and Sect. 4.4.3.

4.3.4 Current standards

Several organisations have taken initiatives for the standardisation of hash functions. The SHA-1 hash function is a U.S. government standard, developed by NIST as Federal Information Processing Standard (FIPS) 180-1, and it can be used in conjunction with the DSA standard for digital signatures. Recently NIST has updated this standard to FIPS 180-2 [393] which includes, besides SHA-1, also three new hash functions — SHA-2/256, SHA-2/384 and SHA-2/512 — with longer output lengths (in order to match the security level of the new block cipher standard AES). ANSI has adopted hash functions in its public-key based banking standards: standard X9.30 [14] specifies SHA-1 to be used in conjunction with DSA; standard X9.31 [15] specifies MDC-2 to be used in conjunction with an RSA-based signature scheme.

ISO/IEC has developed standard 10118 for hash functions, with separate parts for different classes of hash functions. Part 10118-2 [258] details hash functions based on block ciphers, more specifically the Matyas-Meyer-Oseas construction, a block cipher independent MDC-2 and two more functions producing a hash value of double and triple block length respectively. Part 10118-3 [259] specifies three dedicated algorithms: RIPEMD-128, RIPEMD-160 and SHA-1. This part of the standard is currently being revised, with an ongoing assessment of new cryptographic primitives to be adopted as ISO standards. Besides the original three algorithms, the following hash functions are being considered: SHA-2/256, SHA-2/384, SHA-2/512 and Whirlpool. Part 10118-4 [260] describes the MASH-1 and MASH-2 hash functions which use modular arithmetic.

4.4 Hash functions considered during Phase II

The Whirlpool algorithm was submitted to NESSIE and selected for study during phase II of the NESSIE project. Furthermore, because they are standards of NIST [393], and under discussion in ISO [259], the algorithms SHA-1 and SHA-2 were selected for study during NESSIE phase II.

4.4.1 Whirlpool

Whirlpool is a hash function designed by Paulo Barreto and Vincent Rijmen [31] and submitted to the NESSIE project as a (conjectured) collision-resistant hash function. The algorithm is byte-oriented and operates on blocks of 512 bits, producing a message digest (hash value) of 512 bits.

4.4.1.1 The design

The design of Whirlpool consists of the iterative application of a compression function which is based on an underlying dedicated 512-bit block cipher that uses a 512-bit key and runs in 10 rounds. In the following we first describe the individual components that build up Whirlpool, and then specify the complete hash function in terms of these components.

Input and output.
The hash state is internally viewed as a matrix in $\mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$. Therefore, 512-bit data blocks (externally represented as byte arrays by sequentially grouping the bits in 8-bit chunks) must be mapped to and from this matrix format. This is done by function $\mu : \mathrm{GF}(2^8)^{64} \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ and its inverse:

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{8i+j},\ 0 \leqslant i, j \leqslant 7.$$

The nonlinear layer γ.
Function $\gamma : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ consists of the parallel application of a nonlinear substitution box $S : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$, $x \mapsto S[x]$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \leqslant i, j \leqslant 7.$$

The substitution box proposed in the tweaked version of Whirlpool is different from the one proposed in the original version. It is built in a simple way from smaller 4-bit substitution boxes. For its description we refer to the new algorithm specification [31].

The cyclical permutation π.
The permutation $\pi : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ cyclically shifts each column of its argument independently, so that column j is shifted downwards by j positions:

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod 8, j},\ 0 \leqslant i, j \leqslant 7.$$

The purpose of π is to disperse the bytes of each row among all rows.

The linear diffusion layer θ.
The diffusion layer $\theta : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ is a linear mapping based on the $[16, 8, 9]$ MDS code with generator matrix $G_C = [I\, C]$ where

$$C = \begin{bmatrix}
01_x & 01_x & 03_x & 01_x & 05_x & 08_x & 09_x & 05_x \\
05_x & 01_x & 01_x & 03_x & 01_x & 05_x & 08_x & 09_x \\
09_x & 05_x & 01_x & 01_x & 03_x & 01_x & 05_x & 08_x \\
08_x & 09_x & 05_x & 01_x & 01_x & 03_x & 01_x & 05_x \\
05_x & 08_x & 09_x & 05_x & 01_x & 01_x & 03_x & 01_x \\
01_x & 05_x & 08_x & 09_x & 05_x & 01_x & 01_x & 03_x \\
03_x & 01_x & 05_x & 08_x & 09_x & 05_x & 01_x & 01_x \\
01_x & 03_x & 01_x & 05_x & 08_x & 09_x & 05_x & 01_x
\end{bmatrix},$$

so that $\theta(a) = b \Leftrightarrow b = a \cdot C$. The effect of θ is to mix the bytes in each state row.

The key addition $\sigma[k]$.
The affine key addition $\sigma[k] : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ consists of the bitwise addition (exor) of a key matrix $k \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$:

$$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij},\ 0 \leqslant i, j \leqslant 7.$$

This mapping is also used to introduce round constants in the key schedule.

The round constants c^r.
The round constant for the r-th round, $r > 0$, is a matrix $c^r \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, defined as:

$$\begin{array}{rcll} c^r_{0j} & \equiv & S[8(r-1)+j], & 0 \leqslant j \leqslant 7, \\ c^r_{ij} & \equiv & 0, & 1 \leqslant i \leqslant 7, 0 \leqslant j \leqslant 7. \end{array}$$

The round function $\rho[k]$.
The r-th round function is the composite mapping $\rho[k] : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, parameterised by the key matrix $k \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ and given by:

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma.$$

The key schedule.
The key schedule expands the 512-bit cipher key $K \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ onto a sequence of round keys $K^0, \ldots, K^{10}$:

$$\begin{array}{rcl} K^0 & = & K, \\ K^r & = & \rho[c^r](K^{r-1}),\ 1 \leqslant r \leqslant 10, \end{array}$$

The internal block cipher W.
The dedicated 512-bit block cipher $W_K : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, parameterised by the 512-bit cipher key K, is defined as

$$W_K = \left(\bigcirc_{1}^{r=10} \rho[K^r] \right) \circ \sigma[K^0],$$

where the round keys $K^0, \ldots, K^{10}$ are derived from K by the key schedule.

Padding and MD-strengthening.
Before being subjected to the hashing operation, a message M of bit length $L < 2^{256}$ is padded with a 1-bit, then with as few 0-bits as necessary to obtain a bit string whose length is an odd multiple of 256, and finally with the 256-bit right-justified binary representation of L, resulting in the padded message m, partitioned in t 512-bit blocks $m_1, \ldots, m_t$.

The compression function.
WHIRLPOOL iterates the Miyaguchi-Preneel hashing scheme over the t padded message blocks m_i, $1 \leqslant i \leqslant t$, using the dedicated 512-bit block cipher W:

$$\begin{array}{rcl} \eta_i & = & \mu(m_i),\ 1 \leqslant i \leqslant t \\ H_0 & = & \mu(IV), \\ H_i & = & W_{H_{i-1}}(\eta_i) \oplus \eta_i \oplus H_{i-1},\ 1 \leqslant i \leqslant t, \end{array}$$

where IV (the *initialisation vector*) is a string of 512 0-bits.

Message digest computation.
The WHIRLPOOL message digest for message M is defined as the output H_t of the compression function, mapped back to a bit string:

$$\text{WHIRLPOOL}(M) \equiv \mu^{-1}(H_t).$$

4.4.1.2 Security analysis

The Miyaguchi-Preneel scheme is one of the few still unbroken methods to construct a hash function from an underlying block cipher. In particular, it is "provably secure" if certain ideal properties hold for the underlying block cipher (in [82] Black *et al.* prove tight upper and lower bounds for the collision-resistance and preimage-resistance of this construction, based on a black-box model of the encryption algorithm).

The block cipher W that forms the basis of Whirlpool is very similar to the AES algorithm Rijndael. The main difference between W and Rijndael is that Rijndael supports blocklengths of 128, 192 and 256 bits, while W only works on 512-bit blocks. Rijndael has received quite a bit of analysis during and after the AES process, and given the similarities between Rijndael and W, some of it may carry over to Whirlpool. The designers state three criteria that define the level of security. Assume we take as hash result any n-bit substring of the output of Whirlpool, then the criteria are:

- The workload of generating a collision is expected to be of the order $2^{n/2}$ (collision-resistant).
- Given an n-bit value, the workload of finding a message that hashes to that value is of the order 2^n (preimage-resistant).
- Given a message (and its hash result), the workload of finding a different message that has the same hash value is of the order 2^n (second preimage-resistant).

No attacks have been reported on Whirlpool that falsify these statements. In the following we discuss some observations that have been made and an attack on a reduced round version of Whirlpool.

Choice of the substitution box.
In its nonlinear layer Whirlpool uses a substitution box to map 8-bit inputs into 8-bit outputs. The originally submitted form of Whirlpool used a pseudo-randomly generated substitution box, chosen to satisfy certain conditions with respect to differential and linear analysis, and with respect to the non-linear order. However, a flaw that went unnoticed caused the value of the linear parameter to be incorrectly reported. Therefore, in the tweaked version of Whirlpool an alternative substitution box was described satisfying the design conditions. This new substitution box is built in a simple way from smaller 4-bit substitution boxes.[2]

[2] The new substitution box also leads to more efficient hardware implementations.

Non-random properties of reduced-round Whirlpool.

In [302] it is shown that a variant of the hash function Whirlpool where the number of rounds in the compression function is reduced from ten to six (or less), exhibits some non-random properties. The observations have not shown to be a weakness for Whirlpool. In the following we summarise the analysis of [302].

Consider a collection of 256 texts, which have different values in one byte and equal values in each of the remaining 63 bytes. Then it follows that after two rounds of encryption the texts take all 256 values in each of the 64 bytes, and that after three rounds of encryption the sum of the 256 bytes in each position is zero. Such a structure has been called an integral (see Sect. 2.2.3.15). Also, note that there are 63 other similar structures, since the position of the non-constant byte in the plaintexts can be in any of the 64 positions.

The three-round integral described above can be extended to four rounds by considering a structure with 2^{64} texts. The main observation is that after one round one has all 2^{64} values in the top row and that the remaining three rounds can be seen as a collection of 2^{56} variants of the 3-round integral. Since the texts in each integral sum to zero in any byte after the fourth round, so does the sum of all 2^{64} texts.

In much the same manner, one can define a three-round backwards integral through three rounds of the inverse cipher of W. In versions of W reduced to six rounds one can then combine the first three rounds of the four-round forwards integral and the three-round backwards integral to cover all rounds of the cipher with probability one. We do this by starting our computation after the third round of W. By a first glance it appears that the two three-round integrals cannot be combined.

However, choose a structure of 2^{120} texts such that the fifteen bytes consisting of the eight bytes of the top row and byte j in row $8 - j$ for $j = 1 \ldots 7$ take all possible values. One can view these texts as a collection of 2^{56} 3-round forwards integrals, but one can also view this as a collection of 2^{56} 3-round backwards integrals. Therefore, both when one computes forward three rounds and backward three rounds, the resulting texts will take the values in each byte equally many times.

Thus, with time complexity 2^{120} one can find a collection of 2^{120} inputs to W reduced to six rounds, such that each byte in both the inputs and outputs takes all values equally many times. It is claimed that this distinguishes W reduced to six rounds from a randomly chosen 512-bit permutation. We conjecture that to find a structure of 2^{120} texts with properties similar to the ones outlined for W reduced to six rounds will require the generation of at least 2^{128} outputs and a considerable amount of memory.

In versions of W reduced to seven rounds one can combine the full four-round forwards integral and the three-round backwards integral to cover seven rounds with probability one. However it is unclear whether such an integral can be used to effectively distinguish W reduced to seven rounds from a randomly chosen 512-bit permutation. In recent work David Wagner shows techniques for solving a generalised birthday problem. Although these techniques do not seem to apply directly to our problem, it is possible that they can be adapted hereto.

Quadratic relations in Whirlpool.

In [303] it is examined whether there exist quadratic relations with certainty over the input and output bits of the substitution boxes of Whirlpool. It has been shown by Courtois and Pieprzyk [138] that for the substitution boxes of Rijndael and Serpent there exist quadratic equations in the input and output bits which hold with probability one. Such equations always exist for n-bit to n-bit S-boxes where $n \leq 6$, but not in general for $n > 6$.

Therefore we have investigated whether such quadratic relations exist for the S-box of Whirlpool. This S-box is a permutation of eight bits. There are 137 possible terms of degree at most two in a multivariate expression of the eight input and output bits. It is a simple matter to check whether such equations exist simply by computing the kernel of a 256 times 137 binary matrix. We implemented this in Maple. It was found that there are no quadratic relations for the full S-box of Whirlpool. However, after the tweak, the S-box of Whirlpool is built in a simple way from 4-bit S-boxes. This makes it easy to write a small system of multivariate quadratic equations for Whirlpool. Security of Whirlpool against algebraic attacks is a matter of further research.

4.4.2 SHA-1

SHA-1 [393] has been a NIST hash function standard (FIPS 180-1, recently updated to FIPS 180-2) for a long time, and is also included in the ISO/IEC standard 10118-3. Although the output length of 160 bits is too short for a collision-resistant hash function as requested by the NESSIE call for primitives, many current applications (such as digital signature schemes) use a 160-bit hash function. Therefore SHA-1 is studied as a *legacy* hash function. The algorithm operates on 512-bit message blocks divided in words of 32 bits, and produces a message digest (hash value) of 160 bits.

4.4.2.1 The design

SHA-1 is a dedicated hash function, clearly influenced by the design of MD4 but a strengthened version of this algorithm. It is defined as the iteration of a compression function which we specify below. The computation starts with the initial value

$$IV = 67452301_x \quad \texttt{efcdab89}_x \quad \texttt{98badcfe}_x \quad 10325476_x \quad \texttt{c3d2e1f0}_x \ .$$

Each application of the compression function uses five words as initial values and 16 words of the message as input and it gives five words output, which are then used as initial values for the next application. The final output is the hash value. This works because there is a padding rule which adds bits to the message (including a representation of the message length) so that its length becomes a multiple of 512 bits.

Compression function of SHA-1.

Let the message block of 512 bits be denoted $M = [W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}]$, where W_i are 32-bit words. SHA-1 uses an expansion procedure which is defined as

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})^{<<1} \ , \ 16 \le i \le 79 \ .$$

Define the following constants

$$\begin{aligned} K_i &= 5a827999_x \ , \ 0 \le i \le 19 \ , \\ K_i &= 6ed9eba1_x \ , \ 20 \le i \le 39 \ , \\ K_i &= 8f1bbcdc_x \ , \ 40 \le i \le 59 \ , \\ K_i &= ca62c1d6_x \ , \ 60 \le i \le 79 \ , \end{aligned}$$

and the boolean functions

$$\begin{aligned} f_i(X,Y,Z) &= f_{if}(X,Y,Z) = (X \,\&\, Y) \,|\, (\bar{X} \,\&\, Z) \ , \ 0 \le i \le 19 \ , \\ f_i(X,Y,Z) &= f_{xor}(X,Y,Z) = X \oplus Y \oplus Z \ , \ 20 \le i \le 39 \ , \ 60 \le i \le 79 \ , \\ f_i(X,Y,Z) &= f_{maj}(X,Y,Z) = (X \,\&\, Y) \,|\, (X \,\&\, Z) \,|\, (Y \,\&\, Z) \ , \ 40 \le i \le 59 \ . \end{aligned}$$

Suppose now that the initial values A_0, B_0, C_0, D_0, E_0 are given. Then the compression function proceeds by the following steps for $0 \le i \le 79$ (additions are mod 2^{32}):

$$\begin{aligned} A_{i+1} &= A_i^{<<5} + f_i(B_i, C_i, D_i) + E_i + W_i + K_i \ , \\ B_{i+1} &= A_i \ , \\ C_{i+1} &= B_i^{<<30} \ , \\ D_{i+1} &= C_i \ , \\ E_{i+1} &= D_i \ . \end{aligned}$$

Finally compute the output of the compression function as

$$A = A_0 + A_{80} \ , \ B = B_0 + B_{80} \ , \ C = C_0 + C_{80} \ , \ D = D_0 + D_{80} \ , \ E = E_0 + E_{80} \ .$$

4.4.2.2 Security analysis

SHA-1 is conjectured to be a collision-resistant hash function. For an output length of 160 bits this means that the expected workload of generating a collision is of the order 2^{80}. The workload of finding a (second) preimage should be of the order 2^{160}. There have not been reported any attacks on SHA-1 falsifying these statements. An important effect of the expansion of the 16-word message block to 80 words in the compression function, is that for any two distinct 16-word blocks, the resulting 80-word values differ in a large number of bit positions. This makes the attack strategy that has been used on other algorithms of the MDx-family very difficult and certainly increases the strength of the algorithm. In the following we discuss some observations that have been made and an attack on a previous version of SHA.

Collisions for SHA-0.
An earlier version of SHA — commonly known as SHA-0 — has been cryptanalysed by Chabaud and Joux [117], resulting in a theoretical attack finding collisions (two messages hashing to the same value) with a complexity of about 2^{61} evaluations of the compression function. The only difference between these two versions of SHA is in the expansion procedure which is defined for SHA-0 by

$$W_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \ , \ \ 16 \leq i \leq 79$$

(in contrast to SHA-1 there is no rotation by one bit position to the left).

The general idea of the attack on SHA-0 is to track the propagation of local perturbations and to look for differential masks that can be added to the input block with non-trivial probability of keeping the output of the compression function unchanged. In [117] Chabaud and Joux have first studied a simplified variant of SHA-0 called SHI1 which replaces the modular additions ($+$) and the nonlinear functions (f_i) by bitwise addition ($\oplus$). Single bit errors or perturbations are introduced to the input of SHI1 and the perturbations are traced through the compression function. These perturbations are made to disappear by introducing five other bit errors or corrections. This allows an attack on SHI1 via differential masking. A second variant of SHA-0 called SHI2 is then analysed, where SHI2 replaces modular addition by $\oplus$ but this time keeps the nonlinear functions f_i. However we can still view f_i as acting like $\oplus$ with some probability, and the probability of a successful perturbation attack can be computed as 2^{-24}. A third variant of SHA-0, SHI3, is then analysed, where SHI3 uses modular addition as in SHA-0, but uses $\oplus$ instead of the nonlinear f_i. In this case the additions mod 2^{32} cause the perturbations to spread out due to carry propagation. However one is still able to devise a perturbation attack on SHI3 with probability 2^{-44}. Finally SHA-0 itself is analysed by taking into account the analyses of SHI2 and SHI3, and this leads to a perturbation based attack on SHA-0 requiring 2^{61} evaluations of the compression function.

It should be emphasised that, although SHA-1 and SHA-0 are so similar, this attack does not carry over to SHA-1. The rotation by one bit position to the left which is added in the expansion procedure of SHA-1 means that the linear code of the expansion no longer operates on bit level: a modification of a single bit influences bits at other positions in the words as well. This makes the attack strategy of [117] completely ineffective and provides strong evidence that the transition from SHA-0 to SHA-1 raised the level of security.

Slid pairs in SHA-1.
Recently Saarinen [458] has noted that a slide attack can be mounted on SHA-1 with about 2^{32} effort. Although it is difficult to see how this attack could be exploited to find collisions or preimages for the hash function, the analysis demonstrates an unexpected property of the compression function. Consider two messages $M = [W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}]$ and $M' = [W'_0 \,\|\, W'_1 \,\|\, \ldots \,\|\, W'_{15}]$. The main observation is that the procedure for message expansion can be slid. We simply choose $W'_i = W_{i+1}$ for $0 \leq i \leq 14$ and $W'_{15} = (W_0 \oplus W_2 \oplus W_8 \oplus W_{13})^{<<1}$. After message expansion the following is true: $W'_i = W_{i+1}$ for $0 \leq i \leq 78$. The

second observation is that for 20 steps in each round of the compression function, the boolean function f_i and the additive constant K_i are unchanged. Hence any two consecutive steps (step i and step $i+1$) of the compression function are similar, except for three transitions between different rounds (this happens for $i = 19, 39, 59$).

Suppose now that the hashing computation for M and M' starts from initial values A, B, C, D, E and A', B', C', D', E' respectively, which are related as follows:

$$B' = A \ , \ C' = B^{>>30} \ , \ D' = C \ , \ E' = D \ .$$

Then the purpose of the attack is to find messages and initial values for which the same relation (between the chaining variables) still holds at the end of the compression function. Such a pair of messages and corresponding initial values is called a "slid pair". The method for finding a slid pair is rather technical but the general strategy is to choose suitable values for the chaining variables in steps $i = 19$ and $i = 39$, and perform a meet-in-the-middle match. This procedure is repeated until the transition for $i = 59$ is also dealt with, which happens with probability 2^{-32}. The overall time and space complexity of the attack are of the order 2^{32}.

Differential analysis.
Differential cryptanalysis has been applied to the encryption mode of SHA-1 (see below) and is relevant for the hash function as well. Although the propagation of a difference in the initial value through the compression function does not immediately help in finding preimages or collisions for the hash function, it may point to unwanted properties of the compression function.

For SHA-1, there exists a 5-step differential characteristic over any 5 steps in the compression function with probability 1. Handschuh and Naccache [237] conjecture that over 80 steps the best differential characteristic has probability around 2^{-116}. It is emphasised that this estimation is over-favourable to the cryptanalyst as it would be impossible to connect up all the constituent characteristics so as to achieve these biases. Van den Bogaert and Rijmen [504] have searched for optimal differential characteristics under the requirement that the Hamming Weight of every 32-bit word of the input is upper bounded by 2. It was found that there are two 10-step characteristics for f_{if} with probability 2^{-12} (this is a factor of 2 better than [237]), a 10-step characteristic for f_{xor} with best case probability 2^{-12}, and a 20-step characteristic for f_{if} and f_{maj} with probabilities 2^{-42} and 2^{-41} respectively (these figures agree with [237]).

Kim *et al.* [290] have found a 28-step differential characteristic with probability 2^{-107}, and a 30-step differential characteristic with probability 2^{-138}. This shows that when the number of steps increases, the probability of a differential characteristic decreases rapidly (this is due to the fact that the Hamming Weight in the difference words grows larger). Overall the security margin of SHA-1 against this type of analysis appears very large.

Encryption mode of SHA-1.
The SHA-1 compression function can also be used in encryption mode, by inserting a key as message (so the expansion procedure is used as key schedule) and a plaintext as initial value, while leaving out the feed-forward operation at the end of the compression function. The resulting 160-bit block cipher, called SHACAL-1, has been submitted to NESSIE (see Chapter 2). One may also view SHA-1 as a Davies-Meyer hashing scheme based on SHACAL-1.

In the submission document Handschuh and Naccache [237] conjecture that a linear cryptanalytic attack on SHACAL-1 would require at least 2^{80} known plaintexts and that a differential attack would require at least 2^{116} chosen plaintexts. These estimations cannot be considered a *break* of the algorithm because Handschuh and Naccache construct very loose lower bounds. Differential cryptanalysis including boomerang attacks [290] and rectangle attacks [68] have been applied to SHACAL-1. The best known attack works for 49 steps of the compression function with a data complexity of $2^{151.9}$ chosen plaintexts and a time complexity of $2^{508.5}$ [68].

Saarinen [460] recently showed that the slide attack on SHA-1 also points to a weakness in the key schedule of SHACAL-1, and this can be exploited in a related-key attack. Given access to two SHACAL-1 encryption oracles whose keys are "slid" (in the same way that the message expansion can be slid for the hash function) the cipher can be distinguished from a randomly chosen 160-bit permutation. This requires about 2^{128} chosen plaintexts. When certain properties hold for the (related) keys, the complexity can be further reduced to about 2^{96} chosen plaintexts.

4.4.3 SHA-2

SHA-2 [393] has recently been included in the new NIST hash function standard (FIPS 180-2), generating hash values of 256, 384 or 512 bits. The main reason for the new standard is to provide a security level comparable to the security level of the new NIST block cipher standard AES (with keylength of 128, 192 or 256 bits respectively). Therefore, SHA-2 will serve as a benchmark for the submissions in this category.

4.4.3.1 SHA-2/256

The SHA-2/256 hash function operates on blocks of 512 bits divided in words of 32 bits and produces a message digest (hash value) of 256 bits. The algorithm is defined as the iteration of a compression function which we specify below. The computation starts with the initial value

$$\begin{aligned} IV \;=\; & \texttt{6a09e667}_x\ \texttt{bb67ae85}_x\ \texttt{3c6ef372}_x\ \texttt{a54ff53a}_x \\ & \texttt{510e527f}_x\ \texttt{9b05688c}_x\ \texttt{1f83d9ab}_x\ \texttt{5be0cd19}_x\ . \end{aligned}$$

Each application of the compression function uses eight words as initial values and 16 words of the message as input and it gives eight words output, which are

then used as initial values for the next application. The final output is the hash value. This works because there is a padding rule which adds bits to the message (including a representation of the message length) so that its length becomes a multiple of 512 bits.

Compression function of SHA-2/256.
Let the message block of 512 bits be denoted $M = [W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}]$, where W_i are 32-bit words. SHA-2/256 uses an expansion procedure defined by

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \ , \ \ 16 \le i \le 63 \ ,$$

with σ_0 and σ_1 defined as follows (where $(\cdot)^{>}$ denotes shift and $(\cdot)^{>>}$ rotation or circular shift to the right):

$$\begin{aligned} \sigma_0(X) &= X^{>>7} \oplus X^{>>18} \oplus X^{>3} \ , \\ \sigma_1(X) &= X^{>>17} \oplus X^{>>19} \oplus X^{>10} \ . \end{aligned}$$

Define the following functions:

$$\begin{aligned} Ch(X,Y,Z) &= (X \,\&\, Y) \oplus (\bar{X} \,\&\, Z) \ , \\ Maj(X,Y,Z) &= (X \,\&\, Y) \oplus (X \,\&\, Z) \oplus (Y \,\&\, Z) \ , \\ \Sigma_0(X) &= X^{>>2} \oplus X^{>>13} \oplus X^{>>22} \ , \\ \Sigma_1(X) &= X^{>>6} \oplus X^{>>11} \oplus X^{>>25} \ . \end{aligned}$$

Suppose now that the initial values $A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0$ are given. Then the compression function proceeds by the following steps for $0 \le i \le 63$ (additions are mod 2^{32}):

$$\begin{aligned} A_{i+1} &= \Sigma_0(A_i) + Maj(A_i,B_i,C_i) + \Sigma_1(E_i) + Ch(E_i,F_i,G_i) + H_i + W_i + K_i \ , \\ B_{i+1} &= A_i \ , \\ C_{i+1} &= B_i \ , \\ D_{i+1} &= C_i \ , \\ E_{i+1} &= D_i + \Sigma_1(E_i) + Ch(E_i,F_i,G_i) + H_i + W_i + K_i \ , \\ F_{i+1} &= E_i \ , \\ G_{i+1} &= F_i \ , \\ H_{i+1} &= G_i \ . \end{aligned}$$

The 32-bit constants K_i are different in each of the 64 steps. We refer to the description of SHA-2 [393] for their exact value. Finally compute the output of the compression function as

$$\begin{aligned} &A = A_0 + A_{64} \ , \ \ B = B_0 + B_{64} \ , \ \ C = C_0 + C_{64} \ , \ \ D = D_0 + D_{64} \ , \\ &E = E_0 + E_{64} \ , \ \ F = F_0 + F_{64} \ , \ \ G = G_0 + G_{64} \ , \ \ H = H_0 + H_{64} \ . \end{aligned}$$

4.4.3.2 SHA-2/512

The main difference between SHA-2/256 and SHA-2/512 is that the latter uses a wordlength of 64 bits (instead of 32 bits). This allows the computation of a message digest which is twice as long compared to SHA-2/256, without changing the structure of the algorithm. Another distinction is that the number of steps in the compression function has been changed from 64 to 80. Hence, the SHA-2/512 hash function operates on blocks of 1024 bits divided in words of 64 bits and produces a message digest (hash value) of 512 bits. The algorithm is defined as the iteration of a compression function which we specify below. The computation starts with the initial value

$$\begin{aligned} IV \;=\; & \texttt{6a09e667f3bcc908}_x \;\; \texttt{bb67ae8584caa73b}_x \;\; \texttt{3c6ef372fe94f82b}_x \;\; \texttt{a54ff53a5f1d36f1}_x \\ & \texttt{510e527fade682d1}_x \;\; \texttt{9b05688c2b3e6c1f}_x \;\; \texttt{1f83d9abfb41bd6b}_x \;\; \texttt{5be0cd19137e2179}_x \;\; . \end{aligned}$$

Each application of the compression function uses eight words as initial values and 16 words of the message as input and it gives eight words output, which are then used as initial values for the next application. The final output is the hash value. This works because there is a padding rule which adds bits to the message (including a representation of the message length) so that its length becomes a multiple of 1024 bits.

Compression function of SHA-2/512.
Let the message block of 1024 bits be denoted $M = [W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}]$, where W_i are 64-bit words. SHA-2/512 uses an expansion procedure defined by

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \;\;,\;\; 16 \leq i \leq 79 \;\;,$$

with σ_0 and σ_1 defined as follows (where $(\cdot)^{>}$ denotes shift and $(\cdot)^{>>}$ rotation or circular shift to the right):

$$\begin{aligned} \sigma_0(X) &= X^{>>1} \oplus X^{>>8} \oplus X^{>7} \;\;, \\ \sigma_1(X) &= X^{>>19} \oplus X^{>>61} \oplus X^{>6} \;\;. \end{aligned}$$

Define the following functions:

$$\begin{aligned} Ch(X,Y,Z) &= (X \,\&\, Y) \oplus (\bar{X} \,\&\, Z) \;\;, \\ Maj(X,Y,Z) &= (X \,\&\, Y) \oplus (X \,\&\, Z) \oplus (Y \,\&\, Z) \;\;, \\ \Sigma_0(X) &= X^{>>28} \oplus X^{>>34} \oplus X^{>>39} \;\;, \\ \Sigma_1(X) &= X^{>>14} \oplus X^{>>18} \oplus X^{>>41} \;\;. \end{aligned}$$

Suppose now that the initial values $A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0$ are given. Then the compression function proceeds by the following steps for $0 \leq i \leq 79$ (additions are mod 2^{64}):

$$
\begin{aligned}
A_{i+1} &= \Sigma_0(A_i) + Maj(A_i, B_i, C_i) + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + H_i + W_i + K_i \ , \\
B_{i+1} &= A_i \ , \\
C_{i+1} &= B_i \ , \\
D_{i+1} &= C_i \ , \\
E_{i+1} &= D_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + H_i + W_i + K_i \ , \\
F_{i+1} &= E_i \ , \\
G_{i+1} &= F_i \ , \\
H_{i+1} &= G_i \ .
\end{aligned}
$$

The 64-bit constants K_i are different in each of the 80 steps. We refer to the description of SHA-2 [393] for their exact value. Finally compute the output of the compression function as

$$
\begin{aligned}
&A = A_0 + A_{80} \ , \ B = B_0 + B_{80} \ , \ C = C_0 + C_{80} \ , \ D = D_0 + D_{80} \ , \\
&E = E_0 + E_{80} \ , \ F = F_0 + F_{80} \ , \ G = G_0 + G_{80} \ , \ H = H_0 + H_{80} \ .
\end{aligned}
$$

4.4.3.3 SHA-2/384

The SHA-2/384 hash function is defined in the exact same manner as SHA-2/512 with the following two exceptions:

The computation starts with a different initial value:

$$
\begin{aligned}
IV \ = \ & \texttt{cbbb9d5dc1059ed8}_x \ \texttt{629a292a367cd507}_x \ \texttt{9159015a3070dd17}_x \ \texttt{152fecd8f70e5939}_x \\
& \texttt{67332667ffc00b31}_x \ \texttt{8eb44a8768581511}_x \ \texttt{db0c2e0d64f98fa7}_x \ \texttt{47b5481dbefa4fa4}_x \ .
\end{aligned}
$$

The 384-bit message digest is obtained by truncating the final hash value to its left-most 384 bits.

4.4.3.4 Security analysis

SHA-2/256, SHA-2/384 and SHA-2/512 are conjectured to be collision-resistant hash functions. This means that the expected workload of generating a collision is of the order $2^{n/2}$ and that the workload of finding a (second) preimage should be of the order 2^n, where n denotes the output length which, for these hash function, is equal to 256, 384 or 512 bits respectively. There have not been reported any attacks falsifying this statement.

SHA-2 is a new design that has some similarities to SHA-1, but there are important differences in the structure. We may note that for the 256-bit version the number of steps in the compression function is lower than for SHA-1 (64 steps compared to 80). On the other hand, two variables are updated in every step of the compression function where for SHA-1 only one variable is updated in a step. With respect to differential cryptanalysis, there exists a 4-step characteristic over any 4 steps in the compression function with probability 1. The probability of differential characteristics appears to decrease faster than for SHA-1. This is due to the multiple rotations in the functions $\Sigma_0(\cdot)$ and $\Sigma_1(\cdot)$. The slide attack on

SHA-1 does not extend to SHA-2 because for SHA-2 every step of the compression function uses a unique additive constant. SHA-2 is a recently designed primitive of which the design strategy was not made public, so more time is needed to perform a careful and thorough security evaluation.

Encryption mode of SHA-2.
The SHA-2 compression function can also be used in encryption mode, by inserting a key as message (so the expansion procedure is used as key schedule) and a plaintext as initial value, while leaving out the feed-forward operation at the end of the compression function. For SHA-2/256 this results in a 256-bit block cipher, called SHACAL-2, which has been submitted to NESSIE (see Chapter 2). One may also view SHA-2/256 as a Davies-Meyer hashing scheme based on SHACAL-2. No security flaws have been identified for SHACAL-2. The weakness in the key schedule of SHACAL-1 does not extend to SHACAL-2.

4.5 Conclusion

The NESSIE project has studied the submitted algorithm Whirlpool and has also studied the NIST hash function standards SHA-1 and SHA-2. No significant security weaknesses have been found for any of these hash functions. The best result on Whirlpool is an attack which finds non-random properties in versions of Whirlpool where the compression function is reduced to six rounds or less (the complete function uses ten rounds). For SHA-1 a slide attack has been found that demonstrates an unexpected property of the compression function, but this is not a threat for any normal use of the hash function. However, this attack also points to a weakness in the key schedule of the encryption mode of SHA-1. SHA-2 is a new design with significant differences from SHA-1; no weaknesses have been reported for it.

Changes from version 1.0 to version 2.0 of the document

—	Typos have been corrected.
§4.1	Introduction expanded and reference to Preneel added.
§4.2.1	Formal security model added.
§4.2.2	Second paragraph has been moved from §4.2.1 to here. Subsection on side-channel attacks added.
§4.3.1	Miyaguchi-Preneel scheme added.
§4.3.4	Section rewritten. Note added on current revision of ISO/IEC 10118-3 (as pointed out by P. Barreto). More details on ANSI standards.
§4.4.2.2	Security analysis for SHA-1 expanded. More details on the SHA-0 collision attack, on differential cryptanalysis, and on the security claims and cryptanalysis of the encryption mode SHACAL-1.
§4.4.3.4	Security analysis for SHA-2 expanded.

5. Message authentication codes

5.1 Introduction

Message Authentication Codes, also known as MACs, are cryptographic primitives used for information authentication. A MAC is a function that takes an input of arbitrary length and produces an output of a fixed length. Contrary to hash functions, the computation of a MAC depends on a secret key. In practical applications this key has to be shared between two parties (a sender and a receiver) so MACs are used in a symmetric setting, contrary to digital signatures (see Chapter 7) which are used for authentication in asymmetric settings. We informally give the conditions we require of a message authentication code:

– **A MAC** with an unknown key should be "hard" to forge on a new message, even when many messages and corresponding MAC values are known.

Before presenting a detailed analysis of the message authentication codes studied by the NESSIE project, we first discuss the security requirements and give an overview of common designs and current standards. For a more comprehensive overview of cryptographic primitives for information authentication (including message authentication codes) we refer to the treatment by Preneel in [440].

5.2 Security requirements

In this section we give practical and formal definitions for message authentication codes and describe the general model of an iterated MAC. We then discuss different types of attacks on message authentication codes and describe the assessment process followed by the project.

5.2.1 Security model

The following informal definition for message authentication codes was given by Preneel in [440]. A MAC is a function h satisfying the following conditions:

1. The argument X can be of arbitrary length and the result $h(K, X)$ has a fixed length of n bits, where the secondary input K denotes the secret key.

[0] Coordinator for this chapter: KUL — Bart Van Rompay

2. Given h and X (but with unknown K), it must be 'hard' to determine $h(K, X)$ with a probability of success 'significantly higher' than $1/2^n$. Even when a large set of pairs $\{X_i, h(K, X_i)\}$ is known, it is 'hard' to determine the key K or to compute $h(K, X')$ for any $X' \neq X_i$.

Most MACs are iterated constructions, in the sense that they are based on a compression function with fixed size inputs; they process every message block in a similar way. The input X is padded by an unambiguous padding rule to a multiple of the block size. Typically this also includes adding the total length in bits of the input. The padded input is then divided into t blocks denoted X_1 through X_t. The MAC involves a *compression function* f and a *chaining variable* H_i between stage $i-1$ and stage i:

$$\begin{aligned} H_0 &= IV_K \ , \\ H_i &= f_K(H_{i-1}, X_i) \ , \ 1 \leq i \leq t \ , \\ h(K, X) &= g_K(H_t) \ . \end{aligned}$$

Here IV denotes the Initial Value and g the *output transformation*. The secret key K may be employed in the IV, in the compression function, and/or in the output transformation.

5.2.1.1 Formal definitions

Before discussing security aspects we first give more precise definitions of a MAC and its cryptographic strength. These definitions are similar to the definitions given by Krovetz in [316]. For an iterated construction similar definitions can be given as for iterated hash functions in Sect. 4.2.1.1 (taking into account the MAC key, and the fact that there usually is an additional output transformation).

Definition 5.1. *A* **MAC** *is a function $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{R}$ where the key space $\mathcal{K} = \{0,1\}^k$, the message space $\mathcal{M} = \{0,1\}^*$ and the range $\mathcal{R} = \{0,1\}^n$ for some $k, n \geq 1$. When given a key $K \in \mathcal{K}$ and a message $X \in \mathcal{M}$, the function produces a MAC value $Y \in \mathcal{R}$.*

Definition 5.2 (Unforgeability). *An adversary has forged a message for a MAC h if, without knowledge of a random key K, he is able to produce a new message X and MAC value Y such that $h(K, X) = Y$. A MAC $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{R}$ is (t, ϵ, q)-secure if, under a randomly chosen key K, an adversary cannot forge a new message in time t with probability better than ϵ even if he is provided with the MAC values of q other messages of his choice.*

5.2.2 Classification of attacks

Depending on the information available to an adversary, the following types of attacks are distinguished for MACs:

– *Known text attack.* An attacker is able to examine some plaintexts and their corresponding MAC values.

– *Chosen text attack.* An attacker is able to select a set of texts and subsequently obtains a list of MAC values corresponding to these texts.
– *Adaptive chosen text attack.* This is the most general attack where an attacker chooses a text and immediately receives the corresponding MAC value. The choice of a text can depend on the outcome of previous queries.

"Breaking" a MAC algorithm can have different meanings:

– *Existential forgery.* An attacker can determine the MAC value for at least one text. As he has no control over this text, it may be random or nonsensical.
– *Selective forgery.* An attacker can determine the MAC value for a particular text chosen a priori by him. Note that practical attacks often require that a forgery is *verifiable*, which means that the attacker knows that the forged MAC is correct with probability close to 1.
– *Key recovery.* This means that an attacker can determine the secret key K. Such a break is more powerful than a forgery, since it allows for arbitrary selective forgeries.

In practice, an adaptive chosen text attack may not always be feasible; moreover, forgeries typically need to have specific redundancy to be of any practical use. However, it is in general better to be conservative and to require that MAC algorithms resist against the strongest attacks possible. We now briefly describe the best known attacks on message authentication codes (see [440]).

Guessing of the MAC

A straightforward attack on a MAC algorithm consists of choosing an arbitrary new message, and subsequently guessing the MAC value. This can be done in two ways: either one guesses the MAC value directly, with a success probability of 2^{-n}, or one guesses the key, and then computes the MAC value, with success probability 2^{-k}. Here n denotes the size in bits of the MAC value and k the size in bits of the secret key. This is a non-verifiable attack: an attacker does not know a priori whether his guess was correct. The feasibility of the attack depends on the number of trials that can be performed and on the expected value of a successful attack; both are strongly application dependent.

Exhaustive Key Search

This is another straightforward attack that can be applied to any algorithm. The attack requires approximately k/n known text-MAC pairs for a given key; one attempts to determine the key by trying one-by-one all the keys. The expected number of trials is equal to 2^{k-1}. In contrast to the previous attack, this attack is carried out off-line and it yields a complete break of the MAC algorithm.

Internal Collision Based Forgery

The observation behind this attack is that if one can find an internal collision (this is a collision which occurs before the output transformation g), this can be used to construct a verifiable MAC forgery based on a single chosen text.

Preneel and van Oorschot [442] have shown that an internal collision for h can be found using u known text-MAC pairs and v chosen texts, where the expected values for u and v are as follows (here l denotes the size in bits of the chaining variable): $u = \sqrt{2} \cdot 2^{l/2}$ and $v = 0$ if the output transformation g is a permutation; otherwise, v is approximately

$$2\left(2^{l-n} + \left\lfloor \frac{l-n}{n-1} \right\rfloor + 1\right) \ .$$

Further optimisations of this attack are possible if the set of text-MAC pairs has a common sequence of s trailing blocks.

Internal Collision Based Key Recovery

For some compression functions, one can extend the internal collision attack to a key recovery attack [443]. The idea is to identify one or more internal collisions; for example, if f is not a permutation for fixed H_i, an internal collision after the first message block gives the equation $f_K(H_0, X_1) = f_K(H_0, X_1')$, in which K and possibly H_0 are unknown (it is assumed that the $H_0 = IV$ is key dependent). For some compression functions f, one can obtain information on the secret key based on such relations.

Divide-and-Conquer Attack

This attack is a special case of an internal collision based key recovery. For some compression functions that use two seperate keys, it is possible to exploit internal collisions for a divide-and-conquer key recovery attack [442]. Let the keys be denoted by K_1, K_2. The general idea is that an attacker first looks for some internal collisions, and then searches exhaustively for a key K_1 that produces these collisions. Once K_1 is determined, an exhaustive search is used to find K_2. Therefore, the strength of such a MAC comes from its individual keys and not from their combined length (although the attack is less practical than a simple exhaustive key search, as it needs a large number of known text-MAC pairs).

Exor Forgery

This type of forgery only works if the value of H_i is computed as a function of $H_{i-1} \oplus X_i$, and if no output transformation is present. The easiest variant of the attack requires only a single known text-MAC pair. Assume that the input X and its padded version $\bar{X}$ consist of a single block. Assume that one knows $h(K, X)$; it follows immediately that $h(K, \bar{X} \,\|\, (X \oplus h(K, X))) = h(K, X)$. This implies that one can construct a new message with the same MAC value, which is a forgery.

Side-Channel Attacks

Side-channel attacks are a major threat for all implementations of cryptographic algorithms. Side-channel attacks on MAC algorithms are similar as for other symmetric primitives. For a detailed discussion on side-channel attacks and countermeasures that can be applied to protect an implementation we refer to Annex A.

5.2.3 Assessment process

The MAC submissions were assessed with reference to the above generic common MAC attacks and by specific attacks when appropriate. Statistical analysis was carried out for various input lengths. MACs were submitted to the dependence test and linear factors test described in Sect. 2.2.4. If the NESSIE submissions were based on compression functions, these were also submitted to the two tests. Furthermore the stream cipher tests described in Sect. 2.2.4 were also applied on the MACs both in output feedback and counter mode. None of the MACs tested exhibited any anomalous behaviour.

5.3 Overview of the common designs

There are few algorithms that are designed for the specific purpose of message authentication. In most cases a message authentication code is constructed from a block cipher or from a hash function as discussed below. A different approach is the use of families of universal hash functions. We also summarise the procedures which have been undertaken by several organisations for the standardisation of message authentication codes.

5.3.1 MACs based on block ciphers

The most common way of basing a MAC on a block cipher is by using the cipher in CBC-mode (cipher block chaining): the MAC key is used as cipher key in each step of the iteration, and the message block to be processed in the current step serves as plaintext input to the cipher, after being added bit by bit to the ciphertext output from the previous step:

$$H_1 = E_K(X_1) \ , \ \ H_i = E_K(X_i \oplus H_{i-1}) \ \ (2 \leq i \leq t) \ .$$

Here we assume that the message X (after padding) is divided into blocks $X_1, \ldots, X_t$ with length appropriate for the block cipher used. E_K denotes encryption with secret key K and H_t forms the output of the MAC algorithm.

The basic CBC-construction is susceptible to the exor forgery attack described in Sect. 5.2.2, therefore it can only be used in applications where the messages have a fixed length. Several more secure variations on the scheme exist however. One example, known as EMAC[1], is the use of an additional encryption as output transformation, where the key for this encryption operation may be derived from the MAC key. Another example, commonly known as the retail-MAC, replaces the last encryption by a two-key triple encryption. The security of these constructions can be proven based on the assumption that the underlying block cipher is pseudo-random [431].

All of these schemes are included in ISO/IEC 9797-1 [256], a standard for MACs using an (unspecified) block cipher.

[1] EMAC stands for Encrypted-MAC. This construction is also known as DMAC (Double-MAC).

5.3.2 MACs based on hash functions

A message authentication code can also be obtained from a hash function by including a secret key in the computation. This is a common approach because these MACs are usually faster than MACs which are based on a block cipher. However, simply prepending or appending the key material to the message input of the hash function does not result in a secure MAC.

HMAC is a nested construction that computes a MAC, for an underlying hash function h, message X and secret key K, as follows:

$$HMAC(K, X) = h((K \oplus opad) \,\|\, h((K \oplus ipad) \,\|\, X)) \ .$$

The key K is first padded with zero bits to a full block, and *opad* and *ipad* are constant values. Bellare *et al.* [36] have proven the security of this construction under the following assumptions: the underlying hash function is collision-resistant for a secret initial value; the compression function keyed by the initial value is a secure MAC algorithm (for messages of one block); the compression function is a weak pseudo-random function.

An alternative to HMAC are the MDx-MAC constructions [442] which can be based on MD5, SHA, RIPEMD or similar hash functions. Here, the underlying hash function is converted into a MAC by small modifications, involving the secret key at the beginning, at the end and in every iteration of the hash function. This is achieved by key-dependent modification of the initial value and the additive constants used by the hash function, and by a key-dependent output transformation. The security of MDx-MAC can be proven based on the assumption that the underlying compression function is pseudo-random.

Both HMAC and MDx-MAC are included in ISO/IEC 9797-2 [257], a standard for MACs using a dedicated hash function. The NESSIE submission TTMAC (see Sect. 5.4.1) is also based on a hash function, more specifically the RIPEMD-160 hash function.

5.3.3 MACs based on universal hashing

A family of hash functions $H = \{h : \mathcal{D} \to \mathcal{R}\}$ is a finite set of functions with common domain $\mathcal{D}$ and (finite) range $\mathcal{R}$. We may also denote this by $H : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ where $H_K : \mathcal{D} \to \mathcal{R}$ is a function in the family for each $K \in \mathcal{K}$. In the latter case, one chooses a random function h from the family by choosing $K \in \mathcal{K}$ uniformly and letting $h = H_K$.

A universal hash function family is a family of hash functions with some combinatoric property. For example, a hash function family $H = \{h : \mathcal{D} \to \mathcal{R}\}$ is "ϵ-almost-universal" if for any distinct $X, X' \in \mathcal{D}$, the probability that $h(X) = h(X')$ is no more than ϵ, when $h \in H$ is chosen at random.

This can be used for message authentication, for example by hashing a message with a function drawn from a universal hash function family, encrypting the output of the hash function, and then producing the encrypted hash output as MAC value. It can be proven that the security of the resulting MAC scheme

depends on the security of the cipher used for encrypting the hash output. The combinatoric property of the universal hash function family is often not difficult to prove, and the resulting MAC schemes are the fastest MACs around. The NESSIE submission UMAC (see Sect. 5.4.2) is an example of a MAC based on universal hashing.

5.3.4 Current standards

Several organisations have taken initiatives for the standardisation of message authentication codes. ISO/IEC has developed standard 9797 for MACs, with two separate parts. Part 9797-1 [256] describes MACs based on a block cipher, more specifically the CBC-MAC for an unspecified block cipher (with some optional extensions including EMAC and retail-MAC). Part 9797-2 [257] details MACs based on a dedicated hash function, more specifically the HMAC and MDx-MAC constructions for an unspecified hash function (and a variant of MDx-MAC for short input strings only).

ANSI has adopted the DES-based CBC-MAC (including retail-MAC) in its banking standard X9.19 [13] and HMAC (with unspecified hash function) in standard X9.71 [17]. NIST has developed FIPS 113 [386] for DES-based CBC-MAC and FIPS 198 [394] for SHA-1-based HMAC.

5.4 MAC primitives considered during Phase II

The TTMAC and UMAC algorithms were submitted to NESSIE and selected for study during phase II of the NESSIE project. Furthermore, the schemes EMAC and HMAC, based on AES and SHA-1 respectively, and RMAC (a variant of EMAC) were selected for study during NESSIE phase II.

5.4.1 Two-Track-MAC

Two-Track-MAC (also known as TTMAC) is a message authentication code designed by Bert de Boer and Bart Van Rompay [507] and submitted to the NESSIE project. The design is based on the RIPEMD-160 hash function with modifications. The algorithm operates on blocks of 512 bits divided into words of 32 bits, uses a secret key of 160 bits, and generates an output of up to 160 bits.

5.4.1.1 The design

The design of Two-Track-MAC is based on the hash function RIPEMD-160. First, the message to be authenticated is padded with a 1-bit, and then 0-bits until its length is 448 mod 512. Then the binary representation of the length of the original message (mod 2^{64}) is appended, so the length of the message becomes a multiple of 512. Each 512-bit block is split into a set of sixteen 32-bit words, $W_0, \ldots, W_{15}$. The secret key is a set of five 32-bit words, $K_0, \ldots, K_4$. The algorithm works by iterating a compression function as follows.

Two sets of five 32-bit words $X_0, \ldots, X_4$ and $Y_0, \ldots, Y_4$ and a set of 16 message words are input to two different functions f_L and f_R that output five 32-bit words each:

$$\begin{aligned} A_0, \ldots, A_4 &= f_L(X_0, \ldots, X_4, W_0, \ldots, W_{15}) \ , \\ B_0, \ldots, B_4 &= f_R(Y_0, \ldots, Y_4, W_0, \ldots, W_{15}) \ . \end{aligned}$$

These two functions are identical to the ones used in RIPEMD-160 (the details are given below). Then we compute two new sets of five words each by subtracting the input words from the results of the previous step:

$$\begin{aligned} C_i &= A_i - X_i \mod 2^{32} \ , \ 0 \le i \le 4 \ , \\ D_i &= B_i - Y_i \mod 2^{32} \ , \ 0 \le i \le 4 \ . \end{aligned}$$

To finish the compression function, the ten words C_i and D_i are mixed in two linear transformations g_L and g_R:

$$\begin{aligned} E_0, \ldots, E_4 &= g_L(C_0, \ldots, C_4, D_0, \ldots, D_4) \ , \\ F_0, \ldots, F_4 &= g_R(C_0, \ldots, C_4, D_0, \ldots, D_4) \ . \end{aligned}$$

E_i and F_i together with the next set of message words, form the inputs to the next application of the compression function. In the first iteration, the five secret keywords K_i are input as both X_i and Y_i, $0 \le i \le 4$.

In the final iteration, where the last message words are input, f_L and f_R swap places. After the subtraction of the input words, instead of executing g_L and g_R at the end of this iteration, we compute

$$E_i = C_i - D_i \mod 2^{32} \ , \ 0 \le i \le 4 \ .$$

The results from these subtractions form the output of Two-Track-MAC. An optional output transformation is defined for the computation of shorter MAC values.

The functions f_L and f_R.

The functions f_L and f_R, which are known as the left and right trail of the compression function, are identical to the functions used in the compression function of RIPEMD-160. They consist of 80 sequential steps which we describe below. We first define the constants and functions that are used.

Additive constants:

$$\begin{aligned} k_i &= 00000000_x, \quad k'_i = 50a28be6_x, \quad 0 \le i \le 15, \\ k_i &= 5a827999_x, \quad k'_i = 5c4dd124_x, \quad 16 \le i \le 31, \\ k_i &= 6ed9eba1_x, \quad k'_i = 6d703ef3_x, \quad 32 \le i \le 47, \\ k_i &= 8f1bbcdc_x, \quad k'_i = 7a6d76e9_x, \quad 48 \le i \le 63, \\ k_i &= a953fd4e_x, \quad k'_i = 00000000_x, \quad 64 \le i \le 79. \end{aligned}$$

Non-linear functions at bit level:

$$\begin{aligned}
f_i(x,y,z) &= x \oplus y \oplus z \ , \ 0 \le i \le 15 \ , \\
f_i(x,y,z) &= (x \,\&\, y) \,|\, (\bar{x} \,\&\, z) \ , \ 16 \le i \le 31 \ , \\
f_i(x,y,z) &= (x \,|\, \bar{y}) \oplus z \ , \ 32 \le i \le 47 \ , \\
f_i(x,y,z) &= (x \,\&\, z) \,|\, (y \,\&\, \bar{z}) \ , \ 48 \le i \le 63 \ , \\
f_i(x,y,z) &= x \oplus (y \,|\, \bar{z}) \ , \ 64 \le i \le 79 \ .
\end{aligned}$$

Selection of message word:

$$\begin{aligned}
r[i] &= i, \ 0 \le i \le 15 \\
r[i] &= 7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8, \ 16 \le i \le 31 \\
r[i] &= 3,10,14,4,9,15,8,1,2,7,0,6,13,11,5,12, \ 32 \le i \le 47 \\
r[i] &= 1,9,11,10,0,8,12,4,13,3,7,15,14,5,6,2, \ 48 \le i \le 63 \\
r[i] &= 4,0,5,9,7,12,2,10,14,1,3,8,11,6,15,13, \ 64 \le i \le 79 \\
r'[i] &= 5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12, \ 0 \le i \le 15 \\
r'[i] &= 6,11,3,7,0,13,5,10,14,15,8,12,4,9,1,2, \ 16 \le i \le 31 \\
r'[i] &= 15,5,1,3,7,14,6,9,11,8,12,2,10,0,4,13, \ 32 \le i \le 47 \\
r'[i] &= 8,6,4,1,3,11,15,0,5,12,2,13,9,7,10,14, \ 48 \le i \le 63 \\
r'[i] &= 12,15,10,4,1,5,8,7,6,2,13,14,0,3,9,11, \ 64 \le i \le 79
\end{aligned}$$

Rotation constants:

$$\begin{aligned}
s[i] &= 11,14,15,12,5,8,7,9,11,13,14,15,6,7,9,8, \ 0 \le i \le 15 \\
s[i] &= 7,6,8,13,11,9,7,15,7,12,15,9,11,7,13,12, \ 16 \le i \le 31 \\
s[i] &= 11,13,6,7,14,9,13,15,14,8,13,6,5,12,7,5, \ 32 \le i \le 47 \\
s[i] &= 11,12,14,15,14,15,9,8,9,14,5,6,8,6,5,12, \ 48 \le i \le 63 \\
s[i] &= 9,15,5,11,6,8,13,12,5,12,13,14,11,8,5,6, \ 64 \le i \le 79 \\
s'[i] &= 8,9,9,11,13,15,15,5,7,7,8,11,14,14,12,6, \ 0 \le i \le 15 \\
s'[i] &= 9,13,15,7,12,8,9,11,7,7,12,7,6,15,13,11, \ 16 \le i \le 31 \\
s'[i] &= 9,7,15,11,8,6,6,14,12,13,5,14,13,13,7,5, \ 32 \le i \le 47 \\
s'[i] &= 15,5,8,11,14,14,6,14,6,9,12,9,12,5,15,8, \ 48 \le i \le 63 \\
s'[i] &= 8,5,12,9,12,5,14,6,8,13,6,5,15,13,11,11, \ 64 \le i \le 79
\end{aligned}$$

Suppose that the initial values a_0, b_0, c_0, d_0, e_0 are given.[2] The function f_L (left trail of the compression function) consists of the following steps for $0 \le i \le 79$ (additions are mod 2^{32}):

[2] These values are $X_0, \ldots, X_4$ in the description above.

$$\begin{aligned} a_{i+1} &= e_i \ , \\ b_{i+1} &= (a_i + f_i(b_i, c_i, d_i) + W_{r[i]} + k_i)^{<<s[i]} + e_i \ , \\ c_{i+1} &= b_i \ , \\ d_{i+1} &= c_i^{<<10} \ , \\ e_{i+1} &= d_i \ . \end{aligned}$$

Similarly, when the initial values a_0, b_0, c_0, d_0, e_0 are given[3], the function f_R (right trail of the compression function) consists of the following steps for $0 \leq i \leq 79$ (additions are mod 2^{32}):

$$\begin{aligned} a_{i+1} &= e_i \ , \\ b_{i+1} &= (a_i + f_{79-i}(b_i, c_i, d_i) + W_{r'[i]} + k'_i)^{<<s'[i]} + e_i \ , \\ c_{i+1} &= b_i \ , \\ d_{i+1} &= c_i^{<<10} \ , \\ e_{i+1} &= d_i \ . \end{aligned}$$

The functions g_L and g_R.

The linear transformations g_L and g_R are used to mix the outputs of the two trails of the compression function. For inputs $C_0, \ldots, C_4$ and $D_0, \ldots, D_4$, the function g_L computes five words $E_0, \ldots, E_4$ as follows (operations are mod 2^{32}):

$$\begin{aligned} E_0 &= (C_1 + C_4) - D_3 \ , \\ E_1 &= C_2 - D_4 \ , \\ E_2 &= C_3 - D_0 \ , \\ E_3 &= C_4 - D_1 \ , \\ E_4 &= C_0 - D_2 \ . \end{aligned}$$

For inputs $C_0, \ldots, C_4$ and $D_0, \ldots, D_4$, the function g_R computes five words $F_0, \ldots, F_4$ as follows (operations are mod 2^{32}):

$$\begin{aligned} F_0 &= C_3 - D_4 \ , \\ F_1 &= (C_4 + C_2) - D_0 \ , \\ F_2 &= C_0 - D_1 \ , \\ F_3 &= C_1 - D_2 \ , \\ F_4 &= C_2 - D_3 \ . \end{aligned}$$

[3] These values are $Y_0, \ldots, Y_4$ in the description above.

5.4.1.2 Security analysis

The security of Two-Track-MAC can be proven based on the assumption that the underlying compression function is pseudo-random. This function is very similar to the compression function used by RIPEMD-160 [441], which is a well studied primitive for which no weaknesses have been reported.

The functions f_L and f_R consist of 80 step iterations, and use the different bit operations AND, OR, XOR and bit complementation. The step functions also include bit rotations and addition mod 2^{32}, and iterated 80 times it seems very hard to trace unknown key bits through it. No weaknesses in Two-Track-MAC have been reported so far.

It is worth noting that the only place where key material is used is in the initial value. The 160-bit output of the algorithm is the difference between two 160-bit quantities, so the knowledge of this difference still gives 160 bits of uncertainty about which two values produced it. In other words, guessing what these two values are, and computing backwards to find the input, i.e. the key, is no faster than guessing on the key directly.

The large size of the internal state (320 bits) in Two-Track-MAC gives the algorithm a high level of security against attacks based on internal collisions. If we denote the output length by m (values for m between 32 and 160 bits — in steps of 32 — are supported), the complexity of generic attacks on this primitive is as follows:

- About 2^{159} MAC computations and $160/m$ known text-MAC pairs are needed for an exhaustive key search.
- Guessing the MAC value has a success probability of 2^{-m}.
- Internal collision based attacks need about 2^{160} known text-MAC pairs and about 2^{320-m} chosen texts.

5.4.2 UMAC

UMAC is a message authentication code designed by Ted Krovetz, John Black, Shai Halevi, Hugo Krawczyk and Phillip Rogaway [317] and submitted to the NESSIE project. The design is based on families of universal hash functions (see Sect. 5.3.3 for a definition) and offers provable security in the sense that there are provable collision bounds for the compression, so that the security depends on the AES cipher which is used for the encipherment of a nonce and the derivation of key material. Compared to conventional MAC algorithms, UMAC offers the benefits of faster speed (especially for long messages) and provable security at the cost of greater complexity.

5.4.2.1 The design

The UMAC message authentication code evolved from an earlier version UMAC (1999). We first describe this first version, next the additions for the new version, and we also describe some practical specifications and parameter sets. Due to the complexity of the scheme we give only a general outline, for details we refer to the algorithm specification in [317].

The previous version of UMAC.

UMAC (1999) computes the MAC by first compressing the message by a fixed ratio using the NH universal hash function family. A nonce is then concatenated to the compressed message and the result processed by a PRF (pseudo-random function) to obtain the authentication tag. HMAC (based on a conventional hash function) and CBC-MAC (based on a block cipher) were proposed as PRF. In short we can say that the universal hash function family NH is used as an accelerant to HMAC or CBC-MAC.

NH works by dividing the message into blocks of a certain length (except for the last block which can be shorter). Each block is processed by adding key material with the same length to it (the same key is used for every block), and compressing it by multiplying pairs of words and adding the results (e.g., starting from a block of 1024 32-bit words one can obtain a compressed value of 64 bits, which means a compression factor of 512). All compressed blocks are then concatenated and length information is appended.

The algorithm has some parameters, like the block size and word size, the PRG (pseudo-random generator) used to compute the needed key material from the user key (NH needs a key with the same length as the blocks in which the message is divided), and the PRF used to process the compressed message and nonce. Furthermore it is possible to use the Toeplitz construction to reduce the chance of forgery (by applying NH several times with keys that are shifted versions of each other, and concatenating the results), and/or to use two-level hashing to reduce the amount of needed key material. There are some other variations that allow optimisation for certain architectures (e.g., MMX).

Most of the limitations of UMAC (1999) come from the fact that the message is compressed by a fixed ratio rather than to a fixed length. In the first case the authentication tag is computed with $\mathrm{PRF}(hash||nonce)$, in the second case it can be computed with $hash \oplus \mathrm{PRF}(nonce)$, which has some advantages (the use of the PRF is limited to the minimum, it does not have an input of unbounded length). NH, the universal hash function family used in UMAC (1999), could also compress to a fixed output length, but then it would need a key (generated by the PRG) with length equal to the message (the entire message would be treated as a single 'block' in the description given above).

The submitted version of UMAC.

The new version of UMAC (2000) introduces extra complexity to solve the problem of compressing the message to a fixed length so the MAC can be computed with $hash \oplus \mathrm{PRF}(nonce)$. The PRF part works by enciphering the nonce with a block cipher. The hash part (also called UHash), which compresses the message, consists of three different layers:

- Compression: The first layer uses the fast NH hash family to compress the message by a fixed ratio.
- Hash-to-fixed-length: The second layer uses the RP hash family, which is not as fast as NH but generates an output of fixed length using a fixed-length key.
- Strengthen-and-fold: The third layer uses the IP hash family, which reduces the length of its input to a more appropriate size.

The RP universal hash function family is polynomial-based. A string made of n words of bitlength w can be viewed as a polynomial of degree n over a finite field, where each word of the string serves as a coefficient. To compute the hash, one evaluates the polynomial for a randomly chosen point (the key). For efficiency reasons, the computations are performed in a prime field, using the largest prime less then 2^w. The function has been tweaked in order to allow expansion of the domain to arbitrary strings (allowing variations in length, and dealing with strings outside the prime field). RP stands for ramped polynomial hashing: small prime fields are used for short messages, and larger prime fields for longer messages, the reason being that computations in a small prime field are more efficient, but can only handle messages up to a certain length for a given collision probability (for polynomial hashing the collision bounds degrade linearly with the length of the message being hashed, for a given size of the key set). In fact a hybrid scheme is used, where for long messages a small prime field is used on the first part of the message. The result is a hash family with arbitrary length inputs and fixed-length outputs, using a fixed-length key. From a security point of view it adds little to the collision probability compared to the NH hash layer of the algorithm.

The layer with the IP universal hash function family reduces the length of its input because the RP hash layer generates outputs which are quite long compared to the collision probability which is offered (for all but the longest messages being authenticated many of the leading bits of the output string of the RP layer will be zeros). It is based on computing the inner-product over a prime field (multiplying input words with key words and adding the results). While doing this the collision probability from the previous layer of the algorithm is maintained.

Specifications.

Many options are available in an implementation of UMAC but two named parameter sets have been specified: **UMAC16** and **UMAC32**. They are based on the three-layer hash schemes UHash16 and UHash32 respectively, and use the AES (Rijndael) block cipher for enciphering the nonce. The PRG which computes (from the user key) the key material needed in the internal operation of UHash is also based on the AES, in output-feedback mode.

UHash16 uses 16-bit words, representing them as signed integers. The NH hash layer operates on blocks of 2Kbytes, which are compressed to 32-bit values (this corresponds to a compression ratio of 512). The collision probability is proved to be no more than 2^{-15}. The result is passed to the RP hash layer which computes an output string of a fixed length of 128 bits. The RP hash family is a ramped construction using three prime fields with a 32-bit, 64-bit and 128-bit prime modulus respectively. The message length is restricted to a maximum of 2^{64} bits, and it is proved that this layer adds only little (around 2^{-19}) to the collision probability. When the message being authenticated is short to begin with, the RP layer is not needed and it is skipped as an optimisation. The IP hash layer folds its 128-bit input into a 16-bit output, maintaining the collision probability of nearly 2^{-15}. The three-layer construction is iterated a number of times, with independent keys, to increase the length of the authentication tag

and decrease the chance of MAC forgery. The default number is four times, and concatenating the 16-bit output values one obtains a 64-bit MAC with forging probability 2^{-60}.

The main difference in UHash32 is that it uses 32-bit words and iterates over the three-layer scheme only twice (default). This gives differences in the implementation (the use of larger prime fields), but the analysis is mainly the same.

The advantages over the previous UMAC (1999) version are that the use of the cryptographic primitive (AES) is minimised, that (as a result) it is more efficient on short messages, and that it offers extra flexibility for the verifier: he can choose how many of the parallel iterations he wants to perform in the MAC computation, thereby trading computation time for assurance level.

5.4.2.2 Security analysis

The UMAC message authentication code is based on families of universal hash functions and offers provable security in the sense that there are provable collision bounds for the hashing part of the MAC computation, so the security in the end depends on the cryptographic primitive used for enciphering the nonce. The primitive stated in the specification is the AES (Rijndael) block cipher, which has had a lot of analysis during and after the AES process supporting its security claims. There is the added advantage that the encryption is only performed on a short nonce.

No flaws have been found in the security proof of UMAC. For the layer using the NH universal hash function family, a first security proof states that NH is 2^{-w}-almost-universal (this means that the collision probability is no more than 2^{-w}) for strings of equal length, when it operates on words of bitlength w. This corresponds to the use of NH on one block of fixed length of the message. A second security proof allows to extend this result to NH working on any pair of strings, like in the UMAC setting. The reason that the collision probability after the NH hash layer in the UHash16 scheme is 2^{-15} rather than 2^{-16} is that a signed rather than an unsigned version of NH is used. A variant of the first security proof shows that the signed version of NH is 2^{-w+1}-almost-universal.

UHash16 and UHash32 have two additional layers using the RP and IP universal hash function families, and they iterate the three-layer scheme four, respectively two times. It is proven that the hash family UHash16 is 4-wise $(2^{-15} + 2^{-18} + 2^{-28})$-almost-universal, and that the hash family UHash32 is 2-wise $(2^{-31} + 2^{-33})$-almost-universal[4].

5.4.3 CBC-constructions: EMAC and RMAC

The DES-based CBC-MAC [386] is an old NIST MAC standard that will probably be upgraded to AES-based CBC-MAC and the general scheme is also included in the ISO/IEC standard 9797-1. Therefore AES-based CBC-MAC is considered as a benchmark for the submissions in this category. The scheme uses the AES

[4] Stronger universality properties are also proven for UHash16 and UHash32, see [317].

(Rijndael) block cipher in a black box model and generates a MAC value of up to 128 bits. We consider EMAC[5] [431], a variant of CBC-MAC with an additional encryption at the end (this is one of the extensions included in ISO/IEC 9797-1). We also discuss another recently proposed variant called RMAC [392].

5.4.3.1 The design

The computation of EMAC for a secret key K and a message X — divided (after padding) in 128-bit blocks $X_1, \ldots, X_t$ — proceeds as follows (here E_K denotes encryption with the 128-bit block cipher AES using key K):

1. Compute $H_1 = E_{K1}(X_1)$.
2. For $i = 2, \ldots, t$: compute $H_i = E_{K1}(X_i \oplus H_{i-1})$.
3. Compute the output transformation: $H_{out} = E_{K2}(H_t)$. The key $K2$ may be derived from $K1$ by the following procedure: $K2 = K1 \oplus \texttt{f0f0 ...f0}_x$.
4. To obtain an m-bit MAC value, select the leftmost m bits of H_{out}.

RMAC is a randomised variant of this scheme, offering improved resistance against attacks based on internal collisions. The only difference is in the output transformation where one encrypts with a key that is obtained by bitwise addition of $K2$ and a salt R:

$$H_{out} = E_{K2 \oplus R}(H_t) \ .$$

For RMAC, the keys $K1$ and $K2$ may be independent or they can be derived from one master key in a standard way. The salt R is r bits long and should be padded with 0-bits if it is shorter than $K2$. Five different parameter sets have been defined for the sizes m and r.

5.4.3.2 Security analysis

The security of EMAC can be proven based on the assumption that the underlying block cipher is pseudo-random [431], in this case Rijndael which has received a lot of analysis during and after the AES process supporting its security claims.

It is worth noting that without the additional encryption at the end (or when $K2$ would be chosen equal to $K1$), a simple (adaptive chosen text) existential forgery would be possible for the CBC-MAC scheme (due to an exor forgery attack). If $K1$ and $K2$ were to be chosen independently, the level of security against key recovery attacks would be less than suggested by the algorithms key size (due to a divide-and-conquer attack). An internal collision based forgery needs about 2^{64} known text-MAC pairs and 1 chosen text when the output length is 128 bits. More chosen texts are required when the MAC output is truncated, e.g., when the leftmost 64 bits are chosen, the attack needs about 2^{64} known pairs and 2^{64} chosen texts. On the other hand this increases the probability of success for an attack where one tries to guess the MAC value.

The main advantage of the randomised variant RMAC is that it offers improved resistance against attacks that are based on internal collisions. For example, when parameters $m = 128$ and $r = 128$ are chosen, such an attack requires about 2^{128} known pairs and 1 chosen text. On the other hand RMAC needs

[5] This construction was previously known as DMAC.

stronger assumptions for its security proof; for instance, the underlying block cipher must be secure against related-key attacks. In Appendix A of [392] it is noted that for RMAC with two independent keys $K1$ and $K2$ an exhaustive search for the keys is expected to require the generation of 2^{2k-1} MACs, where k is the size of one key. However, as noted in [300], this can be done much faster for parameters $m = 128$ and $r = 128$: under a chosen message attack with just one known message and one chosen message $K2$ can be found with about 2^k decryption operations, subsequently $K1$ can be found in roughly the same time. Document [300] also describes an alternative attack on RMAC ($m = 128$, $r = 128$) requiring 2^{123} chosen texts (in running time 2^{124}), and a serious attack on RMAC using three-key Triple-DES as underlying block cipher instead of AES (in certain cases this attack works with a complexity of about 2^{56} operations and success probability of 2^{-16}).

5.4.4 HMAC

The SHA-1 based HMAC [394] has been standardised by NIST as FIPS-198 and the general scheme is also included in the ISO/IEC standard 9797-2. Therefore it is considered as a benchmark for the submissions in this category. The scheme uses the SHA-1 hash function in a black box model and generates a MAC value of up to 160 bits. SHA-1 operates on blocks of 512 bits that are divided in 32-bit words, computing a 160-bit hash value.

5.4.4.1 The design

The computation of the MAC for a secret key K and a message X proceeds by the following steps (here h denotes hashing with the hash function SHA-1):

1. Compute a key value K' of 512 bits long. Suppose K has bitlength l. If $l = 512$ set $K' = K$; if $l < 512$ obtain K' by appending $512 - l$ zero bits to K; if $l > 512$ obtain K' by computing the hash $h(K)$ (160 bits long) and appending 352 zero bits to this hash value.
2. Exor K' with the 512-bit constant *ipad* and append the message X: $(K' \oplus ipad) \,\|\, X$.
3. Compute the hash of the string resulting from step 2: $h((K' \oplus ipad) \,\|\, X)$.
4. Exor K' with the 512-bit constant *opad* and append the 160-bit result from step 3: $(K' \oplus opad) \,\|\, h((K' \oplus ipad) \,\|\, X)$.
5. Compute the hash of the string resulting from step 4: $h((K' \oplus opad) \,\|\, h((K' \oplus ipad) \,\|\, X))$.
6. To obtain an m-bit MAC value, select the leftmost m bits of the result of step 5.

The string *ipad* is defined as the concatenation of 64 times the hexadecimal value '36', and the string *opad* is defined as the concatenation of 64 times the hexadecimal value '5c'.

5.4.4.2 Security analysis

Bellare *et al.* [36] give theoretical support for HMAC, relating the security of the MAC scheme to the security of the underlying hash function, in this case SHA-1 which is a well studied primitive for which no weaknesses have been reported. More specifically, it has been proved that HMAC is secure if the following assumptions hold (here f is the compression function that is iterated by the hash function for each 512-bit block):

- The hash function h is collision-resistant when the initial value is secret.
- The compression function f keyed by the initial value is a strong MAC algorithm (this means that its output is hard to predict).
- The values $f(K' \oplus ipad)$ and $f(K' \oplus opad)$ cannot be distinguished from truly random values. This means that the compression function f is a 'weak' pseudo-random function ('weak' because the opponent has no direct access to K').

It may be noted that if the HMAC construction is used with two independent keys (rather than using $K_1 = K' \oplus ipad$ and $K_2 = K' \oplus opad$), the level of security against key recovery attacks would be less than suggested by the algorithms key size (due to a divide-and-conquer attack). An internal collision based forgery for SHA-1-based HMAC needs about 2^{80} known text-MAC pairs and 1 chosen text when the output length is 160 bits. More chosen texts are required when the MAC output is truncated, e.g., when the leftmost 80 bits are chosen, the attack needs about 2^{80} known pairs and 2^{80} chosen texts. On the other hand this increases the probability of success for an attack where one tries to guess the MAC value.

5.5 Comparison of studied MAC primitives

In this section we compare the security levels of the MAC algorithms studied in the NESSIE project. No short-cut attacks have been found for any of the algorithms, except RMAC. The estimated complexity for an exhaustive key search depends on the bitlength k of the secret key: about 2^{k-1} off-line MAC computations are required. Likewise, the estimated complexity for an attack guessing the MAC output depends on the bitlength n of the output: about 2^{n-1} on-line MAC verifications are needed for a (non-verifiable) forgery. Note that for UMAC the (provable) forging probability is only near optimal: 2^{-60} for an output size of 64 bits. Table 5.1 below compares the possible values of k and n for the different algorithms[6].

The most effective generic attack on MAC algorithms is the birthday forgery attack (based on internal collisions): for an internal state size of l bits and output size of n bits, this attack requires about $2^{l/2}$ known text-MAC pairs and 2^{l-n} chosen texts. Table 5.2 below compares this complexity for the different algorithms. The entries in the table are denoted (α, β), where α is the number of known pairs and β the number of chosen texts that are needed. Maximum values

[6] Note that other parameter sets can be chosen for UMAC.

Table 5.1. Key length k and output length n for MAC primitives.

Algorithm	k	n
UMAC	128	64
TTMAC	160	≤ 160
EMAC-AES	128,192,256	≤ 128
RMAC-AES	128,192,256	≤ 128
HMAC-SHA-1	≤ 512	≤ 160

are chosen for the output size n of the algorithms (this minimises the number of chosen texts that are needed in the attack).

Table 5.2. Estimated (minimal) complexity of birthday forgery attacks on MACs.

Algorithm	birthday forgery
TTMAC	$(2^{160}, 2^{160})$
EMAC-AES	$(2^{64}, 1)$
RMAC-AES	$(2^{128}, 1)$
HMAC-SHA-1	$(2^{80}, 1)$

It can be seen from Table 5.2 that for EMAC-AES and HMAC-SHA-1 the birthday attack requires respectively 2^{64} or 2^{80} known pairs and (in both cases) 1 chosen text (when the output size n is equal to the internal state size). The randomised RMAC offers better resistance than EMAC: when both the output and the salt value are 128 bits long, the attack needs 2^{128} known pairs and 1 chosen text. TTMAC offers the highest level of security because of its large internal state size: 2^{160} known pairs and 2^{160} chosen texts are needed (when $n = 160$). Note that the birthday attack does not apply to the UMAC algorithm.

For RMAC-AES there is an alternative attack as described in [300]. This attack can be used to find one of the two keys in the system faster than by an exhaustive search (after which RMAC reduces to a simple CBC-MAC for which it is well known that simple forgeries can be found). The estimated complexity of the attack is 2^{123} chosen texts and 2^{124} running time (considering RMAC with output and salt value of 128 bits).

Changes from version 1.0 to version 2.0 of the document

—	Typos have been corrected.
§5.1	Introduction expanded and reference to Preneel added.
§5.2.1	Formal security model added.
§5.2.2	Divide-and-Conquer attack included. Exor Forgery attack included. Side-Channel attacks included.
§5.3	References to MAA removed (Preneel pointed out that it has been withdrawn from standards).
§5.3.1	Section on block cipher based designs rewritten.
§5.3.3	Section on universal hash based designs included.
§5.4.1	Notations for TTMAC description changed because of several overlaps.
§5.4.3	Name EMAC (aka DMAC) introduced. RMAC included and more security analysis.
§5.4.4	Security analysis of HMAC expanded.
§5.5	This section replaces the former 'Conclusion'. Contains a comparison of security levels of studied MACs (with tables).

6. Asymmetric encryption schemes

6.1 Introduction

Asymmetric encryption, also known as public-key encryption, is a method of sending messages securely between two people who do not share a common secret. This is the exact opposite of symmetric encryption, where the communicating parties are assumed to share a common secret key. Examples of symmetric encryption include block ciphers (see Sect. 2) and stream ciphers (see Sect. 3). It was developed out of the ideas of Diffie and Hellman [168] and was first properly realised as the ever popular RSA cryptosystem [452] in 1978.

In the twenty-five years since then, the area has received copious attention from researchers who have tightened security definitions and requirements, proposed and broken new schemes, and expanded the range of applications.

During the three-year lifespan of the NESSIE project, there have been several important shifts of focus within this research area. Much effort has been expended by the research community in the field of provable security and no new primitive is really taken seriously these days unless its security can in some way be related to a "hard" problem. This has also led to an increase in research on so called side-channel attacks: attacks that take advantage of information that may be available in the real world but is not available to an attacker in a mathematical model. These security requirements and limitations will be discussed in Sect. 6.2.

Also within the last three years, the International Organisation for Standardization (ISO) has developed a new framework for asymmetric encryption that attempts to better model the real-world use of public-key cryptography. The new KEM-DEM framework, discussed in Sect. 6.3, was popular enough that almost all the primitives selected for further study in phase II of the NESSIE project were tweaked to fit into this model. Only EPOC-2, discussed in Sect. 6.4.4, remained completely outside of this framework.

6.2 Security Requirements

6.2.1 Preliminaries

We start with a formal definition of an asymmetric encryption algorithm. In order to fulfil the security requirements of Sect. 6.2.2 we will see that it is necessary

[0] Coordinator for this chapter: RHUL — Alex Dent

for encryption algorithms to be probabilistic. We choose here to represent these probabilistic algorithms as deterministic algorithms that take some fixed length random seed as input. This serves to accentuate the problem of adaptively creating multiple random bit strings, e.g. generating one bit string, testing it for some property and then discarding it and generating a new random bit string. However, when there is no danger of confusion, we assume this input is implicit and whenever an algorithm generates some randomness it actually derives the required randomness from this seed.

Definition 6.1. *An asymmetric encryption system is a triple of deterministic algorithms* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$*. The first algorithm* $\mathcal{G}$ *is called the key generation algorithm and need only be run once to set up the system. It takes as input a security parameter* 1^λ *and a fixed length random seed* r*, and outputs a key-pair* (pk, sk)*. The key* pk *is called the public key and needs to be distributed to all people who wish to encrypt messages. The key* sk *is called the secret key or private key*[1] *and should only be known to those people who are permitted to read encrypted messages.*

The second algorithm $\mathcal{E}$ *is the encryption algorithm. It takes as input a message* m*, the public key* pk *and a fixed length random seed* r*, and outputs a ciphertext* C*.*

The last algorithm $\mathcal{D}$ *is the decryption algorithm. It takes as input a ciphertext and the secret key (which may include elements from the public key), and outputs a message* m' *or the error symbol* $\perp$*.*

For the system to be useful we require that it is *sound*, i.e. for any message m, random seed r and valid key-pair (pk, sk) we require that $\mathcal{D}(\mathcal{E}(m, r, pk), sk) = m$. We also require that some kind of security result holds that limits an attacker's power to recover information about a message m or the secret key sk from an encipherment.

It is impossible for an asymmetric encryption scheme to be perfectly secure; an attacker that has access to unlimited (time and computational) resources can always recover a secret key. This is because an attacker with unbounded resources can just search the (finite) space of possible private keys and check their ability to decrypt messages. So, in order to prove any meaningful results, we have to limit the attacker's computational power and there are two approaches to this problem.

The first uses the field of complexity theory. We can assume that the attacker is represented by a (probabilistic) Turing machine that runs in polynomial-time in the security parameter, and then derive results about its ability to break the scheme. This is a very elegant theory but is only useful as an asymptotic approximation. An attacker that runs in polynomial time is not guaranteed to be practical in real terms and an attacker that doesn't run in polynomial time is not guaranteed to be impractical for all useable security parameters. Since the

[1] Some standardisation bodies reserve the term "secret key" for a key used within a symmetric algorithm and insist upon the use of the term "private key" for asymmetric applications.

NESSIE call for primitives [396] required that the security level of a candidate be of a certain specified level we must regretfully put aside this theory and concentrate on something more concrete.

Definition 6.2. *A (t, ϵ) solver for a problem is a probabilistic Turing machine that runs in time bounded above by t and outputs a solution for the problem with probability at least ϵ.*

A (t, ϵ, q_D) attacker for an asymmetric encryption scheme is a probabilistic Turing machine $\mathcal{A}$ that runs in time bounded above by t, makes at most q_D queries to a decryption oracle and succeeds in breaking the scheme with probability at least ϵ.

A further discussion of the access an attacker might have to a decryption oracle and the definition of "breaking the scheme" can be found in Sect. 6.2.2.

Of course, the success probability ϵ depends upon the units we use for the time t — if we measure time in years then we would expect a higher success probability in 1 unit time than if we measure time in seconds! We decide that one unit of time is equal to the time taken for one decryption operation.

However there is still a problem with this approach. In order to prove that a system is as secure as the NESSIE call requires, the submitters had to prove the *absence* of a strong attacker. To do this the submitters were allowed to submit a proof that the existence of a (t, ϵ, q_D) attacker for their scheme implied the existence of a (t', ϵ') solver for some trusted cryptographic problem. The relationship between t, t', ϵ and ϵ' defines the *efficiency* of the security reduction. A discussion of those problems which are trusted to be "hard" by the cryptographic community can be found in Sect. 6.2.3.

6.2.2 The Security Models

So far we have specifically avoided stating what it means for an asymmetric encryption scheme to be secure. In order to do this we have to define two things: the conditions an attacker must fulfil for the scheme to be considered broken, and the access that an attacker has to the system.

There are many different ways in which a cryptosystem might be considered weak. Whether these weaknesses actually "break" the system, i.e. give an attacker some useful information, depends upon the application for which the cryptosystem is being used. We model these various success criteria as games that an attacker plays against a mythical system that controls the encryption scheme and measure the attacker's success as the probability that he wins the game.

The most obvious, and most naive, way in which an attacker can break an encryption scheme is if he can, given some ciphertext, recover the associated message.

Definition 6.3 (One-way (OW)). *Consider the following game that an attacker plays against a system, using an asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ with security parameter λ.*

1. *The system picks a random seed r, runs $\mathcal{G}(1^\lambda, r)$ to generate a key-pair (pk, sk) and passes the value pk to the attacker $\mathcal{A}$.*
2. *The attacker runs until it is ready to receive a challenge ciphertext.*
3. *The system picks a random seed r and a message m uniformly at random from the set of possible messages and calculates the challenge ciphertext $C = \mathcal{E}(m, r, pk)$. The system then passes C back to the attacker.*
4. *The attacker outputs a guess m' for the message m.*

The attacker wins the above game if $m' = m$.

An asymmetric encryption scheme is said to be one-way *or* OW *if the probability that the attacker wins the above game is small.*

This is a fairly weak definition. For example, if an encryption scheme is only used to encrypt a message from a certain small known subset of possible messages then it might be enough for an attacker to tell whether a ciphertext is the encryption of one given message or another. This leads to the stronger definition of message-indistinguishable encryption schemes [227, 446].

Definition 6.4 (Message-indistinguishable (IND)). *Consider the following game that an attacker plays against a system, using an asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ with security parameter λ.*

1. *The system picks a random seed r, runs $\mathcal{G}(1^\lambda, r)$ to generate a key-pair (pk, sk) and passes the value pk to the attacker $\mathcal{A}$.*
2. *The attacker generates two distinct messages m_0 and m_1, and submits them to the system.*
3. *The system*
   a) *Chooses a bit σ uniformly at random from $\{0, 1\}$ and a random seed r.*
   b) *Calculates the challenge ciphertext $C = \mathcal{E}(m_\sigma, r, pk)$ and returns this to the attacker.*
4. *The attacker outputs a guess σ' for σ.*

The attacker wins the above game if $\sigma' = \sigma$.

An attacker $\mathcal{A}$ has an advantage *$Adv_\mathcal{A}$ of winning the above game where*

$$Adv_\mathcal{A} = Pr[\sigma' = \sigma] - 1/2 \tag{6.1}$$

and the scheme is said to have advantage

$$Adv = \max_\mathcal{A} Adv_\mathcal{A} \ . \tag{6.2}$$

An asymmetric encryption scheme is said to be message-indistinguishable *or* IND *if the scheme's advantage is small.*

Consequently it is easy to see that if a system is message-indistinguishable then it is certainly one-way; however there are schemes which are thought to be one-way that are definitely not message-indistinguishable (such as the original RSA cryptosystem [452]). Of course it is difficult to show that a scheme is one-way but not message indistinguishable as a proof that a scheme was one-way would

imply a proof that $P \neq NP$, which would be a major mathematical achievement. However we can show that there exists the scope for a scheme that is one-way but not message indistinguishable. Consider an asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}', \mathcal{D}')$ constructed from a one-way asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ in the following manner:

$$\begin{aligned} \mathcal{E}'(m, pk) &= b||\mathcal{E}(m, pk) \text{ where } b \text{ is the leftmost bit of } m\,, \\ \mathcal{D}'(b||C, sk) &= \begin{cases} \mathcal{D}(C, sk) & \text{provided the leftmost bit of } \mathcal{D}(C, sk) \text{ is } b\,, \\ \perp & \text{otherwise}\,. \end{cases} \end{aligned}$$

This scheme is one-way (possibly slightly less one-way than the original scheme) and yet it is easy to distinguish between messages that have different leftmost bits. Hence the scheme is not message indistinguishable.

Next we will consider the access an attacker has to the system. In its purest form an attacker might only have access to the challenge ciphertext and the public key. However it is possible that an attacker might be able to gain decryptions of certain messages, so we have to define more relaxed attack models.

Definition 6.5 (Attack models). *We assume that the attack algorithm $\mathcal{A}$ runs in two stages: pre-challenge and post-challenge. Let the attacker have access to an oracle $\mathcal{O}_1$ up until the challenge is issued, and access to the oracle $\mathcal{O}_2$ after this time.*

- *The attack is said to be a* chosen plaintext attack (CPA) *if the oracles are both trivial, i.e. $\mathcal{O}_1 = \mathcal{O}_2$ and both return the error symbol $\perp$ for any input.*
- *The attack is said to be a* chosen ciphertext attack (CCA1) *or* lunchtime attack *if the oracle $\mathcal{O}_1$ decrypts messages (so $\mathcal{O}_1(C) = \mathcal{D}(C, sk)$) but the oracle $\mathcal{O}_2$ is trivial.*
- *The attack is said to be an* adaptive chosen ciphertext attack (CCA2) *if both oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ decrypt messages, with the exception that the oracle $\mathcal{O}_2$ returns $\perp$ if it is queried on the challenge ciphertext.*

When the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ are not trivial, they are referred to as decryption oracles.

Again it is easy to see that if a system is secure against an attack in the CCA2 attack model then it is also resistant to that attack in the CCA1 and CPA models.

There has been much discussion about the appropriateness of each of the different types of attack model. It is relatively easy to envision a situation where it is necessary for an encryption scheme to be resistant to attacks in the CCA1 model – for example, a malicious employee who only attempts to attack a system after he has been fired and therefore had his decryption privileges revoked. It is a lot harder to envision a situation where the CCA2 model is appropriate. It is difficult to see why an attacker with such strong decryption access would not be able to just decrypt the challenge ciphertext. However this is not the best way to think of the model. It is better to think of the CCA2 model as proving that the

only way to break the system is to apply the decryption function to the challenge ciphertext.

The combination of success criteria and attack model are usually referred to by their abbreviations. For example, a scheme that is message-indistinguishable in the adaptive chosen ciphertext model is said to be IND-CCA2. This is the strongest measure of security and is the *de facto* standard for an asymmetric encryption scheme. Formally we will use the notion of an attacker derived in Sect. 6.2.1. A (t, ϵ, q_D) attacker in the IND-CCA2 model is a probabilistic Turing machine $\mathcal{A}$ that runs in time at most t, makes at most q_D queries to the decryption oracles (i.e. the total number of queries made to the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ is at most q_D), and distinguishes messages with advantage at least ϵ.

We will consider the security of the submitted schemes primarily in the IND-CCA2 model, i.e. we will consider the resistance to message-distinguishing attacks in the adaptive chosen ciphertext model. However, we will also consider the security of some of the submitted schemes in the IND-CPA model, i.e. in a model where the attacker does not have access to a decryption oracle. Obviously, any security reduction valid in the IND-CCA2 model is also valid in the IND-CPA model. Therefore the IND-CPA model is only of interest if it yields a tighter security proof or a security proof that reduces to a better underlying problem. In the IND-CPA model we will talk about a (t, ϵ) attacker: a probabilistic Turing machine $\mathcal{A}$ that runs in time at most t and distinguishes messages with advantage at least ϵ.

For a further discussion of security models the reader is referred to [37, 351].

6.2.3 Trusted cryptographic problems

As we have already mentioned, it is impossible to prove the security of an asymmetric encryption scheme directly. The best that can be done is to relate the security of a scheme back to some problem that is thought to be hard by the research community and trusted by developers. In this section we will discuss the various trusted problems that the security of the submitted primitives can be reduced to.

The NESSIE primitives use two distinct 'flavours' of trusted problems: those based on the difficulty of solving the discrete logarithm problem and its related problems, and those based on the difficulty of factoring composite numbers. For a further discussion of the trusted cryptographic problems used by the NESSIE primitives see [351].

6.2.3.1 Factoring based problems

All of the schemes based on factoring problems use arithmetic in the equivalence class $\mathbb{Z}/n\mathbb{Z}$ where n is some composite number. In all cases the ability to factor the number (or 'modulus') n implies the ability to solve the hard problem. Note that, for simplicity, we define $\lambda(n) = l.c.m.(p-1, q-1)$ when $n = pq$.

Definition 6.6. *Let n be a composite number.*

- *The factoring problem is the problem of finding the component factors of a composite integer n. In particular, we will concentrate on finding the component factors of an integer n of the form $p^d q$, where p and q are prime and $d \geq 1$.*
- *An RSA key is a pair (n, e) where $n = pq$ with p and q primes, and $1 < e < n$ with g.c.d.$(\lambda(n), e) = 1$. The RSA problem is the problem of finding an integer $1 \leq x \leq n$ such that $x^e = y \bmod n$ when given an RSA key (n, e) and a randomly selected integer $1 \leq y \leq n$.*

The factoring problem has been of interest to mathematicians for centuries and there are many algorithms that efficiently solve it for specific classes of 'modulus' n. However there are no known efficient algorithms for solving the factoring problem on a modulus $n = p^d q$ for small d (i.e. for $d < \sqrt{\log p}$). Again it is quite easy to see that if the RSA problem is hard for a modulus n then the factoring problem is hard for n too. The RSA problem was implicitly defined when the original RSA cryptosystem was proposed and has been widely studied.

Whilst all of the above problems are well established, some of the schemes studied in phase I reduce to one of the following less well known problems.

Definition 6.7. *The following are trusted cryptographic problems:*

- *Consider a modulus of the form $n = p^2 q$ where the factorisation is unknown. The p-subgroup problem is the problem of deciding, given $h \in (\mathbb{Z}/n\mathbb{Z})^*$, if there exists a $g \in (\mathbb{Z}/n\mathbb{Z})^*$ such that $g^p = h$.*
- *For our purposes, the gap-factoring problem is the problem of finding the component factors of a composite integer $n = p^2 q$ when given access to a specific oracle. The oracle returns the bit b when queried with $g^b h^r$ for unknown $b \in \{0, 1\}$ and $r \in \{0, \ldots, p-1\}$, and fixed public $g, h \in (\mathbb{Z}/n\mathbb{Z})^*$ such that*
  - *g^p has order $p - 1$ modulo p^2,*
  - *$h = h_0^n$ for some h_0.*

These problems have not been subject to the same level of peer review by the scientific community as the factoring problem or the RSA problem, however there is no obvious reason to suspect that the p-subgroup problem or the gap factoring problem is particularly easier to solve than the RSA or factoring problems.

6.2.3.2 Discrete logarithm based problems

Problems related to the discrete logarithm problem are usually phrased in terms of a group $G = \langle g \rangle$ where g has prime order q. Technically, since all groups of prime order are isomorphic, the hardness of the problem depends not upon the group itself but upon the representation of the group. For example, the discrete logarithm problem is thought to be hard in elliptic curve subgroups but is definitely easy in the group of integers modulo q.

Definition 6.8. *Let g be an element of a group, and let g have order q.*

- *The discrete logarithm problem (DLP) is the problem of finding a when given (g, g^a).*
- *The computational Diffie-Hellman problem (CDH) is the problem of finding g^{ab} when given (g, g^a, g^b).*

– *The decisional Diffie-Hellman problem (DDH) is the problem of deciding if* $g^{ab} = g^c$ *when given* (g, g^a, g^b, g^c).
– *The gap Diffie-Hellman problem (Gap-DH) is the problem of finding* g^{ab} *when given* (g, g^a, g^b) *and an oracle* $\mathcal{O}$ *that correctly solves the decisional Diffie-Hellman problem.*

Obviously if the DDH problem is hard in the group $\langle g \rangle$ then so is the CDH problem and the DLP. All of these problems are well established, except for the Gap-DH problem, which was only formally introduced in 2001 [418].

In situations where the security of an algorithm can be reduced to the difficulty of solving the CDH problem on some group it is not uncommon for the algorithm constructed to solve the CDH problem to output not a single answer to the problem but a list of some L elements that contains the answer. Obviously, if one could solve the DDH problem on that group (either by means of a dedicated algorithm or by the use of some oracle) then selecting the correct answer from the list would be trivial, but we cannot assume that the DDH problem is tractable on the group so we are forced to use other means to find the correct answer.

The simplest, and fastest, method of selecting an answer is to pick an element of the list uniformly at random and output that element as the algorithm's final answer. The probability that this technique outputs the correct answer is $1/L$ times the probability that the correct answer appears on the list. Since L is typically of order related to q_D, the number of decryption oracle queries, this can have a significant impact on the efficiency of the reduction (see Sect. 6.2.7).

A more sophisticated approach has been suggested by Shoup [486]. Suppose that there exists an algorithm that, given an instance of the CDH problem (g, g^a, g^b), outputs a (small) list of L elements which contains the solution to the CDH problem with probability at least ϵ. Then we may construct an algorithm that will output the correct answer to the CDH problem with probability at least $1 - (1/2)^k$ for some $k > 2$. However, this involves running the original algorithm $2k\lceil 1/\epsilon \rceil$ times with randomised inputs.

Formally, if there exists an algorithm that outputs a list of L elements which contains the solution to the CDH problem with probability at least ϵ, and this algorithm runs in time bounded by t, then there exists a (t', ϵ') solver for the CDH problem with

$$\epsilon' \approx 1 - \frac{1}{2^k}\,, \tag{6.3}$$

$$t' \approx 2k\lceil 1/\epsilon \rceil t + 2kL\lceil 1/\epsilon \rceil T\,, \tag{6.4}$$

where

– k is an integer greater than two,
– and T is the time taken to check an equation of the form

$$\left\{h \cdot g^{-ax_1y_2} \cdot g^{-bx_2y_1} \cdot g^{x_2y_2}\right\}^{x'_1y'_1} = \left\{h' \cdot g^{-ax'_1y'_2} \cdot g^{-bx'_2y'_1} \cdot g^{x'_2y'_2}\right\}^{x_1y_1}\,.$$

for random $x_1, y_1, x_2, y_2, x'_1, y'_1, x'_2, y'_2$ and fixed g^a and g^b.

6.2.3.3 Solving trusted cryptographic problems

As has been mentioned several times, even the trusted cryptographic problems in the previous two sections can be solved given enough computing power. Therefore, if we estimate the different amounts of computing power it would take to solve these problems then we may gain some measure of the comparative difficulty of seemingly unrelated problems. The notation $L_q[\alpha, c] = O(\exp((c+o(1))(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha}))$ is used for estimating asymptotic complexity.

- **Integer factorisation.** The fastest known algorithms for factorising large integers are the Number Field Sieve [331] and the Elliptic Curve Method [335]. The asymptotic time taken by the number field sieve to factor an integer n is approximately $L_n[\frac{1}{3}, c]$, where c depends on the variant of the number field sieve. The asymptotic time taken by the elliptic curve method to factor an integer whose smallest factor is p is $L_p[\frac{1}{2}, \sqrt{2}]$. Both of these algorithms are subexponential in the size of their input.
  An improvement of the elliptic curve method exists for $n = p^2q$ [430, 179] and a special algorithm exists for $n = p^rq$ with large r [97].
- **RSA problem** The fastest method known for solving the general RSA problem involves factoring the modulus.
- **Discrete logarithm over $\mathbb{F}_p$.** The index-calculus method [122, 411, 186] is the fastest known method of solving the discrete logarithm problem over $\mathbb{F}_p$. It is closely related to the number field sieve factoring algorithm and has expected asymptotic running time of $L_p[\frac{1}{3}, c]$, which is subexponential in the input size.
- **Elliptic curve discrete logarithm.** The fastest general method of attack for solving the elliptic curve discrete logarithm problem are the Pollard ρ and the Pollard λ methods [436]. For a group with q elements, the Pollard ρ runs in time $\sqrt{\pi q/2}$ and the Pollard λ runs in time $2\sqrt{q}$ but can be faster in some special cases. Both can be efficiently parallelised [505] and have been slightly improved [212, 522]. No subexponential algorithm has been found for solving the elliptic curve discrete logarithm problem.
  However, there are subexponential attacks for specific elliptic curves: supersingular curves [364, 200, 456] and anomalous curves [470, 461, 492].

NESSIE has concluded that the following key sizes are roughly equivalent. For a further discussion on this subject, the reader is referred to Sect. 7.2.2.3.

Equivalent symmetric key size	56	64	80	112	128	160
Elliptic curve size	112	128	160	224	256	320
Modulus length (pq)	512	768	1536	4096	6000	10000
Modulus length (p^2q)	570	800	1536	4096	6000	10000

It should be noted that these are estimates for classical computers. If it proves possible to build a quantum computer then there exists quantum algorithms that successfully solve both the discrete logarithm problem and the factorisation problem [485]. None of the asymmetric encryption algorithms submitted to NESSIE should be considered secure against attacks made by quantum computers.

6.2.4 The Random Oracle Model

Owing to the complex nature of asymmetric encryption schemes, it is very difficult to prove results about their security without making some assumptions about the properties of the components that make up the cipher. A security proof that does not make any assumptions is called a proof in the *standard model*.

The most common assumption used to simplify a proof is that a good hash function will behave exactly like a completely random function. This is the *random oracle model*, and was introduced by Bellare and Rogaway in 1993 [44]. Random functions (or oracles) and good hash functions share many properties, for example in both cases it is difficult to compute the pre-image of a given output or to find two elements that have the same image, and so this might be considered a reasonable modelling assumption. One of the common interpretations of a proof in the random oracle model used to be that if a scheme had a security proof in the random oracle model then that scheme was secure unless the particular hash function used interacted badly with the rest of the cryptosystem. This was considered unlikely to happen as hash functions are usually composed on the bit level whilst asymmetric encryption schemes take advantage of higher-level properties such as group structures.

Doubt, however, was cast on the random oracle model in a paper by Canetti, Goldreich and Halevi [114]. This paper proved that if there exists a cryptosystem that is secure in the random oracle model then there exists a cryptosystem that is secure in the random oracle model but insecure when the random oracle is replaced with *any* hash function. Whilst this means that the above interpretation is, in fact, incorrect, many people still accept it owing to the highly technical and theoretical nature of the results in [114].

For the purposes of NESSIE, proofs of security that were given in the random oracle model were accepted but regarded as heuristic.

6.2.5 Other models

There are several other models that have been used to examine cryptographic algorithms, but all of the NESSIE submissions were provided with proofs of security in either the standard or random oracle model. The only other model that could be of interest is the *generic group model* [395, 486].

The generic group model examines the security of schemes that can be implemented on many different groups, such as ECIES (see Sect. 6.4.2). A proof of security in the generic group model intends to show that a scheme is secure up to attacks that take advantage of the specific nature of the group on which the scheme operates. It models this by only giving the scheme access to a random encoding of a group element, rather than the group element itself.

Unfortunately the generic group model was shown to have the same weaknesses as the random oracle model: if there exists a scheme that is provably secure in the generic group model then there exists a scheme that is provably secure in the generic group model but insecure when the random encoding function is replaced by *any* fixed encoding function [161, 194]. Also the generic security proofs

cannot be provided for the schemes based on factoring problems, as the factoring problems tend to be defined on specific group encodings (usually $\mathbb{Z}/n\mathbb{Z}$). This means that NESSIE has given less weight to security proofs given in the generic group model.

6.2.6 Side-channel attacks

Due to the increasing prominence of proofs of security in the field of asymmetric encryption, there has been an increase in interest in side-channel attacks. Essentially, a side-channel attack is the only method of breaking a provably secure asymmetric encryption scheme without breaking the underlying problem. For further information on side-channel attacks, the reader is referred to Sect. A. For a further discussion of side-channel attacks against the asymmetric encryption primitives submitted to NESSIE, the reader is referred to [164, 173, 424].

6.2.7 Assessment criteria

The most important criterion in the NESSIE evaluation process is security. The submitters were encouraged to submit their asymmetric encryption primitives with a proof that they are secure in the IND-CCA2 model, although this proof could use the heuristic random oracle model. The security proofs were evaluated in terms of the hardness of the problem to which the security of the scheme reduces (see Sect. 6.2.3) and the efficiency of that reduction.

The efficiency of the reduction is the relationship between a (t, ϵ, q_D) attacker that breaks the cryptosystem and the implied (t', ϵ') solver that would solve the underlying trusted cryptographic problem. We can then examine the best known methods for solving the underlying problem and estimate the best advantage ϵ that an attacker could be said to have. Normally we will also be able to determine the minimum size of the security parameter for which the cryptosystem has the security level specified in the NESSIE call. We classify the efficiency of the security reductions in the following way:

- the security reduction is tight if $\frac{t'}{\epsilon'} \approx \frac{t}{\epsilon}$,
- the security reduction is not so tight if $\frac{t'}{\epsilon'} \approx q_D \frac{t}{\epsilon}$,
- the security reduction is loose if $\frac{t'}{\epsilon'} \gg \frac{t}{\epsilon}$.

Usually we assume that $q_D \ll 2^\lambda$, where λ is the security parameter.

In order for a scheme to be practical we must balance its security against the performance (speed and size) of the protocol. In order for these calculations to be fair and accurate we need to make certain comparisons between the various parameter sizes of the underlying trusted problems. We wish to use parameters that ensure that each of the problems can be solved with the same amount of computational power and that this amount of computational power sufficiently protects the data. A discussion of these issues can be found in [479].

In accordance with the wishes of the NESSIE Project Industry Board, Intellectual Property Right (IPR) issues were also considered.

Lastly we use vulnerability to side-channel attacks only as a final factor in determining suitability. This is because an algorithm that is vulnerable to a side-channel attack can be protected by a careful implementation, indeed this is the most common solution according to the PIB. The attitude of the PIB to side-channel attacks is typified by comments such as

> Only if it can be shown that the side channel attack applies to a primitive, regardless of the implementation (i.e. there is no known defence, or the primitive has an inherent weakness when confronted with side channel attacks), should it be taken into account as a selection criteria.
>
> – Pieter Kasselman, Baltimore Technology

6.3 KEM-DEM cryptosystems

6.3.1 Hybrid encryption

In comparison with symmetric encryption algorithms, asymmetric encryption schemes are often slow and tend to have smaller message spaces. Consequently, in practice, asymmetric encryption schemes are often only used to encrypt a randomly generated symmetric key that is then used to encrypt a longer message. Asymmetric encryption schemes that use this technique are known as hybrid encryption schemes. The KEM-DEM model is a formalisation of this idea. It was introduced in a later version of [142], which also included the security analysis of ACE-KEM described in Sect. 6.4.1, and in the draft ISO proposal [489].

A KEM-DEM cryptosystem is composed of two algorithms: a *key encapsulation mechanism (KEM)* and a *data encapsulation mechanism (DEM)*. A key encapsulation mechanism is a scheme that, given a public-key, derives a random key and provides a method of encrypting (encapsulating) and decrypting (decapsulating) that random key. This typically uses asymmetric techniques. The data encapsulation mechanism uses that random key to encrypt a message. This typically uses symmetric techniques. Formally,

Definition 6.9. *A* key encapsulation mechanism (KEM) *is a triple of deterministic algorithms* $(\mathcal{G}, KEM.Encrypt, KEM.Decrypt)$*. As before,* $\mathcal{G}$ *is a key generation algorithm that takes a security parameter* 1^λ *and a random seed as input and produces a key-pair* (pk, sk)*. The encapsulation algorithm,* $KEM.Encrypt$*, takes the public-key* pk *and a random seed as input and outputs a pair* (K, ψ)*. The decapsulation algorithm,* $KEM.Decrypt$*, takes as input an encapsulation* ψ *and the secret-key* sk*, and outputs a key value* K' *or the error symbol* $\perp$*.*

As before we require that the KEM is sound, i.e. for any valid key-pair (pk, sk) the decapsulation of an encapsulated key is the key itself or in other words, if $KEM.Encrypt(pk, r) = (K, \psi)$ then $KEM.Decrypt(\psi, sk) = K$. We will also need some kind of security result that limits an attacker's ability to derive information about the key K from the public-key pk and the encapsulation ψ. Notice that the KEM encapsulation algorithm does only take as input a random seed and

the public-key pk, which is often considered to be a system parameter, and never has access to the message.

Again, although we define both the key generation algorithm and the encapsulation algorithm as deterministic, it is often easier to think of them as probabilistic algorithms. So, when the context is sufficiently clear, we will implicitly assume the presence of a random input and that any randomness required by an algorithm is actually derived from the random input provided.

Definition 6.10. *A* data encapsulation mechanism (DEM) *is a pair of deterministic algorithms* $(DEM.Encrypt, DEM.Decrypt)$.[2] *The encryption algorithm* $DEM.Encrypt$ *takes as input a message* m *and a key* K *and computes a ciphertext* χ. *The decryption algorithm* $DEM.Decrypt$ *takes as input a ciphertext* χ *and a key* K *and outputs a message* m' *or the error symbol* $\perp$.

The DEM must also be sound, i.e. for all valid keys K and messages m we have that $DEM.Decrypt(DEM.Encrypt(m, K), K) = m$, and it must satisfy some security condition. It should also be noted here that the DEM does not have access to the public key used by the KEM, only the symmetric key produced by the KEM and the message itself.

Definition 6.11. *A KEM-DEM based cryptosystem is a hybrid asymmetric encryption scheme composed of a KEM* $(\mathcal{G}, KEM.Encrypt, KEM.Decrypt)$ *and a DEM* $(DEM.Encrypt, DEM.Decrypt)$ *where the output key-space of the KEM is the same as the key-space of the DEM. To encrypt a message* m *the hybrid scheme runs as follows:*

1. *Generate a random seed* r.
2. *Run* $KEM.Encrypt(pk, r)$ *to produce an encapsulation pair* (K, ψ).
3. *Run* $DEM.Encrypt(m, K)$ *to produce a ciphertext* χ.
4. *Output* $C = (\psi, \chi)$.

Hence the decryption algorithm for a ciphertext C *is*

1. *Parse* C *as appropriately sized* ψ *and* χ.
2. *Run* $KEM.Decrypt(\psi, sk)$ *to obtain a key* K' *or* $\perp$.
3. *Run* $DEM.Decrypt(\chi, K')$ *to obtain a message* m' *or* $\perp$.
4. *Output* m' *or* $\perp$.

Key generation for the hybrid scheme is provided by $\mathcal{G}$.

At this point it might be helpful to consider an elementary example of a KEM-DEM based cryptosystem. The first asymmetric encryption scheme that can be modelled as a KEM-DEM cryptosystem is the ElGamal scheme [185]. This uses

[2] Technically there is no reason why the DEM should be composed of two probabilistic algorithms. However, due to the computationally problems associated with generating random or pseudo-random bits, we do not recommend that probabilistic algorithms are used.

a KEM based on the Diffie-Hellman key agreement protocol and a DEM based on modular multiplication.[3]

This provides a good example of the differences between a key encapsulation mechanism and a key agreement or key exchange protocol. Although a key agreement protocol can be used as the basis for a KEM, the security requirements for a KEM are different from those of a key agreement protocol. A KEM is only required to produce an encapsulation of a key that will be secure *when used once*. The ElGamal scheme is a good example: the scheme is provably as secure as the decisional Diffie-Hellman problem (see Sect. 6.2.3) when a new key is generated for each encryption, but it is totally insecure when keys are reused.

6.3.2 KEM Security

One of the aims of the KEM-DEM model was to provide a security analysis based on the component parts. The security of the KEM is based on the inability of an attacker to distinguish a proper encapsulation pair from a random pair.

Definition 6.12. *Consider the following game an attacker $\mathcal{A}$ plays against a system using a KEM $(\mathcal{G}, KEM.Encrypt, KEM.Decrypt)$ with a security parameter λ.*

1. *The system runs $\mathcal{G}(1^\lambda, r)$ (for some suitably random seed r) to generate a random key-pair (pk, sk) and passes pk to the attacker.*
2. *The attacker runs until it requests a challenge encapsulation.*
3. *The system generates the challenge in the following way:*
   a) *The system generates a suitably random seed r.*
   b) *Next, it runs $KEM.Encrypt(pk, r)$ to generate a pair (K_0, ψ).*
   c) *The system then generates a key K_1 uniformly at random from the entire output space of the KEM.*
   d) *Lastly, it picks a bit σ uniformly at random from $\{0, 1\}$ and returns (K_σ, ψ) to the attacker.*
4. *The attacker outputs a guess σ' for σ.*

The attacker wins the above game if $\sigma' = \sigma$.

The advantage of an attacker $\mathcal{A}$ is

$$Adv_{\mathcal{A}} = Pr[\sigma' = \sigma] - 1/2 \tag{6.5}$$

and the advantage of the KEM is said to be

$$AdvKEM = \max_{\mathcal{A}} Adv_{\mathcal{A}} \ . \tag{6.6}$$

A KEM is said to be indistinguishable *or* IND *if its advantage is small.*

A (t, ϵ) attacker for a KEM in the IND-CPA model is a probabilistic Turing machine $\mathcal{A}$ that runs in time bounded above by t and has advantage at least ϵ. A

[3] Technically, ElGamal does not fit into the formal KEM-DEM model as the DEM requires access to the public key in order to encrypt the message. However the structures are sufficiently similar to be enlightening.

(t, ϵ, q_D) attacker for a KEM in the IND-CCA2 model is a probabilistic Turing machine $\mathcal{A}$ that runs in time bounded above by t, makes at most q_D decryption queries and has advantage at least ϵ.

Most key encapsulation mechanisms are composed of some kind of security mechanism which is highly algebraic and some kind of key derivation function (KDF). The purpose of the key derivation function is more than just formatting: it takes some raw key or seed produced by the security mechanism and produces an appropriately sized bit string that has been stripped of all its algebraic properties. This usually involves some kind of hash function and is often modelled as a random oracle. A security mechanism is defined as follows.

Definition 6.13. *A security mechanism is a pair (Mech.Encrypt,Mech.Decrypt) of algorithms along with some key generation algorithm $\mathcal{G}$. The encryption function Mech.Encrypt takes as input a random seed r and the public-key pk, and outputs a pair (K_{raw}, ψ). The decryption function Mech.Decrypt inverts this operation by returning K_{raw} when given ψ and the secret key sk.*

This 'pared down' version of the KEM allows us to show that, in the random oracle model at least, key encapsulation mechanisms are fairly abundant. The following is shown in [160].

Theorem 6.1. *If (Mech.Encrypt, Mech.Decrypt) is a security mechanism that is OW-CCA2 and KDF is a key derivation function, then we may define a KEM $(\mathcal{G}, KEM.Encrypt, KEM.Decrypt)$ in the following manner. Key generation is provided by the key generation algorithm of the security mechanism. We define the encapsulation function $KEM.Encrypt(pk, r)$ as follows:*

1. *Compute $Mech.Encrypt(pk, r) = (K_{raw}, \psi)$.*
2. *Compute $K = KDF(K_{raw})$.*
3. *Output (K, ψ).*

The corresponding decryption function $KEM.Decrypt(\psi, sk)$ can then be defined as follows:

1. *Compute $Mech.Decrypt(\psi, sk) = K_{raw}$.*
2. *Compute $K = KDF(K_{raw})$.*
3. *Output K.*

If the security mechanism is OW-CCA2 (in the obvious sense, i.e. that an attacker is unable to recover K_{raw} from ψ) then, in the random oracle model, the KEM is IND-CCA2.

As will be seen, both the RSA protocol and the Diffie-Hellman key agreement protocol make good security mechanisms and this construction will be used in RSA-KEM (see Sect. 6.4.6) and ECIES-KEM (see Sect. 6.4.3). Further generic constructions for KEMs from low-level primitives can be found in [162].

There is also a further distinction that one can make with regard to key encapsulation mechanisms. Of the KEMs presented there seem to be two separate flavours: *authenticated* key encapsulation mechanisms and *unauthenticated* key

encapsulation mechanisms. An authenticated KEM is a KEM where a decapsulated key is only released if the encapsulation satisfies some kind of extra criteria that gives the user some assurance that the encapsulation was properly constructed and not just some kind of guess. A good example of this is the difference between ECIES-KEM (see Sect. 6.4.3) and ACE-KEM (see Sect. 6.4.1). ACE-KEM uses a method similar to ECIES-KEM for the actual key encapsulation but adds extra elements to authenticate the fact that a key was encapsulated properly. The difference is typified by the fact that an unauthenticated KEM will generally decapsulate any encapsulated key whilst an authenticated KEM will reject most randomly formed encapsulation. Although authenticated KEMs usually have better security proofs, i.e. security proofs that work in stronger models or reduce more efficiently to weaker assumptions, some concern has been raised as to whether it is necessary to authenticate a key that will be used to decrypt a message that will, most likely, itself be authenticated [160].

6.3.3 Key derivation functions

Almost all of the asymmetric encryption primitives use some kind of key derivation function (KDF) or mask generating function (MGF). Whilst the uses of key derivation functions and mask generating functions are slightly different, the properties that each of the functions must have and the methods used to construct these functions seem to be the same. Hence we will only refer to key derivation functions in this section with the understanding that all of the following discussion is relevant to mask generating functions as well.

Key derivation functions are similar to hash functions in that they map bit strings of any length to fixed length bit strings in an almost random way. However, unlike hash functions, which map a bit string to a bit string of a fixed length determined by the hash function, KDFs and MGFs are families of functions that map a bit string onto a bit string of a fixed length which may depend upon the security parameter or the public-key. KDFs are usually based on hash functions (see Chapter 4) and are often modelled as random oracles (see Sect. 6.2.4) for simplicity.

Whether we regard KDFs (and MGFs) as distinct cryptographic entities or modes of operation of a hash function, they are still outside the initial scope of NESSIE and so have not in themselves received a high level of scrutiny from the NESSIE partners.

With the exception of ACE-KEM (see Sect. 6.4.1), each of the primitives that used a KDF (or MGF) modelled it or them as a random oracle. Hence for the security proof to be valid it is necessary for each of the functions to act in a manner that has no obvious consistencies and are independent of each other and of any hash functions used. For ACE-KEM the KDF is required to have an output that is indistinguishable from random even when some of the leftmost bits of the input are known.

All of the key derivation functions submitted to NESSIE use one of two techniques to construct a KDF $KDF(\cdot)$ from a hash function $Hash(\cdot)$. Suppose

$Hash(\cdot)$ is a hash function with output size $HashLen$ and that we wish to evaluate $KDF(\cdot)$ on an input x and get an output of length $KDFLen(\lambda, pk)$. Set

$$BlockNum(\lambda, pk) := \left\lceil \frac{KDFLen(\lambda, pk)}{HashLen} \right\rceil .$$

The first technique, known as *KDF1* [489] or *MGF1* [455], is to use the leftmost $KDFLen(\lambda, pk)$ bits of the string

$$Hash(x||0_{32}) \,||\, \ldots \,||\, Hash(x||(BlockNum - 1)_{32}) ,$$

where i_{32} is the 32-bit representation of the integer i. The second technique, known as $KDF2$ [489], is to use the leftmost $KDFLen(\lambda, pk)$ bits of the string

$$Hash(x||1_{32}) \,||\, \ldots \,||\, Hash(x||BlockNum_{32}) ,$$

where i_{32} is the 32-bit representation of the integer i. Notice that both of these techniques fail if the output size of the KDF is required to be too large, i.e. $KDFLen(\lambda, pk) > 2^{32} \cdot HashLen$.

These techniques have the advantage that the output of the KDF is random providing the underlying hash function is random (and unavailable to the attacker). However these functions have been criticised by Shoup [489] because of the nature of the way some hash functions work. Whilst this criticism appears valid, it does not appear to be a critical flaw in the design.

Another problem with modelling both hash functions and key derivation functions as random oracles is that the outputs of each of these functions need to be independent of each other. Obviously if the KDF is either *KDF1* or *KDF2* then the output will be correlated to the output of the hash function $Hash(\cdot)$. The simplest solution would be for each primitive to use a different hash function, however this is often impractical. Another good solution is to assign each component that uses the hash function, including the hash function itself, a unique fixed length identifier *id* and prefix all inputs to the hash function with this value. In this case, the outputs of the hash function and the key derivation functions *KDF1* and *KDF2* will be random and uncorrelated.

6.3.4 DEM Security

It might be reasoned from some of the discussion in this chapter that the security of the KEM is in some way more important than the security of the DEM. This is, of course, not true. As we shall see in Sect. 6.3.5, the security of the hybrid scheme depends in equal measure on the security of the KEM and the DEM.

We have seen that the key produced by a KEM is very similar to a random key, hence it makes sense to examine the security of the DEM under the action of a random key. Furthermore, since it makes no sense to query a DEM decryption oracle before a challenge key has been produced, we will have to slightly tweak the attack models for the DEM.

Definition 6.14. *Consider the following game an attacker plays against a system, using a DEM* $(DEM.Encrypt, DEM.Decrypt)$ *with a security parameter* λ.

1. *The system generates a key K for the DEM uniformly at random from the key-space of the DEM (which is defined in terms of the security parameter λ).*
2. *The attacker generates two distinct messages m_0 and m_1 of the same length, and submits these to the system.*
3. *The system*
   a) *Chooses a bit σ uniformly at random from $\{0,1\}$.*
   b) *Calculates the challenge ciphertext $\chi = DEM.Encrypt(m_\sigma, K)$ and returns this to the attacker.*
4. *The attacker outputs a guess σ' for σ.*

The attacker wins the game if $\sigma' = \sigma$.

An attacker $\mathcal{A}$ has an advantage *$Adv_{\mathcal{A}}$ of winning the above game where*

$$Adv_{\mathcal{A}} = Pr[\sigma' = \sigma] - 1/2 \tag{6.7}$$

and the DEM is said to have advantage

$$AdvDEM = \max_{\mathcal{A}} Adv_{\mathcal{A}} \ . \tag{6.8}$$

A DEM is said to be message-indistinguishable *or* IND *if its advantage is small.*

The attack model for a DEM also follows the standard ideas.

Definition 6.15. *We allow the attacker $\mathcal{A}$ access to an oracle $\mathcal{O}$ after the challenge has been issued. The attack is said to be* passive (PAS) *if the oracle always returns the error symbol $\perp$. The attack is said to be a* chosen ciphertext attack (CCA) *if the oracle decrypts messages under the challenge key K (hence $\mathcal{O}(\chi) = DEM.Decrypt(\chi, K)$) with the exception that the oracle returns $\perp$ if it is queried with the challenge ciphertext.*

Obviously if a scheme is secure against an attack in the chosen ciphertext model then it is secure in the passive model, and it is easier to demonstrate the security of a scheme in the passive model than in the chosen ciphertext model. A result of [142] shows that it is possible to combine a DEM that is secure in the passive model with a message authentication code (MAC) (see Chapter 5) to give a DEM that is secure in the chosen ciphertext model.

6.3.5 Hybrid Security

Of course the aim of this entire section is to provide some insight into the security of a hybrid KEM-DEM based cryptosystem. So, whilst it might be very interesting to talk about the security of the KEM and the DEM in abstract models, it is useless unless we can combine these results to prove something about the security of the hybrid scheme. In particular we wish to show that a KEM-DEM based cryptosystem achieves the minimum security requirements specified by the NESSIE call, i.e. that the scheme is IND-CCA2 secure.

A security proof for the KEM-DEM construction was given in [142]. It basically shows that a KEM-DEM scheme composed of a secure DEM and a secure KEM will itself be secure. This shows that the method of construction is in some sense a "black-box" construction: any secure KEM and secure DEM can be chosen to give a secure scheme without reference to each other.

Theorem 6.2. *Suppose* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is a KEM-DEM scheme composed of a KEM and a DEM in the usual manner. If the KEM is IND-CCA2 secure and the DEM is IND-CCA secure then the overall hybrid scheme will be IND-CCA2 secure in the usual sense.*

There has been some work [231] that shows that if the public key pk used by the KEM is considered a system parameter available to all parties then the above theorem no longer holds. A separate security proof for this case was also proposed in that paper but, unfortunately, this security proof no longer allows a black-box construction.

Theorem 6.3. *Suppose* $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ *is a KEM-DEM scheme composed of a KEM and a DEM in the usual manner. Suppose further that the DEM takes the security parameter as an extra input. If the KEM is IND-CCA2 secure and the DEM is IND-CCA secure* **when the attacker has access to a decryption oracle for the KEM**, *then the overall hybrid scheme is IND-CCA2 secure in the standard sense.*

In practice it seems very unlikely that a DEM would consider using the public-key in any way that would compromise the security of the system. However it should be noted that the ElGamal scheme and the EPOC-2 scheme (see Sect. 6.4.4) both fit into this formal model of a KEM-DEM based scheme, as both contain modular arithmetic operations in the DEM phase.

One generic problem associated with KEM-DEM constructions is message expansion. Since only the ciphertext produced by the DEM depends on the message, we necessarily have that the complete ciphertext is bigger than the plaintext by at least a number of bits equal to the size of the encapsulation produced by the KEM. In practice the amount of message expansion may be larger even than this as the DEM ciphertext may contain bits that do not directly relate to the encryption of the message but are instead some kind of assurance of the integrity of the message (such as a MAC). Therefore in situations where message expansion is a critical factor, it might be best to avoid using a hybrid construction.

6.3.6 Assessment criteria

The assessment process for KEM-DEM based cryptosystems was roughly the same as for general asymmetric encryption schemes (see Sect. 6.2.7). The most important criterion is security. However, since this section of the NESSIE project is concerned with asymmetric techniques, we have concentrated our resources on examining the security of the key encapsulation mechanisms. We therefore assume the existence of a DEM that is equally secure for each KEM even in the

presence of a decryption oracle for that KEM. This greatly simplifies matters as the security of the KEM can be examined independently of the DEM. Again, each of the submitted primitives came with a proof of security and these proofs were evaluated in terms of the efficiency of their reductions and the hardness of the underlying trusted problems.

Next, performance was considered, on platforms of equal security, and lastly side-channel attacks were considered only if they could be applied to multiple platforms. Intellectual Property Rights (IPR) were also considered.

6.4 Asymmetric encryption primitives considered during Phase II

The following algorithms were selected for study during phase II of the NESSIE project:
– ACE-KEM,
– ECIES,
– EPOC-2,
– PSEC-KEM,

and, because they are under discussion in ISO [489], the following algorithms were selected for study during NESSIE phase II as *de facto* standards for asymmetric encryption algorithms:

– ECIES-KEM,
– RSA-KEM.

We deal with the security considerations for each algorithm in turn. Note that the algorithms given here are not complete specifications but rather the mathematical basis for each algorithm. In particular we assume that variables are stored in binary form even if they are integers, elliptic curve points, etc. We also assume that all hash functions, mask generating functions, key derivation functions and symmetric encryption schemes take inputs and produce outputs of a "correct" length for the asymmetric scheme. References are given to complete specifications which may be found on the NESSIE website.

6.4.1 ACE-KEM

The ACE-KEM cryptosystem is based on the work of Cramer and Shoup [142] and is purposely designed to be secure without needing the heuristic random oracle model. It has been submitted to NESSIE by IBM. The scheme consists loosely of the following algorithms. A complete specification is given in [489].

6.4.1.1 The design

Key Generation. ACE-KEM is described on an abstract group and so can be realised either as an elliptic curve scheme or as a scheme working in the multiplicative group of integers for some modulus. We will represent the group

as an additive group where group elements are represented as capital letters (as in an elliptic curve group). We assume that the group is a cyclic group generated by some element P of order p and that p has length equal to the security parameter λ. The key generation algorithm is a probabilistic algorithm that takes the group parameters (P, p, λ) as input. It runs as follows.

1. Generate random and independent integers $w, x, y, z \in \{0, \ldots, p-1\}$.
2. Set $W := wP$, $X := xP$, $Y := yP$ and $Z := zP$.
3. Set $pk := (P, p, W, X, Y, Z, \lambda)$ and $sk := (w, x, y, z, pk)$.
4. Output the key-pair (pk, sk).

Encapsulation Algorithm. The encapsulation algorithm is a probabilistic algorithm that takes the public-key pk as input. It uses a public and pre-agreed hash function $Hash(\cdot)$ and key derivation function $KDF(\cdot)$. It runs as follows.

1. Generate a random integer $r \in \{0, \ldots, p-1\}$.
2. Set $C_1 := rP$.
3. Set $C_2 := rW$.
4. Set $Q := rZ$.
5. Set $\alpha := Hash(C_1||C_2)$.
6. Set $C_3 := rX + \alpha rY$.
7. Set $C := (C_1, C_2, C_3)$.
8. Set $K := KDF(C_1||Q)$.
9. Output the encapsulated key-pair (K, C).

The ACE-KEM encapsulation algorithm is also shown pictorially in Fig. 6.1.

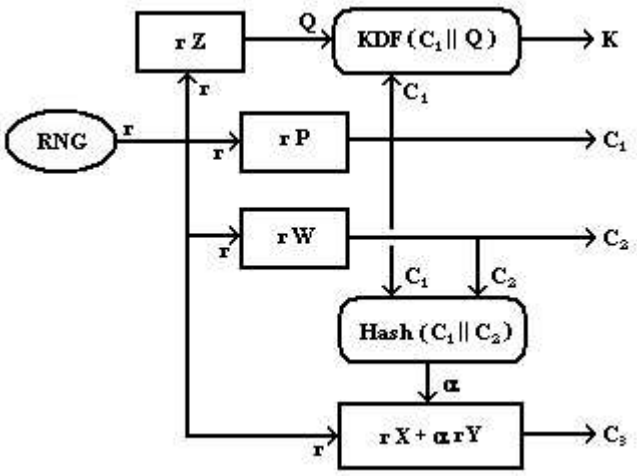

Fig. 6.1: The ACE-KEM encapsulation algorithm.

Decapsulation Algorithm. The decapsulation algorithm is a deterministic algorithm that takes a key encapsulation C and the secret-key sk as input. It also uses the pre-agreed hash function $Hash(\cdot)$ and key derivation function $KDF(\cdot)$. It runs as follows.

1. Parse the encapsulated key C as (C_1, C_2, C_3).
2. Set $\alpha := Hash(C_1||C_2)$.
3. Set $t := x + y\alpha$.
4. Check that $C_2 = wC_1$. If not, output `Invalid Ciphertext` and halt.
5. Check that $C_3 = tC_1$. If not, output `Invalid Ciphertext` and halt.
6. Set $Q := zC_1$.
7. Set $K := KDF(C_1||Q)$.
8. Output K.

6.4.1.2 Security analysis

The following is a summary of the security analysis of ACE-KEM; for more details see [119, 160].

The main advantage of ACE-KEM is that the security of the scheme can be proven without the use of the heuristic random oracle model (see Sect. 6.2.4). This combines nicely with the security proof for the generic hybrid construction discussed in Sect. 6.3.5 to form a hybrid scheme that is provably secure without the need for random oracles.

The security proof for ACE-KEM [142] reduces the problem of breaking ACE-KEM in the IND-CCA2 sense to the problem of solving the decisional Diffie-Hellman problem in the group generated by P. As we do not model the hash function or the key derivation function as random oracles we must formally define their security conditions.

Definition 6.16. *A hash function $Hash(\cdot)$ is 2^{nd} pre-image resistant if, given a value x, it is hard to find a value $y \neq x$ such that $Hash(x) = Hash(y)$. We define a (t, ϵ) attacker for a hash function to be a probabilistic Turing machine that runs in time bounded above by t and, given a randomly generated x, finds a second pre-image with probability at least ϵ.*

Definition 6.17. *A key derivation function $KDF(\cdot)$ is indistinguishable if, given x, it is computationally infeasible to distinguish between $KDF(x||y)$ and a random generated bit string of the same length when y is unknown. We define a (t, ϵ) attacker for a key derivation function to be a probabilistic Turing machine that runs in time bounded above by t and solves the above problem with probability at least $1/2 + \epsilon$.*

If there exists a (t, ϵ, q_D) attacker for ACE-KEM in the IND-CCA2 sense then there exists a (t_1, ϵ_1) solver for the decisional Diffie-Hellman problem on the group generated by P, a (t_2, ϵ_2) attacker for the hash function $Hash(\cdot)$ and a (t_3, ϵ_3) attacker for the key derivation function $KDF(\cdot)$ with

$$\epsilon \approx \epsilon_1 + \epsilon_2 + \epsilon_3 + \frac{q_D}{p}, \tag{6.9}$$

$$t \approx t_i \text{ for } i \in \{1, 2, 3\}. \tag{6.10}$$

Whilst the reduction to the decisional Diffie-Hellman (DDH) problem is tight, the assumption that the DDH problem is hard is quite a strong one. For better comparison to existing schemes the submitters have also shown that ACE-KEM is at least as secure as ECIES-KEM (see Sect. 6.4.3). Formally, if there exists a (t, ϵ, q_D) attacker for ACE-KEM then there exists a (t, ϵ, q_D) attacker for ECIES-KEM. This serves to show that, in the random oracle model, the security of ACE-KEM in the IND-CCA2 sense tightly reduces to the problem of solving the gap Diffie-Hellman problem on the group generated by P.

The relationship between ACE-KEM and ECIES-KEM also means that in the IND-CPA model, and with the help of the random oracle model, the security of ACE-KEM can be reduced to the problem of breaking the computational Diffie-Hellman (CDH) problem in the group $\langle P \rangle$.[4] Formally, if there exists a (t, ϵ) attacker for ACE-KEM then there exists an algorithm running in time t' that outputs a list of L elements and contains a solution to the CDH problem with probability at least ϵ' with

$$\epsilon' \approx \epsilon \,, \tag{6.11}$$

$$t' \approx t \,, \tag{6.12}$$

$$L \leq q_K \,, \tag{6.13}$$

where the attacker makes at most q_K queries of the random oracle simulating the key derivation function. Of course, we may now use the techniques of Sect. 6.2.3.2 to construct a (t'', ϵ'') solver for the CDH problem with

$$\epsilon'' \approx 1 - \frac{1}{2^k} \,, \tag{6.14}$$

$$t'' \approx 2k\lceil 1/\epsilon' \rceil t' + 2kL\lceil 1/\epsilon' \rceil T \,, \tag{6.15}$$

where T is the time taken to compute a group element of the form

$$x_1^{-1} y_1^{-1} \left\{ P' - a x_1 y_2 P - b x_2 y_1 P + x_2 y_2 P \right\} ,$$

for random integers x_1, y_1, x_2, y_2 and a random elliptic curve point P'.

Recently a new version of the ACE-KEM scheme has been proposed by Lucks [343]. This new scheme works in the multiplicative group of integers modulo n, where $n = PQ$, $P = 2p + 1$, $Q = 2q + 1$ and P, Q, p, q are prime numbers. This scheme has the disadvantage of working in a very specific, multiplicative group (and so will probably have longer keys than the elliptic curve version) but has the advantage of a stronger security proof. The Lucks scheme is secure in the standard model providing the factoring problem is secure and the DDH problem is secure in the group of quadratic residues modulo n. This is worse than ACE-KEM as ACE-KEM does not require the factoring problem to be difficult. However, in the random oracle model, the Lucks scheme reduces to solving the CDH problem

[4] In the IND-CCA2 attack model, and using the random oracle model, the security of ACE-KEM can be directly reduced to the problem of breaking the CDH problem in the group generated by P [487]. However this reduction is far from tight.

in a very efficient manner. This is a significant improvement over ACE-KEM in theoretical security.

Along with most of the other asymmetric encryption schemes, ACE-KEM is vulnerable to a fault attack that recovers the secret key [164]. It also appears to be vulnerable to power analysis during the scalar multiplications of the elliptic curve faults. This is a common problem with schemes based on elliptic curves, for more information see Sect. A.1.2.3.

6.4.2 ECIES

ECIES is a hybrid encryption scheme submitted to NESSIE by Certicom Corp. It loosely consists of the following algorithms. A complete specification can be found in [115].

6.4.2.1 The design

Key Generation. Since ECIES is an elliptic curve based cryptosystem, it is necessary to generate a suitably secure elliptic curve E and a point $P \in E$ that has prime order p. We will assume that the length of p is equal to the security parameter λ. The key generation algorithm for ECIES is a probabilistic algorithm that takes (E, P, p, λ) as input. It runs as follows. (For notational purposes, let $\mathcal{O}$ be the 'point at infinity' – the identity element of an elliptic curve group).

1. Randomly generate an integer $s \in \{1, \ldots, p-1\}$.
2. Set $W := sP$.
3. Set $pk := (E, P, p, W, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm is a probabilistic algorithm that takes as input a message m and the public-key pk. The scheme also relies on a set of public and pre-agreed functions that are available to all parties. These include a key derivation function $KDF(\cdot)$, a message authentication code algorithm $MAC(\cdot, \cdot)$ and a symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It runs as follows. (For notational purposes, let $\mathcal{O}$ be the 'point at infinity' – the identity element of an elliptic curve group).

1. Generate a random integer $r \in \{1, \ldots, p-1\}$.
2. Set $C_1 := rP$.
3. Set $Q := rW$.
4. Check that $Q \neq \mathcal{O}$. If so, output `Invalid Encryption` and halt.
5. Set x to be the x-coordinate of Q.
6. Set $K := KDF(x)$.
7. Parse K as EK and MK, where EK is a suitably sized key for the symmetric encryption scheme and MK is a suitably sized key for the message authentication scheme. [5]

[5] Note that if the lengths of EK and MK are not predetermined, then the attack of Shoup [489, 15.6.4] applies.

8. Encrypt the message m using the symmetric encryption scheme under the key EK, i.e. $C_2 := Sym.Encrypt(m, EK)$.
9. Generate a message authentication code (MAC) for the ciphertext C_2 using the message authentication scheme under the key MK, i.e. $C_3 := MAC(C_2, MK)$.
10. Output the ciphertext $C := (C_1, C_2, C_3)$.

Decryption Algorithm. The decryption algorithm is a deterministic algorithm that takes a ciphertext C and the secret-key sk as input. It also uses the same pre-agreed key derivation function $KDF(\cdot)$, message authentication code algorithm $MAC(\cdot, \cdot)$ and symmetric encryption scheme ($Sym.Encrypt$, $Sym.Decrypt$) as the encryption algorithm. It runs as follows.

1. Parse C as (C_1, C_2, C_3).
2. Set $Q := sC_1$.
3. Check that $Q \neq \mathcal{O}$. If so, output `Invalid Ciphertext` and halt.
4. Set x to be the x-coordinate of Q.
5. Set $K := KDF(x)$.
6. Parse K as EK and MK, where EK is a suitably sized key for the symmetric encryption scheme and MK is a suitably sized key for the message authentication scheme.
7. Check that C_3 is a valid message authentication code for C_2 under the key MK, i.e. that $C_3 = MAC(C_2, MK)$. If not, output `Invalid Ciphertext` and halt.
8. Decrypt the ciphertext C_2 using the symmetric encryption scheme under the key EK, i.e. $m := Sym.Decrypt(C_2, EK)$.
9. Output m.

6.4.2.2 Security analysis

The following is a summary of the security analysis of ECIES; for more details see [477].

The first thing to notice about the ECIES algorithm as it stands is that it is not secure in the IND-CCA2 sense. If an attacker is given a challenge ciphertext (C_1, C_2, C_3) then he may submit the ciphertext $(-C_1, C_2, C_3)$ to the decryption oracle (providing $C_1 \neq -C_1$, a condition which occurs with overwhelming probability) and the oracle will return the correct message m. Therefore if we are going to find any meaningful result on the security of ECIES in this model then we will have to formally deny the attacker the power to make these queries. Shoup [489] calls this property *benign malleability*.

The security proof for ECIES is sketched in [2]. This paper describes the security of the DHAES scheme, a scheme similar to ECIES but defined on an abstract cyclic group of order p rather than specifically on an elliptic curve group. It differs from ECIES in the generation of the symmetric key K which, in DHAES, is derived from a complete representation of Q and from C_1. The security proof shows that, provided the symmetric encryption scheme and the MAC are in some sense secure, the problem of breaking the scheme reduces to the problem

of differentiating the output of the key derivation function from a random string of the same size.

A security proof for ECIES in the generic group model (see Sect. 6.2.5) has also been proposed [493].

ECIES is essentially an example of ECIES-KEM (see Sect. 6.4.3) used with a particular data encapsulation mechanism (known as DEM-1 [489]). We consider ECIES-KEM to be more flexible than ECIES without having any computational or security loss.

6.4.3 ECIES-KEM

ECIES-KEM is essentially the asymmetric heart of ECIES described in the KEM-DEM framework. Although ECIES-KEM was never formally submitted to the NESSIE project, it is considered by NESSIE as a *de facto* standard for a KEM-DEM based cryptosystem and because it has been proposed for standardisation in the ISO/IEC standard 18033-2 [489]. It consists loosely of the following algorithms. A complete specification can be found in [489].

6.4.3.1 The design

Key Generation. ECIES-KEM, like ECIES, is an elliptic curve scheme. This means that before the scheme can be implemented a suitably secure elliptic curve E must have been generated and a point $P \in E$ with prime order p must have been chosen. We assume that the length of p is equal to the security parameter λ. The key generation algorithm is a probabilistic algorithm that takes the elliptic curve parameters (E, P, p, λ) as input and runs as follows.

1. Randomly generate an integer $s \in \{1, \ldots, p-1\}$.
2. Set $W := sP$.
3. Set $pk := (E, P, p, W, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encapsulation Algorithm. The encapsulation algorithm is a probabilistic algorithm that takes the public-key as input. It uses a public and pre-agreed key derivation function $KDF(\cdot)$ that must be available to all parties wishing to use the scheme. Its encapsulation algorithm runs as follows.

1. Generate a random integer $r \in \{1, \ldots, p-1\}$.
2. Set $C := rP$.
3. Set x to be the x-coordinate of rW.
4. Set $K := KDF(C||x)$.
5. Output the encapsulated key-pair (K, C).

Decapsulation Algorithm. The decapsulation algorithm is a deterministic algorithm that takes as input an encapsulated key C and the secret-key sk. It also uses the pre-agreed key derivation function $KDF(\cdot)$ that was used in the encapsulation process. The algorithm runs as follows. (For notational purposes, let $\mathcal{O}$ be the 'point at infinity' – the identity element of an elliptic curve group).

1. Set $Q := sC$.
2. Check that $Q \neq \mathcal{O}$. If so, output `Invalid Ciphertext` and halt.
3. Set x to be the x-coordinate of Q.
4. Set $K := KDF(C||x)$.
5. Output K.

6.4.3.2 Security analysis

It is very easy to show that ECIES-KEM is IND-CCA2 secure in the random oracle model [162]. The security of the scheme can be very efficiently reduced to the problem of solving the gap Diffie-Hellman problem in the elliptic curve subgroup generated by P. Formally, if there exists a (t, ϵ, q_D) attacker for ECIES-KEM then there exists a (t', ϵ') solver for the gap Diffie-Hellman problem in the group $\langle P \rangle$ with

$$\epsilon' \approx \epsilon\,, \tag{6.16}$$
$$t' \approx t + 2q_K T\,, \tag{6.17}$$

where

– q_K is the number of queries that the attacker submits to the random oracle simulating the key derivation function,
– and T is the time taken to check a Diffie-Hellman triple (i.e. the time taken for the oracle to check whether $g^c = g^{ab}$ for a triple (g^a, g^b, g^c)).

In the IND-CPA model, the security of ECIES-KEM can be improved. The security of the scheme, in the random oracle model, can be reduced to the problem of breaking the computational Diffie-Hellman (CDH) problem in the group $\langle P \rangle$. Formally, if there exists a (t, ϵ) attacker for ECIES-KEM then there exists an algorithm running in time t' that outputs a list of L elements and contains a solution to the CDH problem with probability at least ϵ' with

$$\epsilon' \approx \epsilon\,, \tag{6.18}$$
$$t' \approx t\,, \tag{6.19}$$
$$L \leq q_K\,, \tag{6.20}$$

where the attacker makes at most q_K queries of the random oracle simulating the key derivation function. Of course, we may now use the techniques of Sect. 6.2.3.2 to construct a (t'', ϵ'') solver for the CDH problem with

$$\epsilon'' \approx 1 - \frac{1}{2^k}\,, \tag{6.21}$$
$$t'' \approx 2k\lceil 1/\epsilon' \rceil t' + 2kL\lceil 1/\epsilon' \rceil T\,, \tag{6.22}$$

where T is the time taken to compute a group element of the form

$$x_1^{-1} y_1^{-1} \left\{ P' - a x_1 y_2 P - b x_2 y_1 P + x_2 y_2 P \right\},$$

for random integers x_1, y_1, x_2, y_2 and a random elliptic curve point P'.

The ECIES-KEM mechanism can be viewed as a subroutine of both the PSEC-KEM (see Sect. 6.4.5) and the ACE-KEM algorithms (see Sect. 6.4.1). This means that the performance costs of PSEC-KEM and ACE-KEM are at least that of ECIES-KEM.

The only side-channel attacks that ECIES-KEM seems to be vulnerable to is a fault attack [164] and a power attack based on its use of elliptic curve groups (see Sect. A.1.2.3).

6.4.4 EPOC-2

EPOC-2 is a hybrid asymmetric encryption scheme submitted by NTT Corporation. It consists loosely of the following algorithms. A complete specification is given in [404].

6.4.4.1 The design

Key Generation. The key generation algorithm is a probabilistic algorithm that takes a security parameter λ as input and runs as follows.

1. Generate (usually randomly) two distinct λ-bit primes p and q. Set $n := p^2q$.
2. Generate (usually randomly) an element $g \in (\mathbb{Z}/n\mathbb{Z})^*$ such that $g_p := g^{p-1} \bmod p^2$ has order p in $(\mathbb{Z}/p^2\mathbb{Z})^*$.
3. Set $h := g^n \bmod n$.
4. Set $w := (g_p - 1)/p \bmod p$.
5. Set $pk := (n, g, h, \lambda)$ and $sk := (p, q, w, pk)$.
6. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm takes as input the public-key pk and a message m. It also relies on a set of pre-agreed system parameters. In order for the scheme to work, the two parties must have agreed on the use of a hash function $Hash(\cdot)$, a mask generating function $MGF(\cdot)$, a key derivation function $KDF(\cdot)$ and a symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It runs as follows.

1. Generate a random octet string $R \in \{0, \ldots, 255\}^{\lfloor(\lambda-1)/8\rfloor}$.
2. Derive a suitable symmetric key $K := KDF(R)$.
3. Encrypt the message m using the symmetric scheme and the key K, $C_2 := Sym.Encrypt(m, K)$.
4. Set $DB := m||R||C_2$.
5. Set $H := MGF(Hash(DB))$.
6. Set $C_1 := g^R h^H \bmod n$.
7. Output the ciphertext $C = (C_1, C_2)$.

This process is also shown pictorially in Fig. 6.2.

Decryption Algorithm. The decryption algorithm uses the same set of system parameters as the encryption algorithm, i.e. a pre-agreed hash function $Hash(\cdot)$, mask generating function $MGF(\cdot)$, key derivation function $KDF(\cdot)$ and symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It takes as input a ciphertext C and the secret-key sk and runs as follows.

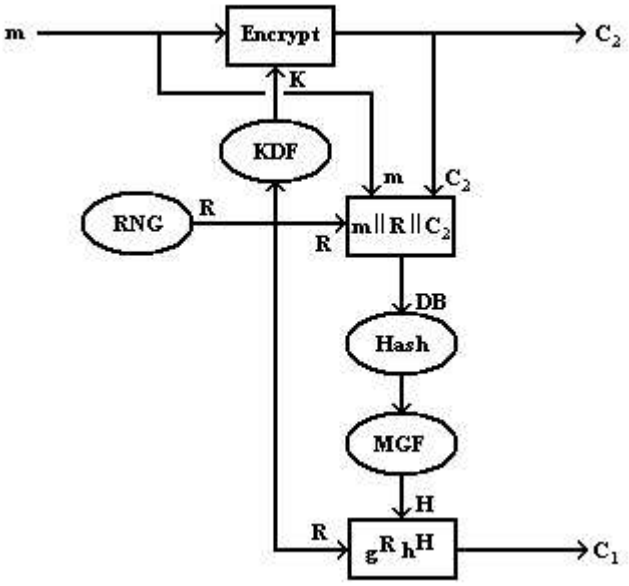

Fig. 6.2: The EPOC-2 encryption algorithm.

1. Parse the ciphertext C into (C_1, C_2).
2. Check that $0 \leq C_1 < n$. If not, output `Invalid Ciphertext` and halt.
3. Set $C_1' := C_1^{p-1} \bmod p^2$.
4. Set $C_1'' := (C_1' - 1)/p \bmod p$.
5. Set $R := C_1''/w \bmod p$.
6. Check that $0 \leq R < 256^{\lfloor(\lambda-1)/8\rfloor}$. If not, output `Invalid Ciphertext` and halt.
7. Derive a suitable symmetric key $K := KDF(R)$.
8. Decrypt the ciphertext C_2 using the symmetric encryption scheme and the key K, $m := Sym.Decrypt(C_2, K)$.
9. Set $DB := m||R||C_2$.
10. Set $H := MGF(Hash(DB))$.
11. Derive a reduced public key $\hat{PK}$ with $\hat{g} := g \bmod q$, $\hat{h} := h \bmod q$ and $\hat{H} := H \bmod q - 1$.
12. Calculate $\hat{C}_1 = \hat{g}^R \hat{h}^{\hat{H}} \bmod q$.
13. Check that $\hat{C}_1 = C_1 \bmod q$. If not, output `Invalid Ciphertext` and halt.
14. Output m.

6.4.4.2 Security analysis

The following is a summary of the security analysis of EPOC-2; for more details see [519, 159]. Some arguments as to why EPOC-2 was selected to be studied in

NESSIE phase II over the other EPOC candidates (see Sect. 6.5.1 and Sect. 6.5.2) can be found in [482].

It should be noted that the structure of EPOC-2 is very similar to that of a KEM-DEM based cryptosystem. It has, presumably, not been phrased in these terms because the 'KEM' output would not be indistinguishable from a random key in the CCA2 model. This is a good example of a hybrid scheme that is secure but does not satisfy the security requirements for a KEM-DEM based scheme. It should also be noted that the 'DEM' section uses group operations derived from the public key as a MAC tag, and this is formally excluded from a DEM construction as the DEM has no access to the public key. In the security proof the symmetric cipher is assumed to be a Vernam cipher, in which the key is bitwise XORed with the message to form the ciphertext.

The security of EPOC-2 is based on the Okamoto-Uchiyama cryptosystem [420], which is provably as secure as factoring in the IND-CPA model, and an improved version of the Fujisaki-Okamoto transform [205] which is specific to the Okamoto-Uchiyama cryptosystem. The security proof [201] for the scheme proves that, in the random oracle model, one can reduce the problem of attacking the scheme in the IND-CCA2 setting to the problem of factoring the modulus $n = p^2q$. For convenience we define $|n|$ to be the size of the integer n in bits. Formally, if there exists a (t, ϵ, q_D) IND-CCA2 attacker for EPOC-2 in the random oracle model then there exists a (t', ϵ') solver for the problem of factoring the modulus n with

$$\epsilon' \approx \frac{\epsilon}{3}(1 - 2^{-3\lambda+3})(1 - 2^{-\gamma})^{q_D}\,, \tag{6.23}$$

$$t' \approx t + q_H T_1 + q_H q_D T_2\,, \tag{6.24}$$

where

- q_H is the number of queries the attacker makes of the random oracle,
- γ is a constant that depends only upon the public key,
- T_1 is the time taken to compute the greatest common divisor of two $|n|$-bit numbers,
- and T_2 is the time taken to check an equation of the form $C_1 = \hat{g}^R \hat{h}^{\hat{H}} \bmod q$.

The problem of factoring a number of the form $n = p^2q$ is one of the less well researched trusted cryptographic problems used by the NESSIE phase II submissions. Most of the research into factoring algorithms concentrates on attempting to factor integers of the form pq. The attack of [97], whilst not directly applicable to the Okamoto-Uchiyama modulus n, only serve to underline the fact that factoring a number of the form p^2q is unlikely to be harder than factoring a number of the form pq. Furthermore, factoring the modulus is enough to recover the secret key (which depends only on p and q). So any attack that can reliably distinguish messages in the CCA2 model also provides, in the random oracle model, an attack that can recover the secret key.

EPOC-2 is also vulnerable to a few important side-channel attacks (see Sect. 6.2.6). Whilst it is true that EPOC-2 is the only primitive in Phase II

that does not seem susceptible to fault attacks [164], it is vulnerable to a Hamming weight attack [164] and to an error message attack [163]. This seems to be indicative of an inherent weakness of EPOC-2; the security proof for the primitive shows that if any information can be gained about the value derived from the Okamoto-Uchiyama decryption process (the number R calculated in step 5 of the decryption process) then an attacker is likely to be able use this information to launch a key recovery attack. For this reason we consider EPOC-2 to be weak against side-channel attacks.

For fair comparison EPOC-2 was not only evaluated against the other NESSIE candidates but also against existing schemes with similar security assumptions [175]. The scheme was compared against the HIME(R) scheme [403] and the Rabin-SAEP scheme [92], the security of both of which can be reduced to the problem of factoring a modulus n of the form $n = pq$ or $n = p^r q$ for small r. Whilst all of the schemes had comparable security reductions, EPOC-2 was found to have a comparatively large key size and a slow running time.

6.4.5 PSEC-KEM

PSEC-KEM is a tweaked version of PSEC-2 and was submitted by the NTT Corporation. It consists loosely of the following algorithms. A complete specification is available in [405].

6.4.5.1 The design

Key Generation. Since PSEC-KEM is defined over an elliptic curve, a suitable curve E will have to be generated before the key generation algorithm is executed. The curve should have a point P that generates a secure cyclic subgroup of E with prime order p. We assume that p is of size λ where λ is the security parameter.

The key generation algorithm is a probabilistic algorithm that takes the elliptic curve E, the point P and the order p of P as input. It runs as follows.

1. Generate an integer s uniformly at random from $\{0, \ldots, p-1\}$.
2. Set $W := sP$.
3. Set $pk := (E, P, W, p, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encapsulation Algorithm. The key encapsulation algorithm is a probabilistic algorithm that takes as input the public-key pk. In order for the scheme to work the two communicating parties must have agreed on the use of a common key derivation function $KDF(\cdot)$. It runs as follows.

1. Generate a suitably sized random bit string r (of size comparable to p, say).
2. Set $H := KDF(0_{32}||r)$, where 0_{32} is the 32-bit representation of the integer 0.
3. Parse H as $t||K$ where t is a $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
4. Set $\alpha := t \bmod p$.
5. Set $Q := \alpha W$.

6. Set $C_1 := \alpha P$.
7. Set $C_2 := r \oplus KDF(1_{32}||C_1||Q)$, where 1_{32} is the 32-bit representation of the integer 1.
8. Set $C := (C_1, C_2)$.
9. Output the encapsulated key-pair (K, C).

The PSEC-KEM encapsulation algorithm is also shown pictorially in Fig. 6.3.

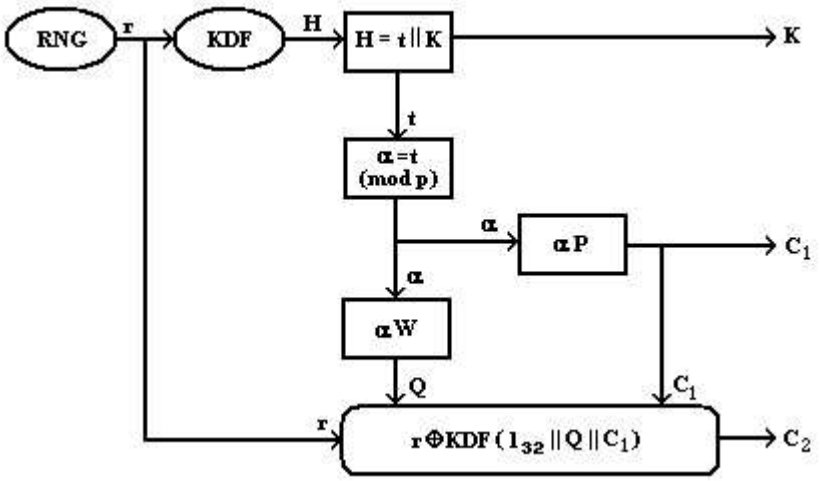

Fig. 6.3: The PSEC-KEM encapsulation algorithm.

Decapsulation Algorithm. The decapsulation algorithm is a deterministic algorithm that takes as input a key encapsulation C and the secret-key sk. It also uses the pre-agreed key derivation function $KDF(\cdot)$. It runs as follows.

1. Parse C as (C_1, C_2).
2. Set $Q := sC_1$.
3. Set $r := C_2 \oplus KDF(1_{32}||C_1||Q)$, where 1_{32} is the 32-bit representation of the integer 1.
4. Set $H := KDF(0_{32}||r)$, where 0_{32} is the 32-bit representation of the integer 0.
5. Parse H as $t||K$, where t is an $(\lambda + 128)$-bit integer and K is a suitably sized symmetric key.
6. Set $\alpha := t \bmod p$.
7. Check $C_1 = \alpha P$. If not, output `Invalid Ciphertext` and halt.
8. Output K.

6.4.5.2 Security analysis

In many ways the PSEC family of asymmetric encryption schemes are counterparts to the EPOC schemes, in the sense that they both use the same constructions to turn a base problem into useable asymmetric encryption schemes. However, when the schemes were tweaked at the start of NESSIE phase II, PSEC-2 was rephrased as a KEM-DEM based cryptosystem (see Sect. 6.3), whereas EPOC-2 (see Sect. 6.4.4) was not. The following security analysis is a summary of the work found mainly in [480, 481]. Some arguments as to why PSEC-2 was chosen to be studied in NESSIE phase II over the other PSEC candidates (see Sect. 6.5.3 and Sect. 6.5.4) were given in [482].

The security proof for PSEC-KEM can be found in [489]. It shows that, in the random oracle model, the problem of breaking the system can be reduced to the problem of solving a computational Diffie-Hellman problem on the elliptic curve E. Formally, if there exists a (t, ϵ, q_D) IND-CCA2 attacker for PSEC-KEM then there exists an algorithm running in time t' that outputs a list of L elements that contains the solution to the CDH problem with probability ϵ' and

$$t' \approx t\,, \tag{6.25}$$

$$\epsilon' \approx \epsilon - \frac{q_{K_0} + 2q_D}{p} - \frac{q_{K_0} + q_D}{2^\lambda}\,, \tag{6.26}$$

$$L \leq q_D + q_{K_1}\,, \tag{6.27}$$

where

- the encryption algorithm generates a random integer of length λ in step 1,
- the attacker makes at most q_{K_i} queries to the random oracle representing the key-derivation function where the initial 32-bits are a representation of the integer i.

Of course we may now use the probability amplification techniques given in Sect. 6.2.3.2 to give a (t'', ϵ'') solver for the CDH problem on E with

$$\epsilon'' \approx 1 - \frac{1}{2^k}\,, \tag{6.28}$$

$$t'' \approx 2k\lceil 1/\epsilon' \rceil t' + 2kL\lceil 1/\epsilon' \rceil T\,, \tag{6.29}$$

where T is the time taken to compute a group element of the form

$$x_1^{-1} y_1^{-1} \left\{ P' - a x_1 y_2 P - b x_2 y_1 P + x_2 y_2 P \right\},$$

for random integers x_1, y_1, x_2, y_2 and a random elliptic curve point P'.

Since the reduction shown for PSEC-KEM appears to be close to optimal in a group for which a DDH oracle does not exist, we do not expect the security reduction to be significantly improved in the IND-CPA model.

PSEC-KEM is an authenticated KEM. The PSEC-KEM algorithm can be viewed as using a version of ECIES-KEM (see Sect. 6.4.3) as a mask for the key. It also produces data that is used to perform a consistency check.

Along with most of the other NESSIE phase II candidates, PSEC-KEM is vulnerable to a fault attack and, along with the other schemes that are based on elliptic curves, it does seem to be vulnerable to power analysis (see Sect. A.1.2.3). It does not appear to be vulnerable to either an error message attack or a Hamming weight attack [164].

6.4.6 RSA-KEM

The RSA based key encapsulation mechanism is one of the simplest key encapsulation mechanisms. It uses the RSA trapdoor permutation as a security mechanism (see Sect. 6.3.2). Although RSA-KEM was never formally submitted to the NESSIE project, it is included in the evaluation process as a *de facto* standard for a KEM-DEM based cryptosystem and because it has been proposed for inclusion in the ISO/IEC standard 18033-2 [489]. It consists loosely of the following algorithms. A complete specification can be found in [274, 489].

6.4.6.1 The design

Key Generation. The key generation algorithm for RSA-KEM is very similar to that of RSA-OAEP (see Sect. 6.5.5) as both concentrate on producing a valid RSA key. The RSA-KEM key generation algorithm is a probabilistic algorithm that takes the security parameter λ as input and runs as follows.

1. Generate a public exponent e, an odd integer greater than 1.
2. Randomly generate a prime p of length λ such that $gcd(p-1, e) = 1$.
3. Randomly generate a prime q of length λ such that $gcd(q-1, e) = 1$.
4. Set $n := pq$.
5. Set d to be the unique integer in $\mathbb{Z}/\lambda(n)\mathbb{Z}$ such that $ed \equiv 1 \bmod \lambda(n)$, where $\lambda(n) = l.c.m.(p-1, q-1)$.
6. Set $pk := (n, e, \lambda)$ and $sk := (d, pk)$.
7. Output the key-pair (pk, sk).

Encapsulation Algorithm. The encapsulation algorithm is a probabilistic algorithm that takes as input the public key pk. It also uses a common, public key derivation function $KDF(\cdot)$ that is available to all parties. It runs as follows.

1. Generate a random integer $r \in \{0, \ldots, n-1\}$.
2. Set $C := r^e \bmod n$.
3. Set $K := KDF(r)$.
4. Output the encapsulated key-pair (K, C).

Decapsulation Algorithm. The decapsulation algorithm is a deterministic algorithm that takes an encapsulation C and the secret-key sk as input. It also uses the pre-agreed key derivation function $KDF(\cdot)$ and runs as follows.

1. Set $r := C^d \bmod n$.
2. Set $K := KDF(r)$.
3. Output K.

6.4.6.2 Security analysis

The following is a summary of the security analysis for RSA-KEM and is found mainly in [231] or derived from the general results on KEM-DEM cryptosystems in [160].

The security of RSA-KEM is based on the security of the RSA cryptosystem [452] in the OW-CPA model. We use the general result about security mechanisms of [160] to show that, in the random oracle model, the security of RSA-KEM can be reduced to the RSA problem. Formally, if there exists a (t, ϵ, q_D) attacker for RSA-KEM in the IND-CCA2 sense then there exists a (t', ϵ') solver for the RSA problem with

$$\epsilon' \approx \epsilon\,, \tag{6.30}$$

$$t' \approx t + (q_K + q_D)T\,, \tag{6.31}$$

where

– q_K is the number of queries the attacker makes to the random oracle,
– and T is the time taken to calculate x^e for some x.

Since the reduction shown for RSA-KEM appears to be close to optimal, we do not expect the security reduction to be significantly improved in the IND-CPA model.

The RSA cryptosystem exhibits some homomorphic properties, most noticeably that if $C_1 = m_1^e \bmod n$ and $C_2 = m_2^e \bmod n$ then $C_1 C_2 \bmod n$ is the encryption of $m_1 m_2 \bmod n$. Whilst this is highly desirable in certain applications, it does allow unknown messages to be manipulated in quite specific ways. In RSA-KEM we rely on the nature of the key derivation function $KDF(\cdot)$ to destroy any relations between keys that might be a result of relations in the encapsulations. Hence the properties of the key derivation function are more critical in RSA-KEM than they might be in, say, ACE-KEM (see Sect. 6.4.1). This homomorphic property is also useful when implementing side-channel attacks. It has been shown that RSA-KEM is vulnerable to a fault attack that recovers the secret key [292], a chosen modulus attack [292, 173] and a Hamming weight attack [292, 164]. See Sect. 6.2.6 for a discussion of the relevance of these attacks.

RSA-KEM was selected as a suitable *de facto* standard for hybrid RSA based cryptosystems after it was evaluated against RSA-REACT [419] in a paper by Granboulan [231]. RSA-KEM was selected as the *de facto* standard because its security was shown to be at least that of RSA-REACT and it was shown to have better performance characteristics.

6.5 Asymmetric encryption primitives not selected for Phase II

The following algorithms were submitted to NESSIE but not selected for further study in the second phase of the project:
– EPOC-1

– EPOC-3
– PSEC-1
– PSEC-3
– RSA-OAEP

Again we choose to specify the algorithms in a general mathematical way rather than as a detailed specification. In particular we assume that variables are stored in binary form even if they are integers, elliptic curve points, etc. We also assume that hash functions, mask generating functions, key derivation functions and symmetric encryption schemes all take inputs and produce outputs of a "correct" length for the asymmetric scheme. References are given to complete specifications for the algorithms.

6.5.1 EPOC-1

EPOC-1 was the first of the EPOC series submitted by the NTT Corporation. It is based on the Okamoto-Uchiyama problem [420], which is provably secure in the IND-CPA sense in the random oracle model. The techniques of [204] are then used to transform this into an asymmetric encryption scheme that is IND-CCA2 secure in the random oracle model. The scheme consists loosely of the following algorithms. A complete specification can be found in [203].

6.5.1.1 The design

Key Generation. The key generation algorithm is a probabilistic algorithm that takes a security parameter λ as input and runs as follows.

1. Randomly generate two λ-bit primes p and q. Set $n := p^2q$.
2. Randomly generate an element $g \in (\mathbb{Z}/n\mathbb{Z})^*$ such that $g_p := g^{p-1} \bmod p^2$ has order p in $(\mathbb{Z}/p^2\mathbb{Z})^*$.
3. Randomly generate an element $h_0 \in (\mathbb{Z}/n\mathbb{Z})^*$ and set $h := h_0^n \bmod n$.
4. Set $w := \frac{g_p-1}{p} \bmod p$.
5. Choose positive integers $mLen$ and $rLen$ such that $mLen + rLen \leq \lambda - 1$.
6. Set $pk := (n, g, h, mLen, rLen, \lambda)$ and $sk := (p, q, w, pk)$.
7. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm takes as input the public-key pk and a message m of length $mLen$. It also uses a pre-agreed hash function $Hash(\cdot)$. It runs as follows.

1. Generate a random string R of length $rLen$ and compute $r := Hash(m||R)$.
2. Set $C := g^{m||R}h^r \bmod n$.
3. Output the ciphertext C.

Decryption Algorithm. The decryption algorithm takes as input a ciphertext C and the secret-key sk. It also uses the pre-agreed hash function $Hash(\cdot)$ and runs as follows.

1. Set $C_p := C^{p-1} \bmod p^2$.
2. Set $C'_p := \frac{C_p-1}{p} \bmod p$.

3. Set $X := \frac{C'_p}{w} \bmod p$.
4. Check that $0 \leq X \leq 2^{mLen+rLen}$. If not, output `Invalid Ciphertext` and halt.
5. Check that $C = g^X h^{Hash(X)}$. If not, output `Invalid Ciphertext` and halt.
6. Set m to be the first $mLen$ bits of X.
7. Output the message m.

6.5.1.2 Security analysis

The following is a summary of the security analysis of EPOC-1 ; for more details see [519]. Some of the arguments as to why EPOC-2 (see Sect. 6.4.4) was selected to be studied in NESSIE phase II over EPOC-1 were given in [482].

It has been shown that, in the random oracle model, EPOC-1 is secure in the IND-CCA2 sense [203]. The security of the scheme reduces to the problem of solving the p-subgroup membership problem on the group $(\mathbb{Z}/n\mathbb{Z})^*$ (see Sect. 6.2.3). Formally, if (t, ϵ, q_D) is an IND-CCA2 attacker for the EPOC-1 scheme then there exists a (t', ϵ') solver for the p-subgroup problem with

$$\epsilon' \approx (\epsilon - q_H 2^{-(rLen-1)})(1 - 2^{-2\lambda})^{q_D}\,, \tag{6.32}$$

$$t' \approx t + q_H(T + c\lambda)\,, \tag{6.33}$$

where

– q_H is the number of queries the attacker makes to the random oracle,
– T is the time taken to calculate $g^m h^r \bmod n$,
– and c is a constant.

EPOC-1 was designed for key distribution, which is outside the scope of the NESSIE project. The scheme was not selected for NESSIE phase II because it had a worse security reduction than EPOC-2 for similar performance costs. Furthermore, the submitters withdrew their support for EPOC-1 in favour of EPOC-2 (see Sect. 6.4.4).

6.5.2 EPOC-3

EPOC-3 was submitted by the NTT Corporation and, like the other members of the EPOC series, uses the Okamoto-Uchiyama cryptosystem [420]. This time the submitters use the REACT transform [419] to improve the security of the basic scheme. The EPOC-3 scheme consists loosely of the following three algorithms. A complete specification is given in [203].

6.5.2.1 The design

Key Generation. The key generation algorithm is a probabilistic algorithm that takes a security parameter λ as input and runs as follows.

1. Randomly generate two λ-bit primes p and q. Set $n := p^2 q$.
2. Randomly generate an element $g \in (\mathbb{Z}/n\mathbb{Z})^*$ such that $g_p := g^{p-1} \bmod p^2$ has order p in $(\mathbb{Z}/p^2\mathbb{Z})^*$.

3. Randomly generate an element $h_0 \in (\mathbb{Z}/n\mathbb{Z})^*$ and set $h := h_0^n \bmod n$.
4. Set $w := \frac{g_p - 1}{p} \bmod p$.
5. Set $pk := (n, g, h, \lambda)$ and $sk := (p, q, w, pk)$.
6. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm takes as input the public-key pk and a message m. It also relies on a set of pre-agreed system parameters. In order for the scheme to work the two communicating parties must have agreed upon a public choice of a hash function $Hash(\cdot)$, a key derivation function $KDF(\cdot)$ and a symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). The encryption algorithm runs as follows.

1. Generate two independent and uniform random bit strings r and R of length $\lambda - 1$.
2. Derive a suitable symmetric key $K := KDF(R)$.
3. Set $C_1 := g^R h^r \bmod n$.
4. Set $C_2 := Sym.Encrypt(m, K)$, i.e. the encryption of the message m in the symmetric scheme under the key K.
5. Set $C_3 := Hash(C_1||C_2||R||m)$.
6. Output the ciphertext $C = (C_1, C_2, C_3)$.

Decryption Algorithm. The decryption algorithm uses the same set of system parameters as the encryption algorithm: a hash function $Hash(\cdot)$, a key derivation function $KDF(\cdot)$ and a symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It takes as input a ciphertext C and the secret key sk. It runs as follows.

1. Parse C as an appropriately sized triple (C_1, C_2, C_3).
2. Set $C_1' := C_1^p \bmod p^2$.
3. Set $C_1'' := \frac{C_1' - 1}{p} \bmod p$.
4. Set $R := \frac{C_1''}{w} \bmod p$.
5. Derive a suitable symmetric key $K := KDF(R)$.
6. Decrypt the ciphertext C_2 using the symmetric encryption scheme and the key K, i.e. set $m := Sym.Decrypt(C_2, K)$.
7. Check if $R < 2^{\lambda - 1}$. If not, output `Invalid Ciphertext` and halt.
8. Check that $C_3 = Hash(C_1||C_2||R||m)$. If not, output `Invalid Ciphertext` and halt.
9. Output the message m.

6.5.2.2 Security analysis

The following is a summary of the security analysis of EPOC-3; for more details see [519]. Some of the arguments as to why EPOC-2 (see Sect. 6.4.4) was selected to be studied in NESSIE phase II over EPOC-3 were given in [482].

It has been shown in [203] that EPOC-3 is IND-CCA2 secure in the random oracle model, and reduces to the problem of solving the gap-factoring problem for the modulus n. As with EPOC-2 (see Sect. 6.4.4) we will assume that the symmetric encryption system used is a Vernam cipher. Formally, if there exists

a (t, ϵ, q_D) attacker for EPOC-3 then there exists a (t', ϵ') solver for the gap-factoring problem (see Sect. 6.2.3) with

$$\epsilon' \approx \frac{1}{2}\epsilon - \frac{q_D}{2^\lambda}\,, \tag{6.34}$$

$$t' \approx t + c(q_H + q_K)\,, \tag{6.35}$$

where

- c is a constant,
- q_H is the number of queries the attacker makes to the hash function random oracle,
- and q_K is the number of queries the attacker makes to the key derivation function random oracle.

The reduction of EPOC-3 is not as efficient as that of EPOC-2, and the underlying assumption is not as strong. Furthermore, the submitters withdrew their support from the scheme in favour of EPOC-2.

6.5.3 PSEC-1

PSEC-1 is the first of the PSEC family of cryptosystems that were submitted to NESSIE by the NTT Corporation of Japan. It consists loosely of the following algorithms. A complete specification can be found in [202].

6.5.3.1 The design

Key Generation. PSEC-1 is defined over an elliptic curve, so, before any of these algorithms are executed, it is necessary to have constructed a suitably secure elliptic curve E and chosen a point $P \in E$ with prime order p. We assume that the length of p is equal to the security parameter λ and that the elliptic curve is defined over a finite field F_q, where q is a prime power of length $qLen$. The key generation algorithm is a probabilistic algorithm that takes $(E, P, p, qLen, \lambda)$ as input. It runs as follows.

1. Generate a random integer $s \in (\mathbb{Z}/p\mathbb{Z})^*$.
2. Set $W := sP$.
3. Choose positive integers $mLen$ and $rLen$ such that $mLen + rLen = qLen$.
4. Set $pk := (E, P, p, qLen, W, mLen, rLen, \lambda)$ and $pk := (s, pk)$.
5. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm is a probabilistic algorithm that takes as input a message m of length $mLen$ and the public-key pk. Before the encryption algorithm can be run the communicating parties must have agreed upon the use of some common, public hash function $Hash(\cdot)$. The encryption algorithm runs as follows.

1. Generate a random bit string r of length $rLen$.
2. Set $\alpha := Hash(m||r)$.

3. Set $C_1 := \alpha P$.
4. Set x to be the x-coordinate of αW.
5. Set $C_2 := (m||r) \oplus x$.
6. Output the ciphertext $C = (C_1, C_2)$.

Decryption Algorithm. The decryption algorithm is a deterministic algorithm that takes as input a ciphertext C and the private-key sk. It also uses the agreed hash function $Hash(\cdot)$. It runs as follows.

1. Parse the ciphertext C into (C_1, C_2).
2. Set x to be the x-coordinate of sC_1.
3. Set m and r to be the appropriately sized bit strings that satisfy $(m||r) = C_2 \oplus x$.
4. Set $\alpha := Hash(m||r)$.
5. Check that $C_1 = \alpha P$. If not, output `Invalid Ciphertext` and halt.
6. Output m.

6.5.3.2 Security analysis

The following is a summary of the security analysis of PSEC-1; for more details see [480]. Some of the arguments that justify the selection of PSEC-2 for further study in NESSIE phase II over PSEC-1 can be found in [482].

The security claims of [202] refer to a security proof in [204]. Here it is loosely shown that PSEC-1 is IND-CCA2 secure in the random oracle model provided that the decisional Diffie-Hellman problem (see Sect. 6.2.3) is intractable on the elliptic curve group generated by P. Formally, if there exists a (t, ϵ, q_D) attacker for PSEC-1 then there exists a (t', ϵ') solver for the decisional Diffie-Hellman problem on the elliptic curve group generated by P, with

$$\epsilon' \approx (\epsilon - q_H 2^{-rLen+1})(1-2p)^{q_D}\,, \tag{6.36}$$

$$t' \approx t + q_H(T + c\lambda)\,, \tag{6.37}$$

where

- the attacker makes at most q_H queries to the random oracle simulating the hash function,
- T is the time taken to calculate sQ on the elliptic curve,
- and c is a constant.

PSEC-1 has a very limited message space. This is because it was designed as a key distribution mechanism, but key distribution mechanisms are outside the scope of the NESSIE project. It shares a lot of properties with PSEC-KEM (see Sect. 6.4.5) but reduces to a weaker security assumption. Furthermore, the submitters withdrew their support from PSEC-1 in favour of PSEC-KEM.

6.5.4 PSEC-3

The last of the PSEC algorithms submitted by NTT Corporation, PSEC-3, is a hybrid encryption scheme based on the hardness of the gap Diffie-Hellman problem on certain elliptic curve groups. It consists loosely of the following algorithms. A complete specification can be found in [202].

6.5.4.1 The design

Key Generation. The key generation algorithm for PSEC-3 is similar to that of PSEC-KEM (see Sect. 6.4.5). Since PSEC-3 is based on the Diffie-Hellman problem on elliptic curves, it is necessary to have generated a suitable curve E and chosen a point $P \in E$ with prime order p. We will assume that the length of p is equal to the security parameter λ and that E is defined over a finite field $\mathbb{F}_q$, where q is a prime power of length $qLen$. The key generation algorithm is a probabilistic algorithm that takes $(E, P, p, qLen, \lambda)$ as input. It runs as follows.

1. Generate a random integer $s \in (\mathbb{Z}/p\mathbb{Z})^*$.
2. Set $W := sP$.
3. Set $pk := (E, P, p, qLen, W, \lambda)$ and $sk := (s, pk)$.
4. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm is a probabilistic algorithm that takes a message m and the public-key pk as input. It is necessary for the two communicating parties to have agreed on the use of some common functions: a hash function $Hash(\cdot)$, a key derivation function $KDF(\cdot)$ and a symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It runs as follows.

1. Generate a random integer $r \in (\mathbb{Z}/p\mathbb{Z})^*$.
2. Set $C_1 := rP$.
3. Set x to be equal to the x-coordinate of rW.
4. Generate a random bit string u of length $qLen$.
5. Set $C_2 = u \oplus x$.
6. Derive a suitable symmetric key $K := KDF(u)$.
7. Encrypt the message m using the agreed symmetric encryption scheme under the key K, i.e. set $C_3 = Sym.Encrypt(m, K)$.
8. Set $C_4 = Hash(C_1||C_2||C_3||u||m)$.
9. Output the ciphertext $C = (C_1, C_2, C_3, C_4)$.

Decryption Algorithm. The decryption algorithm is a deterministic algorithm that takes a ciphertext C and the secret-key sk as input. It also uses the pre-agreed hash function $Hash(\cdot)$, key derivation function $KDF(\cdot)$ and symmetric encryption scheme (*Sym.Encrypt*, *Sym.Decrypt*). It runs as follows.

1. Parse the ciphertext C as (C_1, C_2, C_3, C_4).
2. Set x to be the x-coordinate of sC_1.
3. Set $u := C_2 \oplus x$.
4. Derive a suitable symmetric key $K := KDF(u)$.
5. Decrypt the symmetric ciphertext C_3 using the key K, i.e. set $m := Sym.Decrypt(C_3, K)$.
6. Check that $C_4 = Hash(C_1||C_2||C_3||u||m)$. If not, output `Invalid Ciphertext` and halt.
7. Output m.

6.5.4.2 Security analysis

The following is a summary of the security analysis for PSEC-3; for more details see [480]. Some justification for the selection of PSEC-KEM (see Sect. 6.4.5) for further study over PSEC-3 is given in [482].

PSEC-3 is an amalgamation of two techniques used to enhance the security of a key distribution scheme similar to PSEC-1 (see Sect. 6.5.3). The scheme uses standard techniques to transform the weak key distribution scheme into a weak hybrid encryption scheme. It then uses the techniques of [204] to improve the security of the scheme.

The security analysis of PSEC-3 assumes that the implementation uses a Vernam cipher as its symmetric component. The specifications [202] claim that PSEC-3 is IND-CCA2 secure in the random oracle model and reduces to solving the gap Diffie-Hellman problem on the elliptic curve group generated by P. A formal proof of security was never given.

PSEC-KEM has an efficient reduction to a better assumption than PSEC-3, and PSEC-3 does not appear to be significantly faster than a hybrid scheme using PSEC-KEM. PSEC-3 is also very similar in structure to ECIES (see Sect. 6.4.2). Furthermore, the submitters withdrew their support for PSEC-3 in favour of PSEC-KEM.

6.5.5 RSA-OAEP

The RSA-OAEP algorithm is a well-established asymmetric encryption scheme that uses the OAEP padding scheme developed by Bellare and Rogaway [45]. It was submitted to NESSIE by RSA Laboratories and consists loosely of the following algorithms. A complete specification can be found in [274].

6.5.5.1 The design

Key Generation. The key generation algorithm for RSA-OAEP is similar to that of RSA-KEM (see Sect. 6.4.6). It takes the security parameter λ as input and runs as follows.

1. Generate a public exponent e, an odd integer greater than 1.
2. Randomly generate a prime p of length λ such that $gcd(p, e) = 1$.
3. Randomly generate a prime q of length λ such that $gcd(q, e) = 1$.
4. Set $n := pq$.
5. Set d to be the unique integer in $(\mathbb{Z}/\lambda(n)\mathbb{Z})^*$ such that $ed \equiv 1 \bmod \lambda(n)$, where $\lambda(n) = l.c.m.(p-1, q-1)$.
6. Choose positive integers $mLen$ and $rLen$ such that the length of n in bits is equal to $mLen + rLen + 32$.
7. Set $pk := (n, e, mLen, rLen, \lambda)$ and $sk := (d, pk)$.
8. Output the key-pair (pk, sk).

Encryption Algorithm. The encryption algorithm is a probabilistic algorithm that takes a message m of length $mLen$ and the public key pk as input. In order to use the scheme all parties must have agreed on the use of a common, public mask generating function $MGF(\cdot)$. Note that here we use the mask generating function to produce outputs of different lengths. We trust that the size of the output required from the mask generating function will be clear from the context.

The encryption algorithm runs as follows.

1. Generate a random bit string R of length $rLen$.
2. Set $X := MGF(R) \oplus (1_{16}||m)$, where 1_{16} is the 16-bit representation of the integer 1.
3. Set $Y := R \oplus MGF(X)$.
4. Set $Z := 0_{16}||Y||X$ where 0_{16} is the 16-bit representation of the integer 0.
5. Set $C = Z^e \bmod n$.
6. Output the ciphertext C.

Decryption Algorithm. The decryption algorithm is a deterministic algorithm that takes a ciphertext C and the secret-key sk as input. It requires access to the same mask generating function that is used in the encryption process, and runs as follows.

1. Set $Z := C^d \bmod n$.
2. Check that the leftmost 16 bits of Z are equal to 0. If not, output `Invalid Ciphertext` and halt.
3. Parse Z as $0_{16}||Y||X$, where Y has length $rLen$, X has length $mLen + 16$ and 0_{16} is the 16-bit representation of the integer 0.
4. Set $R := Y \oplus MGF(X)$.
5. Set $W := X \oplus MGF(R)$.
6. Check that the leftmost 16 bits of W are equal to the 16-bit representation of the integer 1. If not, output `Invalid Ciphertext` and halt.
7. Parse W as $1_{16}||m$, where m has length $mLen$ and 1_{16} is the 16-bit representation of the integer 1.
8. Output m.

6.5.5.2 Security analysis

The following is a summary of the security analysis of the RSA-OAEP asymmetric encryption scheme; for more details see [350].

The RSA cryptosystem [452] is well known not to be message-indistinguishable in any normal attack model; however it is thought to be OW-CPA secure in the standard model. Indeed its security has been so well studied that it is considered to be a trusted cryptographic problem in its own right and so needs no more justification as a cryptosystem. The OAEP padding method was introduced in [45] and provided with a proof that it transforms a scheme that is OW-CPA secure into a scheme that is IND-CCA2 secure in the random oracle model. Unfortunately this proof was shown to have a flaw in [488]. The proof was corrected for the RSA cryptosystem in [206].

This proof reduces the security of the cryptosystem to the RSA problem. Formally, if (t, ϵ, q_D) is an attacker for RSA-OAEP in the IND-CCA2 model then there exists a (t', ϵ') solver for the RSA problem with

$$\epsilon' \geq \frac{\epsilon^2}{4} - \epsilon \cdot \left(\frac{2q_D q_M + q_D + q_M}{2^{rLen}} + \frac{2q_D}{2^{16}} + \frac{32}{4^{\lambda - rLen}} \right), \tag{6.38}$$

$$t' \geq 2t + 3q_M^2 + O(\lambda^3), \tag{6.39}$$

where

– q_M is the number of queries the attacker makes to the random oracle representing the mask generating function.

Note that this security bound is not tight.

Although side-channel attacks were not considered during phase I, the wealth of literature on RSA based cryptosystems means that they are easy to find. It is vulnerable to an error-message attack [348] and to a Hamming weight attack [292] that recover a hidden message.

At the end of phase I NESSIE invited RSA Laboratories to submit a tweaked version of the RSA cryptosystem with a padding method that gave rise to a more efficient security reduction (such as RSA-SAEP or RSA-SAEP+ [92]). RSA Laboratories responded by saying:

> The only known RSA-based encryption scheme with a better security reduction than RSA-OAEP is Victor Shoup's adaption RSA-OAEP+. However, the security reduction for RSA-OAEP+ is not tight enough either in practice. Since neither of the schemes are provably secure for concrete parameters, replacing one with the other does not appear meaningful.
>
> – J. Jönsson, RSA Laboratories

Hence RSA-OAEP was not selected for further study in NESSIE phase II because it offers no advantages over a hybrid scheme that uses RSA-KEM (see Sect. 6.4.6).

Changes from version 1.0 to version 2.0 of the document

—	Typos have been corrected.
—	Some performance considerations have been moved to D21.
—	Pictures added in §6.4.1, §6.4.4 and §6.4.5.
§6.2.2	Message Indistinguishable vs. one-way expanded.
§6.2.3	Updated trusted problems to be more in line with Ch. 7.
§6.2.3.2	Now includes a section on changing list solutions for CDH into a single solution.
§6.3.2	Improved, added reference to [162].
§6.3.3	Added a section on KDFs.
§6.4.5	Updated PSEC-KEM, including a reference to [415].

7. Digital signature schemes

7.1 Introduction

What is a digital signature? A signature is used by a signer to authenticate a document, in such a way that anyone can check the validity of the signature. A digital signature scheme does not mimic exactly the classical handwritten signatures, because the signature is different for each message. If this were not the case, a signature could be extracted from a document and copied to another document.

The digital information that allows the signer to generate valid digital signatures is the private key, and the digital information that allows the verifier to check the validity of the signature is the public key.

Components of a digital signature scheme. A signature scheme is described by the following four algorithms, with the security parameter k:

- a parameter generation $\mathsf{Generate} : (k, \rho) \mapsto \mathsf{param}$,
- a key generation $\mathsf{KeyGen} : (\mathsf{param}, \rho') \mapsto (\mathsf{pk}, \mathsf{sk})$,
- a signature generation $\mathsf{Sign} : (\mathsf{param}, \mathsf{sk}, m, r) \mapsto \sigma$,
- a signature verification $\mathsf{Ver} : (\mathsf{param}, \mathsf{pk}, \sigma, r') \mapsto m$ or reject.

All these algorithms are deterministic. pk is the public key, sk is the private key, m is a message and σ a signed message. The inputs ρ, ρ', r and r' (if non empty) contain the optional randomisation for the algorithms. Usually these are fixed length bit strings.

The digital signature scheme is sound if the following condition holds.

- For all k and m and for all ρ, ρ', r and r', let $\mathsf{param} = \mathsf{Generate}^{\rho}(k)$ and $(\mathsf{pk}, \mathsf{sk}) = \mathsf{KeyGen}^{\rho'}(\mathsf{param})$, and let $\sigma = \mathsf{Sign}^{r}_{\mathsf{param},\mathsf{sk}}(m)$. Then $\mathsf{Ver}^{r'}_{\mathsf{param},\mathsf{pk}}(\sigma) = m$.
  It may be accepted that the scheme is sound either with probability 1 or for all but a negligible proportion of ρ, ρ', r and r'.

The digital signature scheme is secure if it is hard to generate a valid σ without knowing sk. When studying the security of a digital signature scheme, the exact description of the algorithms $\mathsf{Generate}$, KeyGen and Sign is not needed. Only the probability distribution of their output makes a difference (see also the notion of

[0] Coordinator for this chapter: ENS — Louis Granboulan

equivalent signature schemes in Sect. 7.2.3). This is why we will begin the description of signature schemes by the description of their most specific component: Ver.

Appendix or message recovery. Signature schemes with appendix have the property that $\sigma = m \| s$ and s is called the appendix. Signature schemes with partial message recovery have the property that $\sigma = \hat{m} \| s$ and that the whole message is $m = \hat{m} \| \bar{m}$, where $\bar{m}$ is the recovered part of the message. They typically have a lower bound for the size of the whole message, which is also the number of message bits recovered. This lower bound can be overcome by storing the length of the actual recovered part in $\bar{m}$, e.g. by padding $\bar{m}$ with a 1 followed by a string of 0s. With this padding, one bit of message expansion is added. All the signature schemes submitted to NESSIE are signature schemes with appendix.

Stateful schemes. In a stateful digital signature scheme, the signing algorithm Sign is allowed to keep an internal state that tracks the number of signatures generated and optionally their values.

All the signature schemes submitted to NESSIE are stateless, but there exist interesting examples of stateful schemes [229, 178, 140].

Public key and identity. It is important that the verifier is convinced that the public key used to verify a signature corresponds to the alleged signer. Generally, it is not the goal of the signature scheme to decide how the public key infrastructure is organised, but any application that uses a signature scheme needs to decide how public keys are linked to meaningful identities.

Some digital signature schemes are identity-based, which means that the signer can choose his public key to be equal to his identity, and that a trusted authority generates the corresponding secret key. No identity-based signature scheme was submitted to NESSIE.

Description of the scheme, public parameters and public keys. Many descriptions of signature schemes only include the precise description for the algorithms Sign and Ver. However, to study the performance and security of the scheme one needs to know what is part of the scheme, what is a parameter that can be tuned to different applications, and what is in the keys specific to each user.

In many cases, parameters and keys are not clearly separated, and parameter generation and key generation are merged in a single algorithm $\mathsf{KeyGen}' : (k, \rho \| \rho') \mapsto (\mathsf{param} \| \mathsf{pk}, \mathsf{param} \| \mathsf{sk})$, but separating the two is useful in practice.

The public parameters of a scheme usually include some information on the key size, but may include other additional information like the description of an elliptic curve subgroup. The choice of a hash function for the scheme may be fixed, but is usually left as a parameter, or may even be left up to the user (see also the discussion on hash identifiers [281]).

Parameters and key validation. Many descriptions of signature schemes only include the algorithms Sign and Ver. They certainly are the most specific components of a scheme, and their performance and security are usually studied assuming that the parameters and keys are uniformly randomly distributed. However,

the way parameters and keys are actually generated can be the source of weaknesses and it is important that the verifier and the signer can trust the parameters and the keys. This document will not go deeply into this topic, but this issue will be highlighted when necessary.

Short to long term security. Some applications may need long term security (50 to 80 years) but current knowledge does not allow us to make predictions about cryptanalytic advances for such a long time. Some applications only need short term validity of a signature (1 day to 1 month). Medium term security is 5 to 10 years and is the main target of this document.

All signature schemes should incorporate in their public parameters a deadline for validity. If the validity needs to be extended beyond this deadline, re-signing with more recent public parameters is mandatory.

7.2 Security requirements

7.2.1 Security model

7.2.1.1 Existential unforgeability under adaptive chosen message attack

Unforgeability. An *existential forger under adaptive chosen message attack* is a (randomised) algorithm that inputs a public key and tries to produce a valid signature, the forgery. The forger is able to make queries to a black box that generates valid signatures and the forgery must be new (see below). A (t, ε, q_S)-forger succeeds in time t with probability ε and is allowed to make q_S signing queries. A signature scheme with no (t, ε, q_S)-forger is said to be (t, ε, q_S)-secure.

The original definition of an existential forger [229] required that the forgery is a valid signed message for a message that was not the input of a signature query. Our definition only requires that the forgery is a valid signed message that was not the answer to a query. This means that another valid appendix for the same message is a valid forgery. This requirement is called "super-security" by Goldreich [222, Volume 2, Sect. 6.5.2], "strong unforgeability" by Bellare and Namprempre (introduced for MACs [42]) or "non-malleability" by Stern *et al.* [497].

We will aim at non-malleable existential unforgeability under adaptive chosen message attack. Non-malleability may not be really important [10] but some applications may need it.

Some weaker security requirements. The adaptive chosen message signing oracle can be replaced by a less powerful oracle.

– Single-occurrence chosen message attacks (SO-CMA) [497]. The forger is not allowed to make multiple signing queries on the same message. This attack model appears during the study of non-deterministic signature schemes, especially ESIGN. See Sect. 7.3.1.2 and 7.4.2.

– Random message attacks (RAND). The forger has access to a list of signed messages that correspond to random messages. This attack model depends on a probability distribution on the message space. Schemes that are secure in this attack model can be used as building blocks for schemes that are secure against adaptive chosen message attacks. See Sect. 7.3.3.3.
Goldwasser, Micali and Rivest [229] defined *known-message attacks* where the forger has access to a list of signed messages, but they don't say how these messages are chosen, only that the forger did not choose them.

Security level. The security level of a scheme is k bits if there exists no (t, ε, q_S)-forger with $\log_2(t/\varepsilon) < k$.

This value k depends on the time unit used for t. A natural time unit is the running time of the verification algorithm.

Non-repudiation of origin. In a similar way to the two flavours of unforgeability, two flavours of non-repudiation of origin exist.

Basic non-repudiation of origin means that any third party can be convinced that a valid signed document corresponds to a message that has been deliberately signed by the secret key holder. It makes it impossible to repudiate a message. Existential unforgeability under adaptive chosen message attack implies (basic) non-repudiation.

Strong non-repudiation of origin means that any third party can be convinced that a valid signed document has been deliberately signed by the secret key holder. It makes it impossible to repudiate a signed message. Non-malleable (strong) existential unforgeability under adaptive chosen message attack implies strong non-repudiation.

> REMARK: The verification algorithm needs to get the whole signed message σ to compute its answer. Therefore, for a signature scheme with appendix, it is meaningless to consider properties where the appendix is studied separately from the message.
> For example, the notion of duplicate signatures is meaningless. It was defined [497] as an attack against an appropriate definition of non-repudiation. Duplicate signatures are values m, m', s such that $\mathsf{Ver}^{r'}_{\mathsf{param},\mathsf{pk}}(m\|s) = m$ and $\mathsf{Ver}^{r'}_{\mathsf{param},\mathsf{pk}}(m'\|s) = m'$. This property also appeared in [105] where it was seen as a potential weakness.
> While the existence of duplicate signatures may be surprising, it is not a weakness of a digital signature scheme. If the scheme is existentially unforgeable, it proves that both signed messages were produced by a secret key holder.
> Moreover, one can notice that the secret key can easily be deduced from the duplicate signatures described in [105, 497]. Therefore a user that shows duplicate signatures in these schemes is a user that publishes his secret key.

It is important that the secret key is not compromised, because any holder of the secret key can sign documents. Therefore, non-repudiation of origin is not a proof of security against viruses or Trojan horses. Forward secrecy and related properties [11, 41, 314, 3, 264, 345] deal with this problem. Such properties are not in the scope of the NESSIE evaluation and we make the hypothesis that the secret key is never compromised.

7.2.1.2 Other security models

While existential unforgeability under adaptive chosen message attack is the *de facto* standard notion of security for signature schemes, it does not cover all possible attacks. This security notion only considers a unique randomly generated param and a unique $(\mathsf{pk}, \mathsf{sk})$ pair. Non-uniqueness or non-random generation lead to realistic attack models that are not covered by existential unforgeability under adaptive chosen message attack.

Authentication of origin. An attacker against this property of a digital signature scheme is an algorithm that inputs a signed message σ valid for a public key pk and outputs another public key pk' such that σ is valid for pk'.

This property is not implied by existential unforgeability, but it is useful to mimic some properties of hand-written signatures, e.g. in applications where some write-only and time-stamped database is used to receive signed messages and then to solve disputes for anteriority.

The attack against the property of authentication of origin is named *duplicate-signature key selection* [83] or *key substitution* [365] or *key-collision* [454].

Multi-key/multi-user setting. These settings consider the case where a fixed param and many pk_i are used, and the adversary has access to signature oracles for all those keys.

The multi-user setting for digital signature schemes first appeared in [365, 211] and its security requirements are related to those of the multi-user setting for asymmetric encryption schemes [35].

By definition, a (t, ε, n, q_S)-mu-forger has access to the signature oracles corresponding to n public keys pk_i, is allowed to make at most q_S queries to the oracles, runs in time t and outputs a forgery for some pk_i with probability ε. A scheme for which the existence of a (t, ε, n, q_S)-mu-forger implies the existence of a (t, ε, q_S)-forger is *secure in the multi-user setting*. This also implies authentication of origin.

Using similar techniques to those of Bellare *et al.* for multi-user security with asymmetric encryption, Galbraith *et al.* [211] showed how random self-reducibility of Schnorr-like signature schemes proves multi-user security.

A similar requirement appeared in [230] under the name *multi-key*. Here the best simultaneous attack on all the keys should be an independent attack on each key.

By definition, a (t, ε, n, q_S)-mk-forger has access to the signature oracles corresponding to n public keys pk_i, is allowed to make at most q_S queries to each oracle, runs in time nt and outputs a list of forgeries for each pk_i such that each forgery is valid with probability ε. A scheme for which the existence of a (t, ε, n, q_S)-mk-forger implies the existence of a (t, ε, q_S)-forger is *secure in the multi-key setting*.

This is the case if all the verification algorithms with distinct $(\mathsf{param}, \mathsf{pk})$ values are information-theoretically independent. However, most digital signature schemes share a common hash function for all public keys. Finding a collision in the hash function is a more efficient simultaneous attack on multiple keys than on

a single key. Therefore it is good practice to have a key-dependent hash function, for example by including param and pk in the input of a common hash function.

Parameter manipulation. If param is not chosen at random, new attacks may appear. For example, param can be chosen with a trapdoor enabling the generation of valid signatures without knowing the secret key [85] or such that a collision is introduced in the hash function [510, 369].

One technique that protects against parameter manipulation is parameter validation, e.g. the publication of (k, ρ) together with param, but this might not be sufficient if the algorithm that generates the parameters from the seed leaves some freedom, as is the case for the ECDSA standardised technique [513]. Ideally, one should provide a security proof for Generate. Another heuristic protection has been proposed [285, 369] to protect against an attacker that studies the properties of a hash function used in the scheme to select weak or trapdoor parameters: one can include the values param, pk in the input to this hash function.

Side-channel attacks. A hidden assumption of our security model is that the attacker may have access to the input and output of the signing algorithm, but the attacker should not be able to get any information about intermediate values that appear during the computation of a signature.

An adversary that has access to this type of information is said to be able to mount a side-channel attack (see Appendix A for more details).

Key recovery vs. forgery. If one access to a forger allows an attacker to compute the secret key, then additional forgeries can be made without the forger and the message for these forgeries can be chosen by the attacker. For example, if a side-channel attack can succeed in producing a forgery, then a few accesses to side-channel information can allow an attacker to make an unlimited number of chosen-text-forgeries. Therefore, the existence of a reduction from forgery to key recovery can be seen as a weakness of the system.

7.2.2 Intractability assumptions

7.2.2.1 Mathematical problems

A *mathematical problem* is described by the set of instances of size l, a probability distribution on this set, and a set of possible solutions. For each instance, some of the possible solutions are valid and the others invalid.

Computational and decisional problems. If the set of possible solutions is {yes, no} with one valid and one invalid for each instance, then it is called a decisional problem; else it is called a computational (or search) problem.

A *solution-checker* is an efficient algorithm that inputs an instance and a solution and tells if the solution is valid. The problem of the existence of a solution-checker is the decisional problem associated with the problem.

Concrete intractability. A *solver* for the problem is an efficient algorithm that inputs an instance and outputs a valid solution. A mathematical problem is (t', ε')-intractable for size l if there exists no solver running in time t' that succeeds with probability better than ε' for a random instance of size l. The problem has intractability k' bits if there exists no (t', ε')-solver with $\log_2(t'/\varepsilon') < k'$. The value k' depends on the time unit.

Asymptotic intractability. A problem is intractable if for any polynomial p the problem with size l has intractability $p(l)$ bits for large enough l. With the terminology of complexity theory an intractable problem (usually only defined for decisional problems) is a problem that is not a member of P.

NP-hard problems are proven to be at least as difficult to solve as all NP problems (problems that have a polynomial-time solution-checker). They may be good candidates for intractable problems. However NP-hardness only gives a bound for the worst case and not for an average instance of the problem, and hard instances may be difficult to generate.

7.2.2.2 Trusted cryptographic problems

No mathematical problem is provably intractable, but the following problems are good candidates and their intractability is assumed when proving the security of some cryptographic schemes.

Factorisation-based problems.

- **The integer factorisation problem.** An instance is a composite integer n of l bits. A solution is a non-trivial factor of n.
  The probability distribution of the instances is unclear. Usually, the integer n is constructed as pq or p^2q where p and q are randomly generated primes of similar sizes, because these are the most difficult instances for the current best factorisation algorithms. [1]
  The corresponding decisional problem is easy because the algorithm that computes the gcd of the instance and the solution is a solution-checker.
- **The RSA problem.** An instance is (n, e, y) where n is a composite integer of l bits, e is an integer coprime to $\phi(n)$ and $y \in (\mathbb{Z}/n\mathbb{Z})^\times$, the set of invertible elements modulo n. A solution is some $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ such that $y = x^e$.
  The probability distribution of the instances is as unclear as for the factorisation problem. There exist two techniques for generating random instances: choosing n, then e and then y, or choosing e, then n and then y. The second technique is usually preferred because it allows one to fix the value of e in advance and use the e-th root problem below.
  The only known method of solving this problem is to solve the integer factorisation problem for n. The assumption that the RSA problem is hard is known as the RSA assumption.

[1] There is no simple method to detect if n is of one of these special forms. In general, the best known technique to detect if n is of one of these forms is to factor it.
Many algorithms exist for the generation of random primes and give different probability distributions. One efficient algorithm has been submitted to NESSIE [279].

The corresponding decisional problem is easy because the algorithm that computes $x^e \stackrel{?}{=} y$ is a solution-checker.

- **The e-th root problem.** An instance with parameter e is (n, y) where n is a composite integer of l bits such that $\phi(n)$ is coprime to e and $y \in (\mathbb{Z}/n\mathbb{Z})^\times$. A solution is some $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ such that $y = x^e$.
  This problem looks very similar to the RSA problem, but having a fixed value for e leads to some theoretical results, e.g. if e is a smooth integer [98]. One can also notice that if e is not coprime to $\phi(n)$, e.g. $e = 2$, then the e-th root problem (where y is a e-th power) is proven to be equivalent to the factorisation problem.
  The only known method of solving the e-th root problem is to solve the integer factorisation problem for n.
  The corresponding decisional problem is easy because the algorithm that computes $x^e \stackrel{?}{=} y$ is a solution-checker.
- **The flexible RSA problem.** An instance is (n, y) where n is a composite integer of l bits and $y \in (\mathbb{Z}/n\mathbb{Z})^\times$. A solution is some $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ and $e > 1$ such that $y = x^e$.
  The only known method of solving this problem is to solve the integer factorisation problem for n. The assumption that this problem is hard is known as the *strong RSA assumption*.
  The corresponding decisional problem is easy because the algorithm that computes $x^e \stackrel{?}{=} y$ is a solution-checker.
- **The approximate e-th root problem (AER).** An instance with parameters e and d is (n, y) where n is a composite integer of l bits such that $\phi(n)$ is coprime to e and $y \in (\mathbb{Z}/n\mathbb{Z})^\times$. Let $\alpha_d(x) = \left\lfloor \frac{x \bmod n}{n^{(d-1)/d}} \right\rfloor$ the $n^{\frac{1}{d}}$-approximation of x. A solution is some $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ such that $\alpha_d(y) = \alpha_d(x^e)$.
  For $d = 3$, this problem has been solved for $e < 4$. The only known method of solving this problem for $d = 3$ and $e \geq 4$ is to solve the integer factorisation problem for n.
  The corresponding decisional problem is easy because the algorithm that computes $\alpha_d(x^e) \stackrel{?}{=} \alpha_d(y)$ is a solution-checker.
- **The claw-free approximate e-th root problem (Claw-AER).** Parameters and instances are defined as for AER but a solution is a pair $x, z \in (\mathbb{Z}/n\mathbb{Z})^\times$ such that $\alpha_d(yz^e) = \alpha_d(x^e)$. This problem appeared recently [230] and its intractability is not well known.
- **The second-preimage approximate e-th root problem (2nd-AER).** Parameters and instances are defined as for AER but a solution is a value $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ such that $x \neq y$ and $\alpha_d(x^e) = \alpha_d(y^e)$. This problem is related to the non-malleability of ESIGN and its intractability is not well known.

Discrete logarithm-based problems. All problems are parameterised by a cyclic group $\langle G \rangle$ and its generator G. The order of $\langle G \rangle$ is a number q with l bits. Group operations should be easy to compute, but elements of this group may be indistinguishable from elements of a larger group.

$\langle G \rangle$ is usually a subgroup of the multiplicative group of a finite field of prime order p or a subgroup of the group of points on an elliptic curve over a finite field.

- **The discrete logarithm problem.** An instance is H, a random element in $\langle G \rangle$. The solution is $x \in \mathbb{Z}/q\mathbb{Z}$ such that $G^x = H$.
  This value is unique and is the discrete logarithm $\log H$.
  The corresponding decisional problem is easy because the algorithm that computes $G^x \stackrel{?}{=} H$ is a solution-checker.
- **The computational Diffie-Hellman problem.** An instance is (H_1, H_2), both random elements of $\langle G \rangle$. The solution is H such that $\log H = \log H_1 \cdot \log H_2$.
  The only known method for solving this problem is to solve the discrete logarithm problem.
- **The decisional Diffie-Hellman problem.** An instance is (H, H_1, H_2), elements of $\langle G \rangle$. The solution is yes if $\log H = \log H_1 \cdot \log H_2$.
  The probability distribution of instances is given by the following algorithm: take independent random $x_1, x_2, x_3 \in \mathbb{Z}/q\mathbb{Z}$ and a random bit b, let $H_1 = G^{x_1}$, $H_2 = G^{x_2}$ and, depending on b, either $H = G^{x_3}$ or $H = G^{x_1 x_2}$.
  The only known generic method for solving this problem is to solve the computational Diffie-Hellman problem, hence to solve the discrete logarithm problem. However, there exist groups where an efficient algorithm solves the decisional Diffie-Hellman problem without solving the computational Diffie-Hellman problem (they are called GDH groups and are elliptic curves where the Tate/Weil pairing [199, 277] has special properties).
- **The gap Diffie-Hellman problem.** Solve the computational Diffie-Hellman problem with access to an oracle that answers the decisional Diffie-Hellman problem. In practice this problem appears in GDH groups.

Multivariate algebra-based problems.

- **The MQ problem.** The MQ problem is to find a solution to a given set of multivariate quadratic equations over a finite field, and is NP-hard in general.
- **The HFE problem and variants.** The HFE problem is a special case of the MQ problem where the set of equations is not random but constructed so that there is a trapdoor to their solution, and the HFEv$^-$ problem is an extension of the HFE problem which is harder to solve.
  The QUARTZ, FLASH and SFLASH problems are special cases of HFEv$^-$.

7.2.2.3 How to estimate concrete intractability

Best known solvers. There are some algorithms for solving the above problems. The notation $L_q[\alpha, c] = \mathcal{O}(\exp((c+o(1))(\ln q)^\alpha (\ln \ln q)^{1-\alpha}))$ is used for the asymptotic complexity of some of them.

- **Integer factorisation.** The fastest known algorithms for factorising large integers are the Number Field Sieve [331] and the Elliptic Curve Method [335]. The asymptotic time taken by the number field sieve to factor an integer n is approximately $L_n[\frac{1}{3}, c_{\mathrm{NFS}}]$ where c_{NFS} is a constant depending on the variant

of number field sieve. [2] The asymptotic time taken by the elliptic curve method to factor an integer whose smallest factor is p is $L_p[\frac{1}{2}, \sqrt{2}]$. Both algorithms are subexponential in the size of their input.
An improvement of the elliptic curve method exists for $n = p^2q$ [430, 179] and a special algorithm for $n = p^rq$ with large r [97].

- **Discrete logarithm over $\mathbb{F}_p$.** The index-calculus method [122, 411, 186] is the fastest known method of solving the discrete logarithm problem over $\mathbb{F}_p$. It is closely related to the number field sieve factoring algorithm and has expected asymptotic running time of $L_p[\frac{1}{3}, c_{\mathrm{NFS}}]$, which is subexponential.
- **Elliptic curve discrete logarithm.** The fastest general methods of attack for solving the elliptic curve discrete logarithm problem are the Pollard ρ and the Pollard λ methods [436]. For a group with q elements, the Pollard ρ runs in time $\sqrt{\pi q/2}$, and the Pollard λ runs in time $2\sqrt{q}$ but can be faster in some special cases. Both can be efficiently parallelised [505] and have been slightly improved [212, 522]. No subexponential algorithm has been found for solving the elliptic curve discrete logarithm problem.
  There exist subexponential attacks for specific elliptic curves: supersingular and similar curves [364, 200, 238, 456] and anomalous curves [470, 461, 492].
- **Generic group discrete logarithm.** Nechaev [395] and Shoup [486] proved that the best algorithm to solve the discrete logarithm in a generic group runs in time $\mathcal{O}(\sqrt{q})$. However, all known concrete groups can easily be distinguished from a generic group (they have automorphisms that are easy to compute from the binary representation of the elements).
- **MQ and related problems.** For the MQ and HFE problems the situation is somewhat more complicated and not as well studied. The MQ problem is usually solved by looking for a Gröbner basis — a method that can handle generic multivariate equations. Faugère [190] showed that the instances of HFE generated for QUARTZ are easier than generic instances and Courtois, Daum and Felke [132] studied the implications of this result.
  The XL and FXL algorithms are designed for systems where the equations are quadratic and there is an attack of Shamir and Kipnis [472] on the HFE problem.

Quantum computers. Today large quantum computers don't exist, and they may never exist, but their theoretical aspects have been studied and many researchers are trying to build one. The largest quantum computer that has been built handled 7 q-bits. If larger quantum computers can be realised, their impact on cryptology is important, because some algorithms have been designed that can efficiently solve the integer factorisation or discrete logarithm problem with a quantum computer [485]. No algorithm is known that solves MQ and related problems faster with a quantum computer than with current computers.

[2] The General Number Field Sieve works for any integer n and has $c_{\mathrm{NFS}} = (\frac{64}{9})^{1/3} \simeq 1.923$. The Special Number Field Sieve works for $n = k \cdot a^b \pm c$ and has $c_{\mathrm{NFS}} = (\frac{32}{9})^{1/3} \simeq 1.526$.

Estimations for the difficulty of problems. While it is impossible to be sure that a cryptographic problem will remain intractable (because a fast polynomial algorithm might be discovered, that solves all NP problems), it is necessary to estimate the lifespan of a public key. Articles giving such estimates have been written by many researchers [410, 521, 491, 334] and have led to various conclusions. See also the discussion in [479].

The simplest presentation of their results can be given in terms of equivalent symmetric key size for a given security. The NESSIE call [396] asked for a security equivalent to an 80-bit symmetric key, by asking that the best attacks require the equivalent of 2^{80} Triple-DES operations.

- **Current practice.** For normal security most products use DES (56-bit symmetric key), 512-bit RSA moduli, 112-bit elliptic curve keys and 160/512-bit DSA. Higher security is obtained with Triple-DES (112 bits) or AES (128 bits), 1024-bit RSA and 160-bit elliptic curve keys.
- **Estimates by Certicom.** The Standards for Efficient Cryptography Group [115, appendix B.2] gives the following estimates for comparable key sizes.

Equivalent symmetric key size	56	80	112	128	192	256
RSA modulus length	512	1024	2048	3072	7680	15360
Elliptic curve key size	112	160	224	256	384	512

- **Estimate of Silverman.** A cost-based analysis [491] gives the following table.

Equivalent symmetric key size	56	64	80	96	112	128
RSA modulus length	430	530	760	1020	1340	1620

  With these estimates, the computing power for the factorisation of 1020-bit integers should not be available during the next 20 years, and a 768-bit integer should be factorised by public effort around 2019.
- **Estimates for the computing power available.** Blaze *et al.* [84] estimated in 1996 that a minimum of 75 bits was necessary to have security for commercial use, and that 90 bits were needed to protect data for the next 20 years.
- **Estimates by Lenstra and Verheul.** Their article [334] is the most complete study of this topic. They take many factors into account:
  - Trusted key length for secure block ciphers.
  - Increase of computing power and memory available on constant-cost computers.
  - Increase of the budget of attackers.
  - Cryptanalytic advances.

  Their results can be summarised with the following approximate table.

Equivalent symmetric key size	56	64	72	80	90	100
Elliptic curve key size	105	120	135	160	185	220
RSA modulus length	417	682	1024	1500	2236	3100
RSA cost-based equiv.	288	480	768	1150	1792	2600

  They also find equivalent dates for different key sizes if one makes the hypothesis that DES could be trusted until 1982.

Equivalent symmetric key size	56	64	72	80	90	100
Last year with trust	1982	1992	2002	2012	2025	2040

– **Bernstein's circuit for integer factorisation.** Recently, Bernstein [47] proposed a new hardware design for the linear algebra step of the Number Field Sieve that might reduce the cost of factorisation. Dedicated hardware for the sieving is also studied. The proposed measure of the efficiency of such techniques is "construction cost × run time", which is quite different from the classical count of the number of operations. This approach has been heavily discussed within the community of researchers in cryptography and algorithmic number theory [464, 332], but currently no real consensus emerges. It is likely that dedicated hardware will improve the cost of factorisation, but several estimations give different values, all being asymptotic. However, one can remark that Bernstein's techniques don't reduce the number of operations.
– **Some records.** Here is a table of some historical factorisation records.

Year	1978	1982	1986	1992	1999	2002
Bit size	150	170	289	429	512	525

Exhaustive key search for 64 bits was achieved in 2002 with two years of computation on the spare time of thousands of computers [169]. Factorisation of a 525-bit number was achieved in 2002 with two months of computation on a few dozens of computers [25].
– **A conclusion.** To comply with the NESSIE requirement of an equivalence with a symmetric key of 80 bits, the size given in the above papers for an elliptic curve key is 160 bits, but the sizes for an RSA modulus range from 760 to 1500 bits. All submitted schemes use the intermediate value of 1024 bits.
However, the NESSIE call was phrased in terms of number of operations, because it was felt that cost-based analysis introduces an additional parameter that is not very well understood. Moreover, the difference between attack techniques against symmetric and asymmetric schemes is already taken into account by the fact that the minimal computational cost of an attack against NESSIE-recommended symmetric primitives is 2^{128} operations while the minimal computational cost of an attack against NESSIE-recommended asymmetric primitives is 2^{80} operations.
Therefore, we use the following table, based on the hypothesis that the factorisation of 512-bit numbers with NFS needs a workfactor of 2^{56} and that 190-bit factors are found by ECM with a workfactor of 2^{56}.

Equivalent symmetric key size	56	64	80	112	128	160
Elliptic curve key size	112	128	160	224	256	320
Modulus length (pq)	512	768	1536	4096	6000	10000
Modulus length (p^2q)	570	800	1536	4096	6000	10000

7.2.3 Proven security

A proof of security is the description of a (randomised) algorithm called a *reduction algorithm*. This algorithm is a (t', ε')-solver for some mathematical problem and interacts with a (t, ε, q_S)-forger for the signature scheme. For a random instance of the problem the reduction algorithm sends a random-looking public key to the forger and uses the forgery to solve the problem. Therefore $\varepsilon' \leq \varepsilon$ and $t' \geq t$.

Usually $q_S \ll t$, and we will require $k = \log_2(t/\varepsilon) = 80$ (the attacker's computing power is estimated to 2^{80} triple-DES operations) and $\log_2 q_S = 30$ (the attacker cannot require more than a billion signed messages).

Simulation of signature queries. The black box that is used by the forger to get the q_S signatures depends on the public key. Therefore it is provided by the reduction algorithm, which acts as an oracle. For each signature query made by the forger, the reduction algorithm should answer with a valid signed message, and this answer should be indistinguishable from an answer made by the signing algorithm with the secret key. Therefore the reduction algorithm is also named a *simulator*.

The reduction does not always succeed, partly because its simulation of an equivalent signature scheme may not be perfect, and partly because the forgery may be useless.

Uniformity of proofs. Uniform reduction means that the reduction algorithm does not depend on the description of the forger. Non-uniform reduction means that for any possible forger, there exists a reduction algorithm which is a solver. An example of non-uniform reduction can be found in [435] where the reduction algorithm depends on the success probability and number of queries of the forger.

Efficiency of proofs. We have a *tight* proof of security if $t'/\varepsilon' \simeq t/\varepsilon$.

We have a *not so tight* proof of security if $t'/\varepsilon' \simeq q_S t/\varepsilon$.

We have a *loose* proof of security if $t'/\varepsilon' \gg q_S t/\varepsilon$, for example $t'/\varepsilon' \simeq t^2/\varepsilon$.

For example, for the security proof to give 80 bits of security, factorisation-based schemes should use a 1536-bit modulus if they have a tight proof of security, a 4096-bit modulus with a not so tight proof and a 10000-bit modulus with a loose proof.

Equivalent signature schemes. Two signature schemes are equivalent when the following conditions are satisfied:

- The possible values of param are the same.
- The distributions of the pk generated are indistinguishable.
- Both verification algorithms are the same.
- The output of the two Sign algorithms for fixed m and random r are indistinguishable. When this last condition is omitted, it is a weak equivalence.

Any (t, ε, q_S)-forger for a signature scheme is also a (t, ε, q_S)-forger for all equivalent signature schemes. If $q_S \leq 1$, weak equivalence is sufficient.

A security proof for a signature scheme shows that if the mathematical problem is intractable, then there exists no forger for any scheme equivalent to it.

7.2.4 Proofs in an idealised world

Definition. A proof in an idealised world involves a restriction of the power of the attacker: some components of the verification algorithm cannot be computed by the forger alone, help from the simulator is needed. By definition all our algorithms are deterministic, hence the components are deterministic functions of their inputs.

Basic oracle property. An idealised component has the basic oracle property if the simulator knows the values of all inputs and outputs of this component. Usually the basic oracle property is introduced in the proof by requesting that for each access to this component, the attacker must send to the simulator a query containing the input. Then the simulator computes the output of the component and sends it to the attacker. The simulator behaves like an oracle for this function, hence the name.

Usually the number of queries to the oracle is bounded by q_O, and because the actual computation of the idealised components takes time, a scheme with k bits of security and with the appropriate time unit always has $q_O \leq 2^k$.

Randomness. Instead of computing the idealised component, the simulator may give random answers, provided that they agree with the properties of the component.

The simulator needs to remember all the inputs that are queried and the answers that are made, and make the appropriate consistency checks before answering. In other words, the reduction algorithm constructs the oracle input/output table in answer to queries.

Such an idealised proof makes the assumption that the component has no other useful property than the ones used for the consistency checks.

Programmability. Instead of choosing the answer at random from the set of consistent answers, the simulator is allowed to generate the answer with any technique that is indistinguishable from a random choice.

Example of idealised components and some terminology.

- **Random oracle model.** The idealised component is a hash function. The only consistency check is that two identical inputs get the same answer, hence the name [44]. This is the simplest possible consistency check and it was the first idealised model used for security proofs [193]. The alternative **generic hash model** terminology has been proposed [108].
  The random oracle model is the most widely used model for security proofs in an idealised world [44, 45, 46, 127].
- **Generic group model.** The idealised component is the group where some computations take place. The consistency check has to make sure that algebraic properties of the group operation are respected.
  The generic group model appeared in [395, 486] and has been used to prove the security of some specific schemes [106, 108].
- **Random permutation model.** The idealised component is a permutation, and oracle queries can be made for the permutation or for the inverse permutation. Consistency checks make sure that it is 1-to-1.
  This model is used to prove the security of some specific schemes [232].
- **Ideal cipher model.** The idealised component is a block cipher. This is a simultaneous use of multiple idealised permutations, one for each value of the key.
  This model is used to prove the security of some specific schemes [272, 232].

Can we trust security proofs in an idealised world? Proofs in these models cannot generically be translated into the real world [114, 194, 161], but it is widely believed that a proof in an idealised model gives some confidence in the design of a cryptographic primitive.

The impossibility result of Canetti, Goldreich and Halevi [114] shows that there exist systems that are secure in an idealised model, but insecure when the idealised component is replaced by *any* concrete component. Such a weakness is very unlikely to be present in a simple cryptographic design, but cannot be ruled out. The only solution is to find a proof in the real world.

Another problem with proofs in an idealised world is that they don't specify which concrete attacks against the idealised component should be prevented. The availability of different proofs that idealise distinct components gives a partial answer to this problem.

7.2.5 Assessment process

The digital signature submissions were assessed with reference to the submitted security proof. The underlying intractability assumption was reviewed.

Variants of the scheme were studied to verify whether the designers have made the optimal choices or not. When interesting variants were found, we interacted with the submitters to find justifications for keeping the submitted design unchanged. Compatibility with existing *de facto* standards was not received as a good argument for not improving a scheme, because the goal of NESSIE is not to recommend the *de facto* standards, but to recommend a portfolio of secure and efficient primitives.

7.3 Overview of the common designs

We describe three types of design commonly used to build digital signature schemes.

- The idea of using a trapdoor one-way function to obtain digital signatures dates back to 1978 [452]. This paradigm is also named hash-then-invert or hash-then-sign and NESSIE submissions RSA-PSS, ESIGN, Quartz, Flash and Sflash fall into this category.
- The discrete logarithm problem was the first basis for security in public key cryptography [168] and the first digital signature scheme based on this problem was introduced in 1985 [184]. A wide family of other schemes based on the discrete logarithm problem has been developed and the NESSIE submission ECDSA is one of them.
- The above schemes don't have a security proof without an idealised model. There exist many schemes that have a security proof in the "real world" and the NESSIE submission ACE-Sign is one of those.

This section contains much technical information about the design criteria for digital signature schemes, and the key ideas of the security proofs that support

them. Much of the material in this section appeared in various places, but no previous overview of common designs for digital signature schemes covers all the schemes submitted to NESSIE.

7.3.1 Schemes based on trapdoor one-way functions

7.3.1.1 Properties of one-way functions

Family of one-way functions. A family of one-way functions is a collection of functions $\{\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i | i \in \mathcal{I}\}$ over some index set $\mathcal{I} = \bigoplus_l \mathcal{I}_l$ (disjoint union) with the following properties.

- OW1 There is an efficient randomised algorithm Gen which takes as input a length parameter l and outputs an index $i \in \mathcal{I}_l$.
- OW2 There is an efficient randomised algorithm which on input i outputs $x \in \mathcal{S}_i$. The resulting probability distribution is denoted $x \leftarrow \mathcal{S}_i$.
- OW3 Each f_i is efficiently computable.
  We remark that there is an efficient randomised algorithm that outputs some $y \in \mathcal{H}_i$: this algorithm generates $x \leftarrow \mathcal{S}_i$ and finds $y = \mathsf{f}_i(x)$. The resulting probability distribution is written $y \leftarrow \mathcal{H}_i$.
- OW4 Preimage-resistance: for random $i \leftarrow \mathsf{Gen}(l), y \leftarrow \mathcal{H}_i$, the problem of finding a value $x \in \mathcal{S}_i$ such that $\mathsf{f}_i(x) = y$ is intractable.
- OW5 Second-preimage-resistance: for random $i \leftarrow \mathsf{Gen}(l), x \leftarrow \mathcal{S}_i$, the problem of finding a value $z \in \mathcal{S}_i$, $z \neq x$, such that $\mathsf{f}_i(x) = \mathsf{f}_i(z)$ is intractable.

Property OW5 will imply non-malleability of the signature scheme. It is always true if f_i is injective.
Note that if the preimage-resistance has intractability k bits, then the set $\mathcal{H}_i$ has at least 2^k elements, because exhaustive sampling in $\mathcal{S}_i$ is always possible.

Family of claw-free functions. A family of claw-free functions is a collection of functions $\{\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i, \mathsf{g}_i : \mathcal{T}_i \to \mathcal{H}_i | i \in \mathcal{I}\}$ over some index set $\mathcal{I} = \bigoplus_l \mathcal{I}_l$ with the following properties.

- CF1 There is an efficient randomised algorithm Gen which takes as input a length parameter l and outputs an index $i \in \mathcal{I}_l$.
- CF2 There are efficient randomised algorithms which on input i output $x \in \mathcal{S}_i$ or $z \in \mathcal{T}_i$.
- CF3 Each f_i and g_i are efficiently computable.
- CF4 Claw-freeness: for a random $i \leftarrow \mathsf{Gen}(l)$, the problem of finding two values $x \in \mathcal{S}_i$ and $z \in \mathcal{T}_i$ such that $\mathsf{f}_i(x) = \mathsf{g}_i(z)$ is intractable.

We remark that if (f, g) is claw-free, both f and g are preimage-resistant.
Note that if the claw-freeness has intractability k bits then the set $\mathcal{H}_i$ has at least 2^{2k} elements, or exhaustive sampling in $\mathcal{S}_i$ and $\mathcal{T}_i$ would lead to a collision with the birthday paradox.

Uniformity. In the above general definitions, the distributions $i \leftarrow \mathsf{Gen}(l)$, $x \leftarrow \mathcal{S}_i$ and $y \leftarrow \mathcal{H}_i$ don't need to be uniform.
A family f is said to be uniform if the following additional property holds.

- UN The distribution $y \leftarrow \mathcal{H}_i$ is indistinguishable from the uniform distribution in $\mathcal{H}_i$.

Trapdoor invertibility. The family f is invertible with a trapdoor if the following additional properties hold.

- TR1 The algorithm Gen also outputs a trapdoor trap_i.
- TR2 There is an efficient (randomised) algorithm that on input $i, \mathsf{trap}_i, y \in \mathcal{H}_i$ returns a value $x = \mathsf{f}_i^{-1}(y)$ such that $\mathsf{f}_i(x) = y$.
- TR3 The distribution of x generated by $y \leftarrow \mathcal{H}_i$ and $x = \mathsf{f}_i^{-1}(y)$ is indistinguishable from the distribution $x \leftarrow \mathcal{S}_i$.

Note that in many examples of such families of functions we have $\mathcal{S}_i = \mathcal{T}_i = \mathcal{H}_i$ and f_i and g_i are permutations, but these more general definitions are necessary to cover all submissions to NESSIE.

A generalisation to verifiable simulatable functions. While the usual definition of the hash-then-invert paradigm makes the hypothesis that the f_i are efficiently computable, this requirement can be relaxed for the FDH and PFDH constructions, as described below. This is used e.g. in Sect. 7.3.2.11.

A family of verifiable simulatable one-way functions is similar to a family of one-way functions, but with the properties OW2 and OW3 replaced by

- $\mathrm{OW2}_\mathsf{S}$ There exists an efficient randomised algorithm S_i^f which on input i outputs a pair $(x, y) \in \mathcal{S}_i \times \mathcal{H}_i$ such that $y = \mathsf{f}_i(x)$. The corresponding probability distributions are written $x \leftarrow \mathcal{S}_i$ and $y \leftarrow \mathcal{H}_i$. This algorithm simulates $x \leftarrow \mathcal{S}_i,\ y = \mathsf{f}_i(x)$.
- $\mathrm{OW3}_\mathsf{T}$ Each test function $\mathsf{T}_i^\mathsf{f} : \mathcal{S}_i \times \mathcal{H}_i \to 0/1$ defined by $\mathsf{T}_i^\mathsf{f}(x, y) = \big(y \stackrel{?}{\neq} \mathsf{f}_i(x)\big)$ is efficiently computable.

A family of verifiable simulatable claw-free functions is similar to a family of claw-free functions, but with the properties CF2 and CF3 replaced by

- $\mathrm{CF2}_\mathsf{S}$ There exist two efficient randomised algorithms S_i^f and S_i^g which on input i output a pair $(x, y) \in \mathcal{S}_i \times \mathcal{H}_i$ such that $y = \mathsf{f}_i(x)$ or a pair $(z, y) \in \mathcal{T}_i \times \mathcal{H}_i$ such that $y = \mathsf{g}_i(z)$ and such that the two corresponding distributions $y \leftarrow \mathcal{H}_i$ are indistinguishable.
- $\mathrm{CF3}_\mathsf{T}$ The test functions $\mathsf{T}_i^\mathsf{f} : \mathcal{S}_i \times \mathcal{H}_i \to 0/1$ and $\mathsf{T}_i^\mathsf{g} : \mathcal{T}_i \times \mathcal{H}_i \to 0/1$ defined by $\mathsf{T}_i^\mathsf{f}(x, y) = \big(y \stackrel{?}{\neq} \mathsf{f}_i(x)\big)$ and $\mathsf{T}_i^\mathsf{g}(z, y) = \big(y \stackrel{?}{\neq} \mathsf{g}_i(z)\big)$ are efficiently computable.

7.3.1.2 FDH: Full Domain Hash

Definition. This is the most natural scheme. It was formally introduced in 1988 [38] and was provided with a security proof in the random oracle model in 1993 [44].

The f-FDH digital signature scheme with appendix is defined as follows. For any $i \in \mathcal{I}$ the components are a hash function H_i with output in $\mathcal{H}_i$ and a trapdoor invertible verifiable function $\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i$ with test T_i^f. The appendix is an element of $\mathcal{S}_i$ and H_i is idealised as a (programmable) random oracle.

- The key generation algorithm runs Gen and sets $\mathsf{pk} = i$ and $\mathsf{sk} = \mathsf{trap}_i$.
- The verification algorithm on $m \| s$, where m is the message and $s \in \mathcal{S}_i$ the appendix, checks if $\mathsf{T}_i^\mathsf{f}(s, \mathsf{H}_i(m)) \stackrel{?}{=} 1$.
- The signature generation algorithm computes $h = \mathsf{H}_i(m)$ and uses the trapdoor to compute $s = \mathsf{f}_i^{-1}(h)$. The signed message is (m, s).

Note that in the multi-key setting the hash functions H_i should be independent. However, in most actual published FDH schemes H_i depends only on the length parameter.

Theorem 7.1 (Necessary conditions). *The following conditions are necessary to the existential unforgeability of the* f*-FDH signature scheme.*

1. *Preimage-resistance of* f*.*
2. *Second-preimage-resistance of* f *is necessary for non-malleability.*
3. *Collision-resistance of* H*, which implies second-preimage-resistance.*
4. *Preimage-resistance of* H*.*

Proof. We describe an attacker if one of the conditions does not hold.

1. The attacker computes $h = \mathsf{H}_i(m)$ and finds a preimage $s \in \mathsf{f}_i^{-1}(\{h\})$.
2. The attacker queries for a valid signed message (m, s) and computes another $s' \in \mathsf{f}_i^{-1}(\{\mathsf{H}_i(m)\})$.
3. With a collision $\mathsf{H}_i(m) = \mathsf{H}_i(m')$ with $m \neq m'$, the attacker queries for a valid signed message (m, s). Then (m', s) is a new valid signed message.
4. The attacker computes a pair $(s, h) = \mathsf{S}_i^{\mathsf{f}}$ and finds a preimage $m \in \mathsf{H}_i^{-1}(\{h\})$. Then (m, s) is a valid signed message.

□

Theorem 7.2. (Security result if f is verifiable simulatable one-way). *Let* f *be uniform one-way with preimage-resistance of* $k + \log_2 q_H$ *bits and second-preimage-resistance of* k *bits. If either* f_i^{-1} *is deterministic or the forger is SO-CMA, then in the random oracle model with* q_H *hash queries and* q_S *signing queries* f*-FDH has a security level of* k *bits.*

Proof. This result dates back to Bellare and Rogaway [44]. It is a special case of the security result for PFDH (see next section). □

Generic attack. Theorem 7.2 is the best possible generic security result for a FDH scheme, because it applies to a trapdoor invertible bijection of a set of 2^{2k} elements with preimage resistance of $2k$ bits. Such a trapdoor invertible bijection might exist, and the FDH scheme based on this function can be broken with 2^k hash queries and no signature query, by looking for a collision between random $\mathsf{H}_i(m)$ and random $\mathsf{f}_i(s)$.

Theorem 7.3. (Security result if f comes from a verifiable simulatable claw-free pair). *Let* (f, g) *be uniform claw-free with intractability of* $k + \log_2 q_S$ *bits with* f *having second-preimage-resistance of* k *bits. If either* f_i^{-1} *is deterministic or the forger is SO-CMA, then in the random oracle model with* q_H *hash queries and* q_S *signing queries* f*-FDH has a security level of* k *bits.*

Proof. This result applied to RSA dates back to Coron [127]. Its generalisation to claw-free pairs is due to Dodis and Reyzin [172]. It is a special case of the security result for PFDH (see next section). □

Generic attack. Theorem 7.3 is the best possible security result for a generic FDH scheme, because it applies to a trapdoor invertible bijection of a set of 2^{2k} elements with claw-freeness of k bits. Such a trapdoor invertible bijection might exist, and the FDH scheme based on this function can be broken with 2^k hash queries and no signature query, by looking for a collision between random $\mathsf{H}_i(m)$ and random $\mathsf{f}_i(s)$.

7.3.1.3 FDH with a non-deterministic f_i^{-1}

Definition. For non-deterministic f_i^{-1} the previous results only prove the security of the FDH design if the forger is not allowed to make multiple queries for a single message. However, it is possible to tweak the design to make it deterministic and have proven security. The following FDH-D technique is used in the FLASH and QUARTZ families [428, 136] and in ESIGN-D [230].

Let $\mathsf{f}_i^{-1}(r,h)$ denote the randomised algorithm that computes a preimage of h with random seed $r \leftarrow \mathcal{R}_h$. Let prf be a family of pseudo-random functions such that the output distribution of $\mathsf{prf}_\Delta(h)$ for uniform random Δ is indistinguishable from the distribution $r \leftarrow \mathcal{R}_h$.

- The key generation algorithm chooses a random k-bit index Δ, runs Gen and sets $\mathsf{pk} = i$ and $\mathsf{sk} = (\mathsf{trap}_i, \Delta)$.
- The verification algorithm on $m\|s$, where m is the message and $s \in \mathcal{S}_i$ the appendix, checks if $\mathsf{T}_i^{\mathsf{f}}(s, \mathsf{H}_i(m)) \stackrel{?}{=} 1$.
- The signature generation computes $h = \mathsf{H}_i(m)$ and $r = \mathsf{prf}_\Delta(h)$ and uses the trapdoor to compute $s = \mathsf{f}_i^{-1}(r,h)$. The signed message is (m,s).

A remark. For some schemes (QUARTZ or ESIGN-D) the distribution $r \leftarrow \mathcal{R}_h$ is defined as follows. A value r is chosen uniformly from a set $\mathcal{R}$. If it is incompatible with h then it is discarded and another r is chosen, until a compatible value is found. Then prf_Δ is a function that generates an (infinite) sequence of uniform values $r \in \mathcal{R}$ and takes the first that is compatible with h.

7.3.1.4 PFDH: Probabilistic Full Domain Hash

Definition. This scheme was defined by Coron [129] but the underlying ideas date back to Bellare and Rogaway [46].

For any $i \in \mathcal{I}$ the components are a hash function H_i with output in $\mathcal{H}_i$, a trapdoor invertible verifiable function $\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i$ with test $\mathsf{T}_i^{\mathsf{f}}$ and a set $\mathcal{R}_i$ with uniform sampling in 2^{k_i} elements. The appendix is an element of $\mathcal{S}_i \times \mathcal{R}_i$ and H_i is idealised as a (programmable) random oracle.

- The key generation algorithm runs Gen and sets $\mathsf{pk} = i$ and $\mathsf{sk} = \mathsf{trap}_i$.
- The verification algorithm on $m\|r\|s$, where m is the message, $r \in \mathcal{R}_i$ and $s \in \mathcal{S}_i$, checks if $\mathsf{T}_i^{\mathsf{f}}(s, \mathsf{H}_i(m\|r)) \stackrel{?}{=} 1$.
- The signature generation algorithm generates $r \leftarrow \mathcal{R}_i$, computes $h = \mathsf{H}_i(m\|r)$ and uses the trapdoor to compute $s = \mathsf{f}_i^{-1}(h)$. The signed message is (m,r,s).

An alternative description of PFDH is that it is an FDH signature scheme on messages of the form $m\|r$, with a restriction on the attacker. The attacker is not able to decide the r-part of his queries to the signature oracle. Therefore if $\mathcal{R}_i$ is sufficiently large, the attacker does not learn anything useful from the signing queries.

Theorem 7.4. (Security result if f is verifiable simulatable one-way). *Let f be uniform one-way with preimage-resistance of $k+\log_2 q_H$ bits and second-preimage-resistance of k bits. If either f_i^{-1} is deterministic or $q_S < \sqrt{2^{k_i}}$, then in the random oracle model with q_H hash queries and q_S signing queries f-PFDH has a security level of k bits.*
Note that when based on one-wayness, PFDH does not have better security than FDH.

Proof. This result is a generalisation of the proof by Bellare and Rogaway for FDH [44]. Dodis and Reyzin [172] give an argument showing that no better proof can be found in a black box model.

Assuming the existence of a $(t, \varepsilon, q_S, q_H)$-forger for f-PFDH, we show how to construct an algorithm (the simulator) that runs in time $t' \simeq 2t$ and either breaks the preimage-resistance of f with probability $\varepsilon' \simeq \varepsilon/(q_H + q_S)$ or breaks the second-preimage-resistance of f with probability $\varepsilon'' \simeq \varepsilon$. [3]

The simulator receives a challenge $(i, y \in \mathcal{H}_i)$. Then it chooses a random element $j_0 \in \{1...q_H + q_S\}$ and runs the forger on $\mathsf{pk} = i$.

For a hash query $m_j\|r_j$, if the answer was already defined then it is returned. Else, if it is the j_0-th query then the answer is the challenge y, and otherwise the simulator picks a random $(x_j, y_j) \leftarrow \mathsf{S}_i^{\mathsf{f}}$ and sets $\mathsf{H}_i(m_j\|r_j) = y_j$. This simulates the hash function because y_j has uniform distribution (property UN).

For a signing query m_j the simulator generates $r_j \leftarrow \mathcal{R}_i$ and internally simulates a hash query for $m_j\|r_j$. It returns the corresponding $r_j\|x_j$. This fails if the j_0-th query is a signing query, which happens with probability $\frac{q_S}{q_H+q_S}$.

It is also necessary to show that the simulator makes a good simulation of the signing algorithm. Property TR3 implies that the simulation is valid for signing queries of distinct values of m. A problem may only arise if multiple signature queries for the same message are made [497]. If two answers have the same r_j, then the simulation also answers the same x_j. If f_i^{-1} is deterministic, this is exactly the right behaviour. Else we should avoid such a collision, which we do if $q_S < \sqrt{2^{k_i}}$ or if the forger is $\mathsf{SO\text{-}CMA}$.

The forgery is some $m\|r\|s$. If the signing oracle never received m as a query and returned an answer $r_j\|x_j$ with $r_j = r$, then $\mathsf{H}_i(m\|r)$ is unset unless it was a hash query. In that case, if the forgery corresponds to the j_0-th query (which

[3] Dodis and Reyzin propose a proof where $\frac{\varepsilon'}{\varepsilon} \simeq \frac{1}{q_H+1}$, but their proof is flawed in the following way. (The reader is invited to look at the sketch proof of (b) in Sect. 3 of [172]). If the forger makes the corresponding hash query after each signing query, then with probability $\frac{q_S}{q_H}$ their simulator will be unable to answer the selected hash query. If $q_H - q_S \ll q_H$, the success of the simulator is $\frac{\varepsilon'}{\varepsilon} \simeq \frac{1}{q_H^2}$.

happens with probability $\frac{1}{q_H+q_S}$), then it gives a preimage of y. This contradicts property OW4 of one-way functions.

Else the signing oracle received m as a query and returned an answer $r\|x_j$ with $x_j \neq s$. This contradicts the second-preimage resistance, property OW5 of one-way functions.

The running time of the simulator is the running time t of the forger plus the time corresponding to $q_H + q_S$ executions of the verification algorithm, which is bounded by t. □

Theorem 7.5. (Security result if f comes from a verifiable simulatable claw-free pair). *Let (f,g) be uniform claw-free with intractability of $k + \max(\log_2 q_S - k_i, 0)$ bits, with f having second-preimage-resistance of k bits. If either f_i^{-1} is deterministic or $q_S < \sqrt{2^{k_i}}$, then in the random oracle model with q_H hash queries and q_S signing queries f-PFDH has a security level of k bits. When based on claw-freeness, PFDH does have better security than FDH.*

Proof. This result dates back to Bellare and Rogaway [44] and Coron [129] showed that this is optimal (under reasonable assumptions). The generalisation to claw-free pairs is due to Dodis and Reyzin [172].

Assuming the existence of a (t,ε,q_S,q_H)-forger for f-PFDH, we construct a simulator that either breaks the claw-freeness of (f,g) or breaks the second-preimage-resistance of f.

The simulator receives a challenge i. Then it generates a list L of q_S random elements of $\mathcal{R}_i$ and runs the forger on $\mathsf{pk} = i$. Some elements may have multiple occurrences in L.

For a hash query $m_j\|r_j$, if the answer was already defined then it is returned. Else, if $r_j \in L$ the simulator picks a random $(x_j, y_j) \leftarrow \mathsf{S}_i^{\mathsf{f}}$ and sets $\mathsf{H}_i(m_j\|r_j) = y_j$. Else the simulator picks a random $(z_j, y_j) \leftarrow \mathsf{S}_i^{\mathsf{g}}$ and sets $\mathsf{H}_i(m_j\|r_j) = y_j$. This simulates the hash function because y_j has a uniform distribution (property UN).

For a signing query m_j the simulator takes an element $r_j \in L$ and internally simulates a hash query for $m_j\|r_j$. Then it deletes r_j from L. Since r_j was in L the hash query has generated an x_j and the simulator can return the appendix $r_j\|x_j$.

For the same reason as in the previous proof, this is a good simulation of the signing algorithm if f_i^{-1} is deterministic, if $q_S < \sqrt{2^{k_i}}$ or if the forger is $\mathsf{SO\text{-}CMA}$.

If the forgery $m\|r\|s$ does not contradict the second-preimage resistance, then it corresponds to a hash query, which either had an f answer or a g answer. If it corresponds to a g answer, then it contradicts the claw-freeness because $\mathsf{f}(s) = \mathsf{g}(z_j)$. When the list L contains q elements, a g answer happens with probability $(1-2^{-k_i})^q$. The number of elements of L decreases regularly during the simulation, so the probability that the forgery corresponds to a g answer is $\varepsilon'/\varepsilon = \frac{1}{q_S}\sum_{q=0\ldots q_S}(1-2^{-k_i})^q$.
If $q_S \ll 2^{k_i}$ then $\varepsilon'/\varepsilon \simeq (1-2^{-k_i}q_S) \simeq 1$ and therefore the security loss is $-\log_2(\varepsilon'/\varepsilon) \simeq 0$.

If $q_S \gg 2^{k_i}$ then $\varepsilon'/\varepsilon \simeq \frac{1}{q_S}\frac{1}{2^{-k_i}}$ and therefore the security loss is $-\log_2(\varepsilon'/\varepsilon) \simeq \log_2 q_S - k_i$. □

7.3.1.5 PSS and PSSR: partial message recovery

Definition. PSS means Probabilistic Signature Scheme, and PSSR means Probabilistic Signature Scheme with message Recovery.

The PSS and PSSR schemes are due to Bellare and Rogaway [46]. They include some message recovery in (P)FDH. The key idea is that the output of H can be smaller than $\mathcal{H}_i$, provided that it is collision-intractable and that the input of f_i^{-1} is random.

For any $i \in \mathcal{I}$, the recovered part of the message is a_i bits long and it is an element of the set $\mathcal{A}_i = \{0,1\}^{a_i}$. The components of the scheme are a hash function H_i with output in $\mathcal{B}_i$, a hash function $\mathsf{G}_i : \mathcal{B}_i \to \mathcal{A}_i$ and a trapdoor invertible function $\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i$ such that $\mathcal{H}_i = \mathcal{A}_i \times \mathcal{B}_i$. The functions H_i and G_i are idealised as (programmable) random hash oracles. For k bits of security the set $\mathcal{B}_i$ should have at least 2^{2k} elements.

- The key generation algorithm runs Gen and sets $\mathsf{pk} = i$ and $\mathsf{sk} = \mathsf{trap}_i$.
- The verification of $\hat{m}\|s$ computes $a\|b = \mathsf{f}_i(s)$, $\bar{m}\|r = a \oplus \mathsf{G}_i(b)$ and checks $\mathsf{H}_i(\hat{m}\|\bar{m}\|r) \stackrel{?}{=} b$. The message is $m = \hat{m}\|\bar{m}$.
- The signature generation for the message $m = \hat{m}\|\bar{m}$ computes $h = \mathsf{H}_i(\hat{m}\|\bar{m}\|r)$ and $a = (\bar{m}\|r) \oplus \mathsf{G}_i(h)$ and uses the trapdoor to compute $s = \mathsf{f}_i^{-1}(a\|h)$. The signed message is $\hat{m}\|s$.

PSS is the special case where this scheme is applied to PFDH with $\hat{m} = m$ and $\bar{m} = 0...0$ or another constant.

Theorem 7.6. (Security result). *PSS(R) has the same security as (P)FDH.*

Proof. This result is due to Coron [129] and is an improvement on the original result of Bellare and Rogaway [46]. For a complete proof, the reader should look at those papers.

The main difference from (P)FDH is that the unique hash function $m \mapsto \mathsf{H}_i(m)$ is replaced by $m \mapsto a\|h$ where $h = \mathsf{H}_i(m)$ and $a = \bar{m} \oplus \mathsf{G}_i(h)$. If there is no collision in h, then all oracle queries to H_i or to G_i make a commitment to some value for m. This is the essential reason why the security of PSS(R) is the same as the security of (P)FDH if the number of hash-oracle queries is at most the square root of the number of elements of $\mathcal{F}_i$.

Note that this theorem also applies to the variant with $h\|a$ instead of $a\|h$, which just uses the different definition $\mathcal{H}_i = \mathcal{B}_i \times \mathcal{A}_i$ and is also named PSS. □

7.3.1.6 OPSSR: maximal message recovery

Definition of Basic OPSSR. OPSSR means Optimal Padding for Signature Schemes with message Recovery.

The Basic OPSSR scheme is due to Granboulan [232] and is designed to have maximal message recovery. It is based on the fact that the important property of

H_i is not one-wayness but collision-intractability. Therefore a (random) permutation can be used instead of a hash function.

For any $i \in \mathcal{I}$, the set of possible messages is $\mathcal{M}_i$ and $\mathcal{V}_i$ is a set of 2^k elements. The components of the scheme are a bijection $\mathsf{P}_i : \mathcal{H}_i \to \mathcal{M}_i \times \mathcal{V}_i$ and a trapdoor invertible function $\mathsf{f}_i : \mathcal{S}_i \to \mathcal{H}_i$. The function P_i is idealised as a (programmable) random permutation oracle.

- The key generation algorithm runs Gen and selects an arbitrary public value $v_i \in \mathcal{V}_i$, e.g. $v_i = 0^k$. It sets $\mathsf{pk} = (i, v_i)$ and $\mathsf{sk} = (\mathsf{trap}_i, v_i)$.
- The verification algorithm on s computes $m\|v = \mathsf{P}_i(\mathsf{f}_i(s))$ and checks if $v \stackrel{?}{=} v_i$.
- The signature generation algorithm uses the trapdoor to compute the signed message $s = \mathsf{f}_i^{-1}(\mathsf{P}_i^{-1}(m\|v_i))$.

A probabilistic variant of OPSSR can be defined as in PFDH, by replacing m with $m\|r$.

Definition of OPSSR. The full OPSSR scheme can do partial message recovery, which allows the signer to sign messages of arbitrary length with a fixed public key. It is a variant of Basic OPSSR where one replaces the bijection P_i by a keyed family E_i with keyspace equal to the output of a collision-intractable hash function $\hat{\mathsf{H}}_i$. It is defined by $\mathsf{P}_i(\hat{m}\|a) = \hat{m}\|\mathsf{E}_i[\hat{\mathsf{H}}_i(\hat{m})](a)$. The family E_i is idealised as a (programmable) ideal cipher.

- The key generation algorithm is the same as for Basic OPSSR.
- The verification algorithm on $\hat{m}\|s$ computes $m\|v = \hat{m}\|\mathsf{E}_i[\hat{\mathsf{H}}_i(\hat{m})](\mathsf{f}_i(s))$ and checks if $v \stackrel{?}{=} v_i$.
- The signing algorithm uses the trapdoor to compute the signed message $\hat{m}\|s$ where $s = \mathsf{f}_i^{-1}(\mathsf{E}_i^{-1}[\hat{\mathsf{H}}_i(\hat{m})](\bar{m}\|v_i))$.

Comparison with previous paddings. FDH is a special case of Basic OPSSR based on the involution $\mathsf{P}_i(m\|v) = m\|(v_i \oplus v \oplus \mathsf{H}_i(m))$ and on the family $\{\mathsf{f}_i^{\mathcal{M}} : \mathcal{M} \times \mathcal{S}_i \to \mathcal{M} \times \mathcal{H}_i\}$ defined by $\mathsf{f}_i^{\mathcal{M}}(m\|x) = m\|\mathsf{f}_i(x)$. This family $\mathsf{f}^{\mathcal{M}}$ has the same properties as f, but the proof given below does not apply to FDH because this function P_i is trivially not a random permutation.

PSS(R) is another special case of Basic OPSSR based on $\mathsf{P}_i(\hat{m}\|a\|b) = \hat{m}\|\bar{m}\|(v_i \oplus b \oplus \mathsf{H}_i(\hat{m}\|\bar{m}))$, where $\bar{m} = a \oplus \mathsf{G}_i(b)$, and its inverse $\mathsf{P}_i^{-1}(\hat{m}\|\bar{m}\|v) = \hat{m}\|(\bar{m} \oplus \mathsf{G}_i(b))\|b$, where $b = v_i \oplus v \oplus \mathsf{H}_i(\hat{m}\|\bar{m})$, and on the family $\mathsf{f}^{\hat{\mathcal{M}}}$. But the proof given below does not apply to PSS(R) because this function P_i is trivially not a random permutation.

The main advantage of OPSSR compared to PSSR is that the message expansion can be reduced to k bits (the size of v) instead of $2k$ bits (the size of $\mathcal{B}_i$).

Theorem 7.7. (Security result). *OPSSR has the same security as (P)FDH, where the random permutation model replaces the random oracle model.*

Proof. This result is due to Granboulan [232]. For a complete proof, the reader should look at this paper.

As with PSS(R), the key idea is that all oracle queries to P_i or P_i^{-1} make a commitment to some value for m. □

7.3.1.7 CPC : generalised Chained Patarin Construction

Definition. This technique is used in Quartz [136] and is also named the generalised Feistel-Patarin construction [131].

The parameter $r \geq 1$ is the number of rounds. The components are r hash functions $(\mathsf{H}_{i,j})_{j=1...r}$ with output in $\mathcal{S}_i$ and a trapdoor function $\mathsf{f}_i : \mathcal{S}_i \times \mathcal{T}_i \to \mathcal{S}_i$. The $\mathsf{H}_{i,j}$ are idealised as programmable random oracles, the set $\mathcal{S}_i$ has a group operation $\oplus$ and the appendix is an element of $\mathcal{S}_i \times \underbrace{\mathcal{T}_i \times ... \times \mathcal{T}_i}_{r \text{ times}}$,

- The key generation algorithm runs Gen and sets $\mathsf{pk} = i$ and $\mathsf{sk} = \mathsf{trap}_i$.
- The verification algorithm on $m\|s\|t_1\|...\|t_r$ sets $s_r = s$, computes the sequence $s_{j-1} = \mathsf{f}_i(s_j\|t_j) \oplus \mathsf{H}_j(m)$ for $j = r...1$ and checks if $s_0 \stackrel{?}{=} 0$.
- The signature generation algorithm sets $s_0 = 0$, computes the sequence $s_j\|t_j = \mathsf{f}_i^{-1}(s_{j-1} \oplus \mathsf{H}_j(m))$ for $j = 1...r$ and sets $s = s_r$.

If f_i^{-1} is not deterministic the same technique as for FDH-D can be used and is named CPC-D.

Theorem 7.8. (Security result). *While the proven security of CPC is the same as for FDH, the generic attack against FDH does not work against CPC. Therefore the actual security of CPC may be better than FDH.*

In fact, Courtois proved that if $q_S = 0$ then the generic attack against CPC is the best possible.

Proof. See Courtois' paper [131]. □

Generic attack. Let S be the number of elements of $\mathcal{S}_i$ and let us compute $S^{r/(r+1)}$ random values $(s\|t, \mathsf{f}_i(s\|t))$. These values allow us to invert f_i with probability $S^{-1/(r+1)}$.

The generic attack then generates $S^{r/(r+1)}$ random messages m and computes the corresponding $(\mathsf{H}_{i,j}(m))_{j=1...r}$. For each m the probability that a forgery can be made by using the precomputed partial table for f_i^{-1} is $S^{-r/(r+1)}$, so the average number of forgeries made by this attack is 1.

7.3.2 Schemes based on the Discrete Logarithm Problem

7.3.2.1 Introduction

We describe how most DL-based signature schemes [387, 465, 466, 408, 409, 370, 371, 268, 285, 4, 378, 432] can be built by mixing four components, which we call a group, a hash function, a projection and a category. We describe some possible values of each component, and give some security proofs.

All the security proofs for DL-based signature schemes work in an idealised model. Therefore it may happen that the scheme is not secure when a concrete component is chosen to replace an idealised component, and it may even happen that any choice of a concrete component makes an insecure scheme. We will show how the security can be proven in three independent ways: when the group

is idealised, or when the hash function is idealised, or when the projection is idealised. A scheme for which a security proof exists in all three models is secure in the real world unless all three concrete components are weak choices.

The toolbox. The signature scheme is built on

- A **DL-group** $\langle G \rangle$ of order q (usually a prime number) where computing discrete logarithms is hard.
  The set of possible private keys is $\mathcal{V} \subset \mathbb{Z}/q\mathbb{Z}$. If the private key of the signer is $v \in \mathcal{V}$, the corresponding public key is the element $V = G^v$. Depending on the category, we may need $\mathcal{V} = \mathbb{Z}/q\mathbb{Z}$ or $\mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$.
  In this section all groups will be denoted multiplicatively, even in the case of elliptic curves.
- A **projection**. This is a function $\mathsf{p} : \langle G \rangle \to \mathcal{R}$, where $\mathcal{R}$ can be an arbitrary set.
- A **hash function** that makes a digest of the message. It is a function $\mathsf{H} : \mathcal{M} \times \mathcal{R} \to \mathcal{H}$, where $\mathcal{M}$ is the set of possible messages.
- A **category** that defines the formulae for signature and verification.
  The category defines two functions ϕ and $\psi : \mathcal{H} \times \mathcal{R} \times \mathcal{S} \to \mathbb{Z}/q\mathbb{Z}$ and a function $\sigma : \mathcal{I} \to \mathcal{S}$, where $\mathcal{I} \subset \mathcal{H} \times \mathcal{R} \times \mathcal{V} \times \mathcal{K}$, $\mathcal{S} = \mathbb{Z}/q\mathbb{Z}$ or $(\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{K} = \mathbb{Z}/q\mathbb{Z}$ or $(\mathbb{Z}/q\mathbb{Z})^\times$.

Description. The digital signature scheme works as follows:

- *Verification.* The verification of $(m, r, s) \in \mathcal{M} \times \mathcal{R} \times \mathcal{S}$ computes $h = \mathsf{H}(m, r)$, $\alpha = \phi(h, r, s)$, $\beta = \psi(h, r, s)$, $R = G^\alpha V^\beta$ and checks if $r \stackrel{?}{=} \mathsf{p}(R)$.
- *Signature.* To sign the message m one takes a random $k \in \mathcal{K}$ and computes $R = G^k$, $r = \mathsf{p}(R)$ and $h = \mathsf{H}(m, r)$, until $(h, r, v, k) \in \mathcal{I}$ and $s = \sigma(h, r, v, k)$. The signed message is (m, r, s).
- *Parameters and keys.* For most DL-based schemes, the description of $\langle G \rangle$ is a public parameter and the public key is V. Schemes where both $\langle G \rangle$ and V are in the public key might be more secure in the multi-key setting.
- *Partial message recovery.* For some schemes the p function is designed to allow partial message recovery. The verification $r \stackrel{?}{=} \mathsf{p}(R)$ also extracts the recovered message $\bar{m}$.

7.3.2.2 DL-groups

DL-based signature schemes do computations in a cyclic group $\langle G \rangle$ of known order q and known generator G. Multiplying or taking inverses of elements of the group should be easy, but elements of $\langle G \rangle$ might be indistinguishable from elements of a larger set $\mathbb{G}$. [4]

Usually $\langle G \rangle$ is a cyclic subgroup of the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$ of integers modulo p, or an elliptic curve subgroup, and q is a prime number.

[4] Shoup [489, Sect. 13] defines the more complete notion of "abstract group" $(\mathcal{H}, \mathcal{G}, \mathbf{g}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$. His $\mathcal{G}$ is a cyclic group generated by $\mathbf{g}$: it is $\langle G \rangle$ with our notation. His $\mathcal{H}$ is a group that contains $\mathcal{G}$: it is $\mathbb{G}$ with our notation. His μ is our q and his ν is the index of $\mathcal{G}$ in $\mathcal{H}$. His $\mathcal{E}$ and $\mathcal{D}$ define bijective encodings of the elements of $\mathcal{H}$ to byte strings. His $\mathcal{E}'$ is a partial encoding, and corresponds to what we call a "projection".

The exponentiation is a bijection from $\mathbb{Z}/q\mathbb{Z}$ to $\langle G\rangle$ defined by $k \mapsto G^k$, and the discrete logarithm is the inverse of that bijection. By definition, computing the discrete logarithm in a DL-group is intractable.

$(\mathbb{Z}/q\mathbb{Z})^\times$ is the set of invertible elements of $\mathbb{Z}/q\mathbb{Z}$, and for any $k \in (\mathbb{Z}/q\mathbb{Z})^\times$ the group element G^k is a generator of $\langle G\rangle$. One must know the factorisation of q to compute inverses in $(\mathbb{Z}/q\mathbb{Z})^\times$.

Let $\#q$ be an integer smaller than $\log_2 q$ and let $[\mathbb{Z}/q\mathbb{Z}]_\#$ be the subset of $\mathbb{Z}/q\mathbb{Z}$ that contains the integers smaller than $2^{\#q}$. Then $[\mathbb{Z}/q\mathbb{Z}]_\#$ form a group under the operation $\oplus$ corresponding to the XOR of bit strings of length $\#q$.

7.3.2.3 Hash function

The function $\mathsf{H} : \mathcal{M} \times \mathcal{R} \to \mathcal{H}$ should be easy to compute, should have uniform random output for random m and may have some of the following properties.

- H is *Type I* if $\forall m \in \mathcal{M}$ and $r, r' \in \mathcal{R}$, $\mathsf{H}(m, r) = \mathsf{H}(m, r')$. This common value is called $\mathsf{H}(m)$.
  H is *Type II* if it is not *Type I*.
- H with Type I is *collision-resistant* if it is hard to find distinct inputs $m \neq m'$ such that $\mathsf{H}(m) = \mathsf{H}(m')$.
- H with Type II is *collision-resistant* if it is hard to find distinct inputs $(m, r) \neq (m', r')$ such that $\mathsf{H}(m, r) = \mathsf{H}(m', r')$.
- H with $\mathcal{H} \subset [\mathbb{Z}/q\mathbb{Z}]_\#$ and $\mathcal{R} \subset [\mathbb{Z}/q\mathbb{Z}]_\#$ is *xor-collision-resistant* if given random r, r' it is hard to find m, m' such that $\mathsf{H}(m, r) \oplus r = \mathsf{H}(m', r') \oplus r'$. For a type I hash, this is equivalent to preimage-resistance.
- H with $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{R} \subset \mathbb{Z}/q\mathbb{Z}$ is *add-collision-resistant* if given random r, r' it is hard to find m, m' such that $\mathsf{H}(m, r) + r = \mathsf{H}(m', r') + r'$. For a type I hash, this is equivalent to preimage-resistance.
- H with $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{R} \subset (\mathbb{Z}/q\mathbb{Z})^\times$ is *div-collision-resistant* if given random r, r' it is hard to find m, m' such that $\mathsf{H}(m, r)/r = \mathsf{H}(m', r')/r'$. For a type I hash, this is equivalent to preimage-resistance.
- H is *suitable as a random oracle* if the knowledge of any number of input-output pairs cannot help to build an algorithm that will compute another input-output pair without doing the computation of H.

Usually one takes a cryptographic hash function such as those studied in Chapter 4. It is likely that these functions have uniform output and have all the variants of collision-resistance mentioned above. However, strictly speaking, none is suitable as a random oracle, because of their extensibility property [109].

7.3.2.4 Projections

The function $\mathsf{p} : \langle G\rangle \to \mathcal{R}$ is easy to compute by the signer, and the verifier should be able to test if $r \stackrel{?}{=} \mathsf{p}(R)$ for $R \in \langle G\rangle$ and $r \in \mathcal{R}$. Note that if elements of $\langle G\rangle$ are indistinguishable from elements of $\mathbb{G}$, then p is defined on the complete group $\mathbb{G}$. The projection may have some of the following properties.

- p is ε-*almost uniform* if $\forall r \in \mathcal{R}$, $\Pr_{R \in \langle G \rangle}[\mathsf{p}(R) = r] \geq \varepsilon$. This is similar to the *entropy-smoothing* property defined by Shoup [489].
- p is ε-*almost invertible* if there exists an efficient algorithm to compute the function $\mathsf{p}^{-1} : \mathcal{R} \to \mathcal{P}(\langle G \rangle)$ such that
  - $\forall R \in \mathsf{p}^{-1}(r), \mathsf{p}(R) = r$
  - At least a proportion ε of the sets $\mathsf{p}^{-1}(r)$ is non empty.
  - Elements randomly taken from random sets $\mathsf{p}^{-1}(r)$ are indistinguishable from elements randomly taken from $\langle G \rangle$.
- p is $\ell + 1$-*collision-resistant* for $\ell \geq 1$ if it is hard to find distinct $R_0, ..., R_\ell$ such that $\mathsf{p}(R_0) = ... = \mathsf{p}(R_\ell)$.
- p is *suitable as a random oracle* if the knowledge of any number of input-output pairs cannot help to build an algorithm that will compute another input-output pair without doing the computation of p or p^{-1}. Note that an almost invertible function can be suitable as a random oracle (see also Sect. 7.3.2.7).

Examples of projections.

- **Identity projection.** For $\mathbb{G} \subset \mathcal{R}$ it is $\mathsf{p}(R) = R$.
  This function is almost uniform and collision-resistant. It is not almost-invertible if membership of $\langle G \rangle$ is hard to check. It is not suitable as a random oracle.
- **DSA projection.** For $\mathbb{G} = (\mathbb{Z}/p\mathbb{Z})^\times$ and $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}$ this is $\mathsf{p}(R) = R \bmod q$.
  This function is almost uniform and probably $\log q$-collision-resistant [105]. It is not almost-invertible if membership of $\langle G \rangle$ is hard to check. It is not suitable as a random oracle.
- **EC projections.** If $\mathbb{G}$ is an elliptic curve defined over some finite field $\mathbb{F}$, let (R_x, R_y) be the coordinates of a point R and $\mathsf{i}_\mathbb{F}$ a mapping from $\mathbb{F}$ to the set of integers.
  - **ECxq projection.** For $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}$ it is $\mathsf{p}(R) = \mathsf{i}_\mathbb{F}(R_x) \bmod q$.
  - **ECx2 projection.** For $\mathcal{R} = [\mathbb{Z}/q\mathbb{Z}]_\#$ it is $\mathsf{p}(R) = \mathsf{i}_\mathbb{F}(R_x) \bmod 2^{\#q}$.
  - **ECaddq projection.** For $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}$ it is $\mathsf{p}(R) = \mathsf{i}_\mathbb{F}(R_x + R_y) \bmod q$.

  These functions are almost uniform and almost invertible. They are probably $\log q$-collision-resistant. They are not suitable as a random oracle.
- **KCDSA projection.** p is a hash function with output in $\mathcal{R}$, e.g. based on SHA-1 (see Sect. 4.4.2).
  This function is uniform and collision-resistant. It is almost (because of the extensibility property) suitable as a random oracle. It is not almost-invertible.
- **Permuted projection.** Any projection $\mathsf{p}' : \langle G \rangle \to \mathcal{R}$ can be composed with a random permutation $\mathsf{P} : \langle G \rangle \to \langle G \rangle$ to obtain $\mathsf{p} = \mathsf{p}' \circ \mathsf{P}$.
  The projection p inherits the properties of p', but is also suitable as a random oracle.

Projections with partial message recovery. Let $\mathsf{F} : \langle G \rangle \times \bar{\mathcal{M}} \to \mathcal{R}$ and $\mathsf{F}^{-1} : \langle G \rangle \times \mathcal{R} \to \bar{\mathcal{M}} \cup \{fail\}$ such that $\forall R \in \langle G \rangle$, $\mathsf{F}(R, \bar{m}) = r \Leftrightarrow \mathsf{F}^{-1}(R, r) = \bar{m}$. Then the function $\mathsf{p}(R) = \mathsf{F}(R, \bar{m})$ is a projection that allows partial message recovery. The verification $r \stackrel{?}{=} \mathsf{p}(R)$ is false if, and only if, $\mathsf{F}^{-1}(R, r)$ returns *fail*.

- **PVSSR projection.** This is the composition of an arbitrary encryption function E over $\mathcal{R}$, with a key selected from $\langle G \rangle$, and a redundancy function $\rho : \bar{\mathcal{M}} \to \mathcal{R}$. The definition is $\mathsf{F}(R, \bar{m}) = \mathsf{E}_R \circ \rho(\bar{m})$ and $\mathsf{F}^{-1}(R, r) = \rho^{-1} \circ \mathsf{E}_R^{-1}(r)$. The redundancy function ρ should have the following properties:
  - it is collision resistant,
  - the inverse $\rho^{-1} : \mathcal{R} \to \bar{\mathcal{M}} \cup \{fail\}$ is easy to compute.
  - a random element $\tilde{m} \in \mathcal{R}$ is very unlikely to be the image of some $\bar{m} \in \bar{\mathcal{M}}$,

  If E is a secure cipher, then the projection is uniform, collision-resistant and suitable as a random oracle, but is not almost invertible.
- **Group projection.** This is a special case of the PVSSR projection, based on any other projection $\mathsf{p}' : \langle G \rangle \to \mathcal{G}$ such that $\mathcal{G}$ is a group with an action on $\mathcal{R}$. This defines a one-time-pad encryption scheme and the projection is $\mathsf{p}(R) = \mathsf{p}'(R) \cdot \rho(m)$.
  This projection has the same properties as p'.
- **NS projection.** This is the Group projection where $\mathcal{G} = \mathcal{R} = \mathbb{Z}/q\mathbb{Z}$ with additive action. It is defined by the equation $\mathsf{p}(R) = \mathsf{p}'(R) + \rho(m) \bmod q$.
  This projection has the same properties as p'.
- **NR projection.** This is the Group projection where $\mathbb{G} = \mathcal{G} = (\mathbb{Z}/p\mathbb{Z})^\times$ has a multiplicative action on $\mathcal{R} = \mathbb{Z}/p\mathbb{Z}$, and with a tweak of the Identity projection $\mathsf{p}'(R) = R^{-1} \bmod p$. The NR projection is defined by the equation $\rho(m) = R \cdot \mathsf{p}(R) \bmod p$.

7.3.2.5 Categories

Definition and properties. The category is described by the sets $\mathcal{V}$, $\mathcal{K}$ and $\mathcal{S}$, subsets of $\mathbb{Z}/q\mathbb{Z}$, the sets $\mathcal{H}$ and $\mathcal{R}$, the two functions ϕ and $\psi : \mathcal{H} \times \mathcal{R} \times \mathcal{S} \to \mathbb{Z}/q\mathbb{Z}$ and a set $\mathcal{I} \subset \mathcal{H} \times \mathcal{R} \times \mathcal{V} \times \mathcal{K}$.

A category should meet some of the following properties.

- **Other functions.** Let $\mathcal{A}$ be the set of possible outputs for ϕ and $\mathcal{B}$ the set of possible outputs for ψ. Five additional functions σ, λ_h, λ_s, λ_r, μ_h can be defined, and for each of these functions there exists an efficient algorithm that computes the result.
  - $\sigma : \mathcal{I} \to \mathcal{S}$.
  - $\lambda_h : \mathcal{A} \times \mathcal{B} \times \mathcal{R} \to \mathcal{H}$
  - $\lambda_s : \mathcal{A} \times \mathcal{B} \times \mathcal{R} \to \mathcal{S}$
  - $\lambda_r : \mathcal{A} \times \mathcal{B} \times \mathcal{H} \to \mathcal{R}$
  - $\mu_h : \mathcal{S} \times \mathcal{R} \times \mathcal{V} \times \mathcal{K} \to \mathcal{H}$
- **Main properties.** These properties are mandatory for all DL-based schemes.
  - ($\mathfrak{m}1$) For all $(h, r, v, k) \in \mathcal{I}$, the value $s = \sigma(h, r, v, k)$ is such that if $\alpha = \phi(h, r, s)$ and $\beta = \psi(h, r, s)$ then $k = \alpha + v \cdot \beta$.
  - ($\mathfrak{m}2$) For all $v \in \mathcal{V}$ and $h \in \mathcal{H}$, $\Pr_{r \in \mathcal{R},\, k \in \mathcal{K}}[(h, r, v, k) \in \mathcal{I}] \geq \varepsilon_{\mathfrak{m}}$.

  Property ($\mathfrak{m}1$) implies that all signatures generated by the signing algorithm are valid. Property ($\mathfrak{m}2$) implies that the expected number of random values for k needed to generate a signature is less than $\frac{1}{\varepsilon_{\mathfrak{m}}}$.

– **Other properties.**
  (o1) For all $(h, r, s) \in \mathcal{H} \times \mathcal{R} \times \mathcal{S}$ the equation $\lambda_h(\phi(h, r, s), \psi(h, r, s), r) = h$ holds.
  (o2) For all $(h, r, s) \in \mathcal{H} \times \mathcal{R} \times \mathcal{S}$ the equation $\lambda_s(\phi(h, r, s), \psi(h, r, s), r) = s$ holds.
  (o3) The function $s \mapsto \mu_h(s, r, v, k)$ is the inverse of $h \mapsto \sigma(h, r, v, k)$.
– **Additional properties for security with idealised p.**
  (p1) For fixed (h, r, v) and uniform k such that $(h, r, v, k) \in \mathcal{I}$ the value $\sigma(h, r, v, k)$ is uniform in $\mathcal{S}$.
  Note that this property together with the hypothesis that the function $k \mapsto \mathsf{p}(G^k)$ is (almost-)uniform and one-way implies that the set of possible valid appendices (r, s) for a message m is uniformly distributed.
  (p2) For fixed $h \in \mathcal{H}$ and $v \in \mathcal{V}$ and uniformly random $s \in \mathcal{S}$ and $r \in \mathcal{R}$, the value $k = \phi(h, r, s) + v \cdot \psi(h, r, s)$ is uniformly random in $\mathcal{K}$.
  Note that failure may happen if $\mathcal{I} \neq \mathcal{H} \times \mathcal{R} \times \mathcal{V} \times \mathcal{K}$ and if $k \notin \mathcal{K}$. It is accepted that the probability of this failure is negligible.
  (p3) Given random r and r', it is hard to find some (α, β) and messages m and m' such that $\lambda_h(\alpha, \beta, r) = \mathsf{H}(m, r)$ and $\lambda_h(\alpha, \beta, r') = \mathsf{H}(m', r')$.
  Note that for a type I hash, this property is usually equivalent to the preimage-resistance of H.
– **Additional properties for security with idealised H.**
  (h1) If $h = \lambda_h(\alpha, \beta, r)$ and $s = \lambda_s(\alpha, \beta, r)$, then $\alpha = \phi(h, r, s)$ and $\beta = \psi(h, r, s)$.
  (h2) $\Pr_{\alpha \in \mathcal{A}, \beta \in \mathcal{B}}[\lambda_h(\alpha, \beta, \mathsf{p}(G^\alpha V^\beta)) \in \mathcal{H} \text{ and } \lambda_s(\alpha, \beta, \mathsf{p}(G^\alpha V^\beta)) \in \mathcal{S}] \geq \varepsilon_{\mathfrak{h}}$.
– **Additional properties for security with idealised $\langle G \rangle$.**
  (g1) For all (h, r, s) the equation $\lambda_r(\phi(h, r, s), \psi(h, r, s), h) = r$ holds.
  (g2) For any (h, h', r, s), if $\lambda_r(\phi(h, r, s), \psi(h, r, s), h') = r$ then $h' = h$.

Simple categories. These are the categories where $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{R} \subset \mathbb{Z}/q\mathbb{Z}$ and where each of ϕ and ψ only does one operation in $\mathbb{Z}/q\mathbb{Z}$. These are less general than Meta-ElGamal [251] or TEGTSS [105] schemes, but cover all actual published schemes.

Examples. These are taken from the literature. Properties (m1), (m2), (o1), (o2), (o3), (p1), (p2), (h1) and (h2) hold for all these examples.

– **ElGamal category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{R} = \mathcal{V} = \mathcal{K} = \mathcal{S} = \mathcal{B} = (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = \mathbb{Z}/q\mathbb{Z}$. Because $\mathcal{I} = \{(h, r, v, k) | h + v \cdot r \in (\mathbb{Z}/q\mathbb{Z})^\times\}$ property (p2) can fail with negligible probability. Property (p3) is equivalent to div-collision-resistance of H. Properties (g1) and (g2) hold with the restrictions $\mathcal{H} \subset (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = (\mathbb{Z}/q\mathbb{Z})^\times$. Properties (m2) and (h2) hold with $\varepsilon_{\mathfrak{m}} = \frac{\varphi(q)}{q}$ and $\varepsilon_{\mathfrak{h}} = \frac{|\mathcal{H}|}{q}$.

$$\begin{aligned} \phi(h, r, s) &= h/s & \lambda_h(\alpha, \beta, r) &= \alpha\beta^{-1} \cdot r \\ \psi(h, r, s) &= r/s & \lambda_s(\alpha, \beta, r) &= \beta^{-1} \cdot r \\ \sigma(h, r, v, k) &= (h + v \cdot r)/k & \lambda_r(\alpha, \beta, h) &= \alpha^{-1}\beta \cdot h \\ \mu_h(s, r, v, k) &= k \cdot s - v \cdot r \end{aligned}$$

– **Inverse ElGamal category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{R} = \mathcal{V} = \mathcal{K} = \mathcal{S} = \mathcal{B} = (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = \mathbb{Z}/q\mathbb{Z}$. Because $\mathcal{I} = \{(h, r, v, k) | h + v \cdot r \in (\mathbb{Z}/q\mathbb{Z})^\times\}$ property ($\mathfrak{p}2$) can fail with negligible probability. Property ($\mathfrak{p}3$) is equivalent to div-collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) hold with the restrictions $\mathcal{H} \subset (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = (\mathbb{Z}/q\mathbb{Z})^\times$. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = \frac{\varphi(q)}{q}$ and $\varepsilon_\mathfrak{h} = \frac{|\mathcal{H}|}{q}$.

$$\begin{aligned} &\phi(h, r, s) = h \cdot s && &&\lambda_h(\alpha, \beta, r) = \alpha\beta^{-1} \cdot r \\ &\psi(h, r, s) = r \cdot s && &&\lambda_s(\alpha, \beta, r) = r^{-1} \cdot \beta \\ &\sigma(h, r, v, k) = k/(h + v \cdot r) && &&\lambda_r(\alpha, \beta, h) = \alpha^{-1}\beta \cdot h \\ &\mu_h(s, r, v, k) = k/s - v \cdot r \end{aligned}$$

– **GOST category.** Let $\mathcal{H} \subset (\mathbb{Z}/q\mathbb{Z})^\times$, $\mathcal{V} = \mathcal{K} = \mathcal{S} = \mathcal{A} = \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{R} = \mathcal{B} = (\mathbb{Z}/q\mathbb{Z})^\times$. Property ($\mathfrak{o}3$) needs the restriction $\mathcal{K} = (\mathbb{Z}/q\mathbb{Z})^\times$. Property ($\mathfrak{p}3$) is equivalent to div-collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) hold. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = \frac{|\mathcal{H}|}{q}$.

$$\begin{aligned} &\phi(h, r, s) = s/h && &&\lambda_h(\alpha, \beta, r) = \beta^{-1} \cdot r \\ &\psi(h, r, s) = r/h && &&\lambda_s(\alpha, \beta, r) = \alpha\beta^{-1} \cdot r \\ &\sigma(h, r, v, k) = k \cdot h - v \cdot r && &&\lambda_r(\alpha, \beta, h) = \beta \cdot h \\ &\mu_h(s, r, v, k) = (s + v \cdot r)/k \end{aligned}$$

– **GDSA category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{K} = \mathcal{S} = \mathcal{A} = \mathcal{B} = \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{R} = \mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$. Property ($\mathfrak{p}3$) is equivalent to div-collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) hold with the restrictions $\mathcal{H} \subset (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = (\mathbb{Z}/q\mathbb{Z})^\times$. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = \frac{|\mathcal{H}|}{q}$.

$$\begin{aligned} &\phi(h, r, s) = h/r && &&\lambda_h(\alpha, \beta, r) = \alpha \cdot r \\ &\psi(h, r, s) = s/r && &&\lambda_s(\alpha, \beta, r) = \beta \cdot r \\ &\sigma(h, r, v, k) = (k \cdot r - h)/v && &&\lambda_r(\alpha, \beta, h) = \alpha^{-1} \cdot h \\ &\mu_h(s, r, v, k) = k \cdot r - v \cdot s \end{aligned}$$

– **KCDSAadd category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{R} = \mathcal{K} = \mathcal{S} = \mathcal{A} = \mathcal{B} = \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$. Property ($\mathfrak{p}3$) is equivalent to add-collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) hold. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = \frac{|\mathcal{H}|}{q}$.

$$\begin{aligned} &\phi(h, r, s) = h + r && &&\lambda_h(\alpha, \beta, r) = \alpha - r \\ &\psi(h, r, s) = s && &&\lambda_s(\alpha, \beta, r) = \beta \\ &\sigma(h, r, v, k) = (k - (h + r))/v && &&\lambda_r(\alpha, \beta, h) = \alpha - h \\ &\mu_h(s, r, v, k) = (k - v \cdot s) - r \end{aligned}$$

– **KCDSAxor category.** Let $\mathcal{H} = \mathcal{R} = \mathcal{A} = [\mathbb{Z}/q\mathbb{Z}]_\#$, $\mathcal{K} = \mathcal{S} = \mathcal{B} = \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$. Property ($\mathfrak{p}3$) is equivalent to xor-collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) hold. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = 1$.

$\phi(h,r,s) = h \oplus r$ $\quad\quad$ $\lambda_h(\alpha,\beta,r) = \alpha \oplus r$
$\psi(h,r,s) = s$ $\quad\quad$ $\lambda_s(\alpha,\beta,r) = \beta$
$\sigma(h,r,v,k) = (k - (h \oplus r))/v$ $\quad\quad$ $\lambda_r(\alpha,\beta,h) = \alpha \oplus h$
$\mu_h(s,r,v,k) = (k - v \cdot s) \oplus r$

– **Schnorr category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{V} = \mathcal{K} = \mathcal{S} = \mathcal{A} = \mathbb{Z}/q\mathbb{Z}$ and $\mathcal{B} = \mathcal{H}$. The variable r is not used and is taken from an arbitrary set $\mathcal{R}$. Property ($\mathfrak{p}3$) is implied by the collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) do not hold because λ_r cannot be defined. Property ($\mathfrak{o}3$) needs the restriction $\mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = 1$.

$\phi(h,r,s) = s$ $\quad\quad$ $\lambda_h(\alpha,\beta,r) = \beta$
$\psi(h,r,s) = h$ $\quad\quad$ $\lambda_s(\alpha,\beta,r) = \alpha$
$\sigma(h,r,v,k) = k - v \cdot h$
$\mu_h(s,r,v,k) = (k - s)/v$

– **Swapped-Schnorr category.** Let $\mathcal{H} \subset \mathbb{Z}/q\mathbb{Z}$, $\mathcal{K} = \mathcal{S} = \mathcal{B} = \mathbb{Z}/q\mathbb{Z}$, $\mathcal{V} = (\mathbb{Z}/q\mathbb{Z})^\times$ and $\mathcal{A} = \mathcal{H}$. The variable r is not used and is taken from an arbitrary set $\mathcal{R}$. Property ($\mathfrak{p}3$) is implied by the collision-resistance of H. Properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$) do not hold because λ_r cannot be defined. Properties ($\mathfrak{m}2$) and ($\mathfrak{h}2$) hold with $\varepsilon_\mathfrak{m} = 1$ and $\varepsilon_\mathfrak{h} = 1$.

$\phi(h,r,s) = h$ $\quad\quad$ $\lambda_h(\alpha,\beta,r) = \alpha$
$\psi(h,r,s) = s$ $\quad\quad$ $\lambda_s(\alpha,\beta,r) = \beta$
$\sigma(h,r,v,k) = (h - k)/v$
$\mu_h(s,r,v,k) = v \cdot s + k$

7.3.2.6 Examples of published signature schemes

– The ElGamal scheme [184] is defined on the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$, with a slight variant of the ElGamal category (where $-r$ replaces r), the identity projection and a type I hash.
– The DSA scheme [387] is defined on a prime order subgroup of the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$, with the ElGamal category, the DSA projection and a type I hash.
– The ECDSA scheme [268] is defined on a prime order elliptic curve subgroup with the ElGamal category, the ECxq projection and a type I hash.
– The GOST 34.10 scheme [370] is defined on a prime order multiplicative subgroup of $\mathbb{Z}/p\mathbb{Z}$, with a slight variant of the GOST category (where $-r$ replaces r), the DSA projection and a type I hash.
– The KCDSA scheme [285] is defined on a prime order multiplicative subgroup of $\mathbb{Z}/p\mathbb{Z}$ or on a prime order elliptic curve subgroup, with the KCDSAxor category, the KCDSA projection and a type I hash, where some certification data is hashed together with the message.
– The ECGDSA scheme [9] is defined on a prime order elliptic curve subgroup, with the GDSA category, the ECxq projection and a type I hash.
– The DSA-II scheme [105] is defined on a prime order multiplicative subgroup of $\mathbb{Z}/p\mathbb{Z}$, with the ElGamal category, the KCDSA projection and a type II hash.

- The ECDSA-II scheme [346] is defined on a prime order elliptic curve subgroup, with the ElGamal category, the ECxq projection and a type II hash.
- The ECDSA-III scheme [346] is defined on a prime order elliptic curve subgroup, with the ElGamal category, the ECaddq projection and a type II hash.
- The Schnorr scheme [465] is defined on a prime order multiplicative subgroup of $\mathbb{Z}/p\mathbb{Z}$, with a slight variant of the Schnorr category (where $-h$ replaces h), the identity projection and a type II hash.
- The GPS-sign scheme [221] is defined on a subgroup of $\mathbb{Z}/n\mathbb{Z}$ with a variant of the Schnorr category (where $\mathcal{S} = \mathbb{Z}$), the identity projection and a type II hash. The composite modulus n is viewed as part of the public key and not as a scheme parameter.
- The Nyberg-Rueppel scheme [408, 409] is a scheme with total message recovery, and hence no variable m. It is defined on a prime order multiplicative subgroup of $\mathbb{Z}/p\mathbb{Z}$, with the Schnorr category, the NR projection and the type II hash defined by $\mathsf{H}(r) = r \bmod q$.
- The PVSSR scheme (Pintsov-Vanstone Signature Scheme with message Recovery [432]) is defined on a prime order elliptic curve subgroup with a slight variant of the Schnorr category (where $-h$ replaces h), the PVSSR projection and a type II hash.
- The Naccache-Stern scheme [378] is defined on a prime order elliptic curve subgroup with the ElGamal category, the NS projection based on ECxq projection and a type I hash.
- The Abe-Okamoto scheme [4] is a scheme with total message recovery, and hence no variable m. It is defined on a prime order elliptic curve subgroup with the Schnorr category, the xor variant of the NS projection based on ECx2 projection and the type II hash $\mathsf{H}(r)$.

7.3.2.7 Initial results to be used in the security proofs

The random oracle model for almost invertible functions. The random oracle model builds an oracle for a one-way function f, that answers queries for $\mathsf{f}(x)$ with some uniformly distributed value y. Suitable functions have uniform output, are collision-resistant, etc.

If the function f is almost invertible, then the random oracle model should also allow queries for $\mathsf{f}^{-1}(x)$. Many results that were proven for the original random oracle model are also valid for this model.

The forking lemma. This lemma is found e.g. in [435] and is a tool for proofs of security in the random oracle model. The lemma holds when the scheme has the following property: each forgery can be linked to a unique "critical" query to the random oracle. The critical query is an input x such that knowing $x \overset{\mathsf{f}}{\mapsto} y$ is necessary to check if the forgery is valid.

The forking lemma also holds in the random oracle model for an almost invertible function, if the critical query hypothesis holds.

Let q_S be the number of signature queries, q_H the number of oracle queries, n_H the number of possible outputs for the random oracle and ε the probability that the forger outputs a valid forgery.

Lemma 7.1. (Forking lemma). *There exist constants c_0 and c_1 such that if $\varepsilon \geq c_0/n_H$ then after an expected number of $c_1 \cdot q_H/\varepsilon$ replays of the simulation with different choices for the random oracle, one can obtain (with some probability ε') another forgery with the same critical query but having another uniform random answer.*

In [435, *Lemma 8*] we have $c_0 = 7q_H$, $c_1 = 2(7 + \frac{1}{q_H})$ and $\varepsilon' = 3/25$ and also [435, *Theorem 10*] $c_0 = 7q_H$, $c_1 = 84480$ and $\varepsilon' \simeq 1$.

The improved forking lemma. This lemma is an extension of the forking lemma that is found in [105] and is used together with $\ell + 1$-collision-resistant functions.

Lemma 7.2. (Improved forking lemma). *There exist constants c_0 and c_1 such that if $\varepsilon \geq c_0/n_H$ then after an expected number of $c_1 \cdot q_H/\varepsilon$ replays of the simulation with different choices for the random oracle, one can obtain (with some probability ε') ℓ other forgeries with the same critical query but having other uniform random answers.*

In [105, *Lemma 10*] we have $c_0 = 4$, $c_1 = 24\ell \log(2\ell) + \frac{1}{q_H}$ and $\varepsilon' = 1/96$.

Unique representation.

Lemma 7.3. (Unique representation). *If the discrete logarithm of $V \in \langle G \rangle$ is hard to compute and if two representations $R = G^{\alpha}V^{\beta}$ and $R = G^{\alpha'}V^{\beta'}$ can be computed then $\alpha = \alpha'$ and $\beta = \beta'$.*

Proof. This is proven by $(\alpha - \alpha') = (\beta' - \beta) \cdot \log V$. □

7.3.2.8 Security proof with idealised p

This proof is based on one of the results from [105]. In this proof, H may be a Type I or Type II hash function, and p may be almost invertible.

Theorem 7.9. *A DL-based signature scheme is existentially unforgeable (and non-malleable) under adaptive chosen message attacks if the discrete logarithm is hard, if H is collision-resistant, if p is a random oracle and if the category has properties* ($\mathfrak{o}1$), ($\mathfrak{o}2$), ($\mathfrak{p}2$), ($\mathfrak{p}1$), *and* ($\mathfrak{p}3$). *The security reduction is loose.*

Proof. To answer a signature query for m, the simulator generates a random r and a random s, and computes $h = \mathsf{H}(m, r)$ and $R = G^{\phi(h,r,s)}V^{\psi(h,r,s)}$. With property ($\mathfrak{p}2$), the value R is uniformly distributed and with property ($\mathfrak{p}1$) the value s has the same distribution as for the signing algorithm. The simulator sets the oracle table $\mathsf{p}(R) := r$. The signed message is (m, r, s).

p-oracle queries that were not defined by a signature query are answered with a random value. If p is almost invertible, then p^{-1}-oracle queries are answered with some $R = G^{\alpha'}V^{\beta'}$ for random α' and β'. The probability that the oracle table cannot be set is the probability of a collision in R, which is low if $(q_H + q_S)^2 \leq q$.

When the forger outputs its forgery (m, r, s), the critical query is the value $R = G^{\alpha}V^{\beta}$ where $\alpha = \phi(h, r, s)$, $\beta = \psi(h, r, s)$ and $h = \mathsf{H}(m, r)$.

Let us suppose that the critical query was part of a signature query for some m' that answered $(m', r', s') \neq (m, r, s)$. We define $h' = \mathsf{H}(m', r')$, $\alpha' = \phi(h', r', s')$ and $\beta = \psi(h', r', s')$. Validity of the signature means that $R = G^{\alpha'} V^{\beta'}$, and the unique representation of R implies $\alpha' = \alpha$ and $\beta' = \beta$. We also have $r' = r = \mathsf{p}(R)$. Property ($\mathfrak{o}1$) implies $h = \lambda_h(\alpha, \beta, r) = h'$ and property ($\mathfrak{o}2$) implies $s = \lambda_s(\alpha, \beta, r) = s'$. Therefore $(m', r', s') \neq (m, r, s)$ implies $m' \neq m$. Since $\mathsf{H}(m', r') = \mathsf{H}(m, r)$ we have found a collision in H.

Let us suppose that the critical query was a p-oracle query for R. The forking lemma allows us to find another forgery $(m', r', s') \neq (m, r, s)$ with the same critical R but a different oracle for p. The unique representation of R implies $\alpha' = \alpha$ and $\beta' = \beta$. Therefore the simulator can find α, β, m and m' such that $\lambda_h(\alpha, \beta, r) = \mathsf{H}(m, r)$ and $\lambda_h(\alpha, \beta, r') = \mathsf{H}(m', r')$ for random r and r', which is intractable if ($\mathfrak{p}3$) holds.

Let us suppose that the critical query was a p^{-1}-oracle query that returned $R = G^{\alpha'} V^{\beta'}$. The unique representation of R implies $\alpha' = \alpha$ and $\beta' = \beta$, which is very unlikely because α' and β' were kept secret. □

7.3.2.9 Security proof with idealised H of type II

This proof is based on one of the results from [105]. In this proof, H is a type II hash function.

Theorem 7.10. *A DL-based signature scheme is existentially unforgeable under adaptive chosen message attacks if the discrete logarithm is hard, if* H *is a random oracle with large output set, if* p *is almost uniform and* $\ell + 1$*-collision-resistant and if the category has properties* ($\mathfrak{o}1$)*,* ($\mathfrak{o}2$)*,* ($\mathfrak{h}1$)*, and* ($\mathfrak{h}2$)*. Collision-resistance of* p *also implies non-malleability. The security reduction is loose.*

Proof. To answer a signature query, the simulator generates random $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ and computes $R = G^{\alpha} V^{\beta}$, $r = \mathsf{p}(R)$, $h = \lambda_h(\alpha, \beta, r)$ and $s = \lambda_s(\alpha, \beta, r)$, until $h \in \mathcal{H}$ and $s \in \mathcal{S}$. This is equivalent to using the signature generation algorithm with $k = \alpha + v \cdot \beta$, and therefore this simulation has the same output distribution. Property ($\mathfrak{h}2$) says that the expected number of random α, β needed is less than $\frac{1}{\varepsilon_{\mathfrak{h}}}$. The simulator sets the oracle table $\mathsf{H}(m, r) := h$. The signed message is (m, r, s).

Oracle queries that were not defined by a signature query are answered with a random value. The value of $R = G^{\alpha} V^{\beta}$ is uniformly distributed for random α and β. If p is $\frac{1}{n}$-almost uniform then the probability that the oracle table cannot be set is bounded by the probability of a collision in r, which is low if $(q_H + q_S)^2 \leq n$.

When the forger outputs its forgery (m, r, s), the critical query is the H-oracle query of (m, r).

Let us suppose that the critical query was part of a signature query. This signature query returned a valid (m, r, s') with the same oracle. Therefore $h' = h$. If p is collision-resistant, then $R = R'$ and its unique representation implies $\alpha' = \alpha$ and $\beta' = \beta$, so property ($\mathfrak{o}2$) implies $s' = s$.

Let us suppose that the critical query was an oracle query for (m, r). The improved forking lemma allows us to find ℓ other forgeries (m, r, s_i) with the

same critical (m, r) but different oracles for H. Since all $\mathsf{p}(R_i) = r$, the $\ell + 1$ collision-resistance of p implies that there exists a pair where $R_i = R_j$. Unique representation implies $\alpha_i = \alpha_j$ and $\beta_i = \beta_j$. Property ($\mathfrak{o}1$) implies a unique possible value $h_i = h_j$, which is unlikely to be the one given by the two different oracles for H, because the output set is large. □

7.3.2.10 Security proof with idealised $\langle G \rangle$

The generic group model was introduced by Shoup [486] and extended by Brown [108] to prove the security of ECDSA.

Theorem 7.11. *A DL-based signature scheme is existentially unforgeable under adaptive chosen message attacks in the generic group model if* H *is uniform, one-way and collision-resistant, if* p *is almost uniform and almost invertible and if the category has properties* ($\mathfrak{g}1$) *and* ($\mathfrak{g}2$)*. The security reduction is tight.*

Proof. We don't include in this document the proof given in [108] but we show below how it can be adapted to other schemes than ECDSA, using our general framework.

The proof was written for a type I hash function, but it also works for a type II hash. It was written for ElGamal category, but it works for all categories with properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$).

- In [Table 1] step 3 of **Hint** is replaced with
  $s_{m+1} = z_1 \cdot \sigma(h_{m+1}, \mathsf{p}(A_{m+1}), z_2 z_1^{-1}, z_{m+1})$.
- In [Table 2] steps 1 and 2 of **Hint** are replaced with
  $C_{(m+1)1} = \phi(h_{m+1}, \mathsf{p}(A_{m+1}), s_{m+1})$ and
  $C_{(m+1)2} = \psi(h_{m+1}, \mathsf{p}(A_{m+1}), s_{m+1})$.
- In [Table 4] step 2.b should use $\mathsf{p}^{-1}(\lambda_r(C_{i1}, C_{i2}, e))$.
- In [Table 7] step 1.b.iii should use $\mathsf{p}^{-1}(\lambda_r(C_{i1}, C_{i2}, \hat{e}_i))$.

Property ($\mathfrak{g}2$) is used when the proof shows that
$r = ... = \lambda_r(C_{m1}, C_{m2}, \hat{e}_i) = \lambda_r(\phi(\mathsf{H}(m), r, s_m), \psi(\mathsf{H}(m), r, s_m), \hat{e}_i)$
and then deduces that $\hat{e}_i = \mathsf{H}(m)$. □

7.3.2.11 Security proof with idealised H of type I

A proof where the scheme is seen as an FDH scheme. Under some conditions, DL-based schemes can easily fit into the hash-then-invert paradigm.

Theorem 7.12. *A type I DL-based signature scheme is existentially unforgeable under* ***single-occurrence*** *chosen message attacks if* H *is a random oracle,* p *is invertible, the category has properties* ($\mathfrak{h}1$) *and* ($\mathfrak{o}3$)*, and the following problem is intractable: given* $h \in \mathcal{H}$*, finding* $(r, s) \in \mathcal{R} \times \mathcal{S}$ *such that* $r = \mathsf{p}(G^{\phi(h,r,s)} V^{\cdot \psi(h,r,s)})$.

Proof. The FDH scheme based on the function $\mathsf{f}(r \| s) = \mu_h(s, r, v, \log_G(\mathsf{p}^{-1}(r)))$, is exactly the Type I scheme based on this category and projection p.

- f is not efficiently computable if the discrete logarithm is hard.
- The test function $\mathsf{T}^{\mathsf{f}}(r \| s, h) = \left(r \stackrel{?}{=} \mathsf{p}(G^{\phi(h,r,s)} V^{\psi(h,r,s)})\right)$ is efficiently computable from the public information.

– The simulation function takes random $\alpha \in \mathcal{A}$, $\beta \in \mathcal{B}$ and computes $\mathsf{S}^{\mathsf{f}}(\alpha, \beta) = (r \| s, h)$ with $r = \mathsf{p}(G^{\alpha} V^{\beta})$, $s = \lambda_s(\alpha, \beta, r)$ and $h = \lambda_h(\alpha, \beta, r)$.
– The randomised inverse $\mathsf{f}^{-1}(k, h) = \mathsf{p}(G^k) \| \sigma(h, \mathsf{p}(G^k), v, k)$ is efficiently computable with the trapdoor.

The function f is verifiable simulatable trapdoor and its preimage-resistance is based on the intractability of the following problem: given $h \in \mathcal{H}$, finding $(r, s) \in \mathcal{R} \times \mathcal{S}$ such that $r = \mathsf{p}(G^{\phi(h,r,s)} V^{\cdot \psi(h,r,s)})$. This problem is provably as hard as the discrete logarithm in $\langle G \rangle$ if p is replaced with a random oracle. This result is weaker than the previous ones, because two components need to be simultaneously idealised.

Non-malleability of the scheme is also equivalent to second-preimage resistance of f. □

Proofs based on "semilogarithm" problems. A result of Brown [107] can be seen as an improvement of Theorem 7.12. It is a proof that the single-occurrence security of ECDSA is equivalent, in the random oracle model for H, to the intractability of an *ad hoc* "semilogarithm" problem. This result applies directly to the ElGamal category and can be generalised to other categories.

Definition. *An instance of the $(\bar{\phi}, \bar{\psi})$-semilogarithm problem in the group $\langle G \rangle$ with projection p is a random element $P \in \langle G \rangle$. A solution is a pair (r, u) such that $r = \mathsf{p}(G^{\bar{\phi}(r,u)} P^{\bar{\psi}(r,u)})$.*

Theorem 7.13. (Intractability of the semilogarithm problem is necessary for the security of ECDSA). *If there exists a solver for the (u, ru)-semilogarithm problem, then one can attack all Type I schemes based on the ElGamal category, e.g. ECDSA.*

Proof. To forge a signature of m under public key V, one computes $h = \mathsf{H}(m)$ and $P = V^{1/h}$, finds (r, u) a semilogarithm of P, and computes $s = h/u$. Then $r = \mathsf{p}(G^{h/s} V^{r/s})$ and (m, r, s) is a valid signed message. □

Theorem 7.14. (Intractability of the semilogarithm problem in the random oracle model is sufficient for the SO-CMA security of ECDSA). *If there exists an existential forger under **single-occurrence** chosen message attacks for a Type I scheme based on the ElGamal category, in the random oracle model for H, then there exists a solver for the (u, ru)-semilogarithm problem. The security reduction is loose.*

Proof. This proof is similar to both the proof of Theorem 7.2 (FDH) and that of Theorem 7.10 (Type II).

To find a semilogarithm of $P \in \langle G \rangle$, one pre-selects a random $h_0 \in \mathcal{H}$ and runs the forger with public key $V = P^{h_0}$.

To answer a signature query, the simulator generates random $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ and computes $R = G^{\alpha} V^{\beta}$, $r = \mathsf{p}(R)$, $h = \lambda_h(\alpha, \beta, r)$ and $s = \lambda_s(\alpha, \beta, r)$, and sets $\mathsf{H}(m) := h$. The signed message is (m, r, s).

To answer an H-oracle query, the corresponding signature query is made, with the exception of one query which is answered with h_0.

If the forgery (m, r, s) corresponds to this H-oracle query with answer h_0, then $(r, h_0/s)$ is a (u, ru)-semilogarithm of P. □

Similar results can be obtained for some other categories. Condition ($\mathfrak{h}1$) is necessary, but not sufficient.

- *Inverse ElGamal category.* The security of schemes built with the Inverse ElGamal category is based on the (u, ru)-semilogarithm problem. The proof uses $P = V^{1/h}$ and $s = u/h$.
- *GOST category.* The security of schemes built with the GOST category is based on the (u, r)-semilogarithm problem. The proof uses $P = V^{1/h}$ and $s = u \cdot h$.
- *GDSA category.* It is not clear if a similar security result can be obtained for the GDSA category.
- *KCDSAadd category.* It is not clear if a similar security result can be obtained for the KCDSAadd category.
- *KCDSAxor category.* It is not clear if a similar security result can be obtained for the KCDSAxor category.
- *Schnorr category.* The security of schemes built with the Schnorr category is based on the $(u, 1)$-semilogarithm problem. The proof uses $P = V^h$ and $s = u$. Note that this shows that a Type I scheme based on Schnorr category is insecure, because the $(u, 1)$-semilogarithm problem is easy.
- *Swapped Schnorr category.* It is not clear if a similar security result can be obtained for the Swapped Schnorr category.

7.3.2.12 Comments on the security proofs

Comparison with the results from [105]. The two above results follow closely the proofs from Brickell *et al.* [105], but their interactions with the components of the scheme are more clearly detailed. Property ($\mathfrak{p}3$) was not clearly defined in terms of interaction between the category and H.

Moreover, we showed that the proof with an idealised p also works if p is almost invertible.

Comparison with the results from [108]. Our result is more general but does not go into all the details that are considered by Brown in [108]. For example we don't consider zero-finder-resistance, because our toolbox restricts ElGamal category to $\mathcal{H} \subset (\mathbb{Z}/q\mathbb{Z})^\times$ to meet properties ($\mathfrak{g}1$) and ($\mathfrak{g}2$).

Note that footnote number 13 in [108] explains why collision-resistance together with uniformity implies preimage-resistance, so Theorem 4 of [108] doesn't mention the assumption that the hash function needs to be preimage-resistant.

7.3.2.13 Comments on the toolbox

Type I or Type II. Both approaches have their specific security proof where H is idealised. However, Type II is probably to be preferred, because of the fact that a Type I scheme with Schnorr category is insecure, while a Type II scheme with Schnorr category is probably secure, and the fact that the Type I proof only considers $\mathsf{SO\text{-}CMA}$ security.

Projections. None of the projections previously proposed in the literature has all the required properties for our three proofs. This is why we described how to build a permuted projection. The permuted EC projections probably have all the required properties, but a random fixed permutation of $\langle G \rangle$ may be difficult to design [81]. If partial message recovery is useful, Group projections are the best candidates.

Categories. No category is clearly better than the other ones.

ElGamal category and GOST category have all the required properties, but they rely on distinct semilogarithm problems which are difficult to compare.

Schnorr and Swapped-Schnorr categories are the simplest choice, but they need a Type II hash and are not proven with idealised $\langle G \rangle$.

The KCDSAadd and KCDSAxor categories have the advantage of using simpler computations.

For Schnorr category, the order q can have intractable factorisation, because no inverse is computed. For the KCDSA categories and Swapped-Schnorr category, computing inverses in $(\mathbb{Z}/q\mathbb{Z})^\times$ is only needed for key generation. If the signer's only private information is v^{-1}, neither the verifier nor the signer needs to know the factorisation of q. Intractable factorisation might be useful for an identity based scheme [358].

7.3.3 Schemes with security proven in the "real world"

7.3.3.1 Introduction

Thoughts on the signature oracle. The security proof is the description of a reduction algorithm (aka. a simulator) that interacts with a forger and uses the forgery to solve some intractable problem (see Sect. 7.2.3). One important difficulty is that the simulator needs to be able to answer the signature queries made by the forger. The simulator must know some trapdoor that allows it to generate at least q_S valid signatures for arbitrary messages. Notice that this requirement implies that the public key is generated by the simulator. To reduce the security of the scheme to the intractability of a mathematical problem, it is necessary that the forgery could not have been made by the simulator. This remark was made by Goldwasser, Micali and Rivest [229, Sect. 4].

One-time and fixed-time signature schemes. With a one-time signature scheme, a public key is used for validating one signature only. For fixed-time signature schemes, there is an *a priori* upper bound on the number of messages that can be validated with a given public key.

One-time signature schemes as chameleon hashing. A one-time signature scheme is KS-secure (also called *secure chameleon hashing* [315]) if no KS-attacker exists. Such an attacker is allowed to make *key-then-sign* queries, where the input is a message and the output is a random public key with a valid signed message. The attacker succeeds if it can make another valid signed message for one of those public keys.

The refreshing paradigm. The key idea is that all the messages will be signed by a secure one-time signature scheme, but with a different public key for each message. The public key of the secure scheme is the concatenation of all those one-time public keys. With this simple construction the public key of the whole scheme has length q_S times the length of the public key of the one-time signature scheme.

This technique for constructing signatures is also used in the online/offline approach to improve the performance of digital signature schemes [188], where a public key for a fast one-time scheme is signed by a normal secure and slow signature scheme.

7.3.3.2 Tree constructions based on the refreshing paradigm

This technique can be used to construct secure signature schemes from secure one-time signature schemes (see also [222, Volume 2, Sect. 6.4.2]).

Practical constructions are based on the refreshing paradigm and use an authentication tree to authenticate the one-time public keys with respect to a unique short public key. No such scheme has been submitted to NESSIE.

7.3.3.3 Using a RAND-secure scheme in the refreshing paradigm

Any RAND-secure signature scheme can be used to authenticate the one-time public keys. The key idea is that the RAND scheme is able to securely sign any random one-time public key, and each one-time public key can securely sign one arbitrary message.

Theorem 7.15. *If* $\mathsf{GenerateA}, \mathsf{KeyGenA}, \mathsf{SignA}, \mathsf{VerA}$ *defines a KS-secure signature scheme and if* $\mathsf{GenerateB}, \mathsf{KeyGenB}, \mathsf{SignB}, \mathsf{VerB}$ *defines a RAND-secure signature scheme, then the following signature scheme is secure under adaptive chosen message attacks.*

- $\mathsf{Generate}$ *runs* $\mathsf{GenerateA}$ *and* $\mathsf{GenerateB}$ *and outputs* $\mathsf{param} = \mathsf{paramA} \| \mathsf{paramB}$.
- KeyGen *runs* $\mathsf{KeyGenB}$ *and outputs* $\mathsf{pk} = \mathsf{pkB}$ *and* $\mathsf{sk} = \mathsf{skB}$.
- Sign *computes* $(h, \hat{h}) = \mathsf{KeyGenA}$. *Then it computes* $s = \mathsf{SignA}_{\hat{h}}(m)$ *and* $t = \mathsf{SignB}_{\mathsf{sk}}(h)$, *and outputs* $\sigma = (t, s)$.
- Ver *computes* $h = \mathsf{VerB}_{\mathsf{pk}}(t)$ *and* $m = \mathsf{VerA}_h(s)$.

Proof. Let us name (t, s) the forgery, $h = \mathsf{VerB}_{\mathsf{pk}}(t)$ the corresponding one-time public key and $m = \mathsf{VerA}_h(s)$ the corresponding message. Let us name (t_j, s_j) and (h_j, m_j) the questions and answers to the j-th query to the signing oracle.

All the information that the forger receives about the scheme A is contained in m_j, h_j, s_j, where the forger chooses m_j but the public key h_j is random. This is exactly a KS-attack.

All the information that the forger receives about the scheme B is contained in h_j, pk, t_j, where the forger does not choose the values h_j, which are random values. This is exactly a RAND-attack.

Two variants of the theorem can be proven: with or without non-malleability. With non-malleability, we define two types of forgeries.

– *KS forgery.* $\exists j_0 : t = t_{j_0}$. Therefore $s \neq s_{j_0}$ and $h = h_{j_0}$. The forger has produced a new valid signed message s for an old public key h of the scheme A, which contradicts its KS-security.
– *RAND forgery.* $\forall j, t \neq t_j$. The forger has produced a new valid signed message t for the scheme B, which contradicts its RAND-security.

Without non-malleability, we define two types of forgeries.

– *KS forgery.* $\exists j_0 : h = h_{j_0}$ and $m \neq m_{j_0}$. The forger has produced a valid signature s for a new message m and an old public key h of the scheme A, which contradicts its KS-security.
– *RAND forgery.* $\forall j, h \neq h_j$. The forger has produced a valid signature t for a new message h for the scheme B, which contradicts its RAND-security.

□

The two main examples are the GHR scheme [217] and the ACE-Sign submission to NESSIE, both of whose security is based on the strong RSA assumption. One can compare these two schemes. While all components of ACE-Sign are efficiently computable, GHR needs an efficient collision-resistant hash function with an additional property named *division-intractability*, e.g. a hash function that outputs prime numbers. On the other hand, GHR has a tight reduction to the strong RSA assumption, while the reduction for ACE-Sign is not so tight.

7.3.3.4 The Cramer-Shoup family of schemes: ACE-Sign and variants

The RAND component. The following RAND-secure scheme is the core of this family of schemes.

– *Domain parameters.* The security parameter l is the output length of a collision-resistant hash function H and determines the length of the RSA composite number and of some prime numbers.
– *Key generation algorithm.* The key generation algorithm chooses two random strong primes p and q, computes $n = pq$ and chooses two random values x and g in QR_n (quadratic residues). The public key is $\mathsf{pk} = (n, x, g)$ and the private key is $\mathsf{sk} = (p, q)$.
– *Verification algorithm.* The verification of a signed message $(m, y, e) \in \mathcal{M} \times QR_n \times [2^l, 2^{l+1}]$ begins with $h = \mathsf{H}(m)$ and checks if $y^e \stackrel{?}{=} xg^h$.
– *Signing algorithm.* To sign the message m one takes a random prime $e \in [2^l, 2^{l+1}]$ and uses the secret key to compute $y = (xg^{\mathsf{H}(m)})^{1/e}$.

Theorem 7.16. *This scheme is RAND-secure under the Strong RSA assumption, with not so tight reduction.*

Proof. Using a (t, ε, q_S)-forger, the following algorithm either solves with probability 1 the flexible RSA problem or solves with probability $1/q_S$ the e-th root problem.

We can define two types of forgeries, depending on their common values with the queries. We let h_j, y_j, e_j denote the values for the j-th query and h, y, e the values for the forgery.

- *FLEX forgery.* $\forall j, e \neq e_j$. The reduction wants to solve the flexible RSA problem for (n, u). It generates q_S random primes $e_j \in [2^l, 2^{l+1}]$ and a random $\alpha \in [1...n^2]$, then deduces the elements of the public key $g = u^{2\prod_j e_j}$ and $x = g^\alpha$. It can answer the j-th oracle query by generating a random m_j and computing $y_j = u^{2(\alpha+\mathsf{H}(m_j))\prod_{i\neq j} e_i}$. The forgery will give an e-th root of g, and therefore a non-trivial root of u.
- *GUESS forgery.* $\exists j_0 : e = e_{j_0}$. The reduction wants to find $u^{1/e'} \bmod n$. It chooses a random j_0 in $1...q_S$, sets $e_{j_0} = e'$ and generates $q_S - 1$ other random primes $e_j \in [2^l, 2^{l+1}]$. It generates random values for $\alpha \in [1...n^2]$ and $h' = \mathsf{H}(m')$, then deduces the elements of the public key $g = u^{2\prod_{j\neq j_0} e_j}$ and $x = g^{\alpha e' - h'}$. It can answer the j_0-th oracle query with $m_{j_0} = m'$, $y_{j_0} = g^\alpha$ and $e_{j_0} = e'$. It can answer all other oracle queries by generating a random m_j and computing $y_j = u^{2(\alpha e' - h' + \mathsf{H}(m_j))\prod_{i\neq j, j_0} e_i}$. If indeed the forgery is such that $e = e'$, then $(y/y_{j_0})^e = g^{\mathsf{H}(m)-h'}$ and it gives the e-th root of u. This succeeds if j_0 was correctly guessed (probability $1/q_S$).

□

The chameleon hash. The following one-time signature scheme is used in the original scheme.

- *Domain parameters.* The security parameter l is the output length of a collision-resistant hash function H and determines the length of the RSA composite number n and of a prime number e. A random f in QR_n is chosen and $g = f^e$ is computed. The public domain parameters are n, g and e. The private parameter f is used for key generation. [5]
- *Key generation algorithm.* The key generation algorithm chooses a random value z in QR_n and computes $x = z^e$. The public key is $\mathsf{pk} = (x)$ and the private key is $\mathsf{sk} = (f, z)$.
- *Verification algorithm.* The verification of a signed message $(m, y) \in \mathcal{M} \times QR_n$ begins with $h = \mathsf{H}(m)$ and checks if $y^e \stackrel{?}{=} xg^h$.
- *Signing algorithm.* To sign the message m one uses the secret key to compute $y = zf^{\mathsf{H}(m)}$.

Theorem 7.17. *This scheme is* KS*-secure under the RSA assumption, with tight reduction.*

Proof. The reduction algorithm wants to find $u^{1/e} \bmod n$. It defines $g = u$ and interacts with the forger. In answer to a key-then-sign query for the message m, the reduction generates a random y and computes the public key $x = y^e g^{-\mathsf{H}(m)}$. The forgery is a pair (m', y') such that $(y'/y)^e = g^{\mathsf{H}(m')-\mathsf{H}(m)}$, which gives an e-th root of g. □

ACE-Sign. The actual Cramer-Shoup scheme [141] is an improvement on the straightforward construction based on the two components described above.

[5] An equivalent scheme can be obtained by generating n with known secret factorisation, kept in the private key.

The security of the construction is the same if the values n and g are common to the RAND-secure scheme and to the one-time scheme. With this improvement, the public key is (n, x, g, e'), the signed message is (m, y, e, x', y') and the verification checks if $x \stackrel{?}{=} y^e g^{-\mathsf{H}(x')}$ and if $x' \stackrel{?}{=} (y')^{e'} g^{-\mathsf{H}(m)}$. If we also notice that x' can be omitted from the signed message, the result is ACE-Sign, fully described in Sect. 7.5.1.1.

Another Chameleon hash. The following scheme is also proposed as an alternative in [141]. It is based on the intractability of discrete logarithms.

- *Domain parameters.* The security parameter l is the output length of a collision-resistant hash function H. Computations are made in a group $\langle G \rangle$ of order q. Two random α, α' coprime to q are chosen such that the public parameters $g_1 = G^{\alpha'}$ and $g_2 = g_1^{\alpha}$ are generators of $\langle G \rangle$. The private parameter α is kept for key generation.
- *Key generation algorithm.* The key generation algorithm chooses a random β and computes $x = g_1^{\beta}$. The public key is $\mathsf{pk} = (x)$ and the private key is $\mathsf{sk} = (\alpha, \beta)$.
- *Verification algorithm.* The verification of a signed message $(m, t) \in \mathcal{M} \times \mathbb{Z}/q\mathbb{Z}$ begins with $h = \mathsf{H}(m)$ and checks if $x \stackrel{?}{=} g_1^t g_2^h$.
- *Signing algorithm.* To sign the message m one uses the secret key to compute $t = \beta - \alpha \mathsf{H}(m) \bmod q$.

Theorem 7.18. *This scheme is KS-secure if the discrete logarithm problem in $\langle G \rangle$ is intractable, with tight reduction.*

Proof. The reduction algorithm wants to find the logarithm of g_2 with respect to g_1. In answer to a key-then-sign query for the message m, the reduction generates a random t and computes the public key $x = g_1^t g_2^{\mathsf{H}(m)}$. The forgery is a pair (m', t') such that $g_2 = g_1^{(t-t')/(\mathsf{H}(m')-\mathsf{H}(m))}$. □

Generalisation to any group. Damgård and Koprowski [152] proposed a generalisation of ACE-Sign to any group where the equivalent of the flexible RSA problem is intractable.

Variants without a chameleon hash. The key idea is that the chameleon hash is only needed for the j_0-th query in the GUESS forgery. In all the other cases any arbitrary message can be signed by the reduction algorithm. Therefore a chameleon hash brings unnecessary flexibility.

Instead of replacing $g^{\mathsf{H}(m)}$ by $g^{\mathsf{H}(\text{chameleon}-\text{hash}(m))}$, one first variant, due to Camenisch and Lysyanskaya [112], replaces $g^{\mathsf{H}(m)}$ by $g_1^t g_2^{\mathsf{H}(m)}$. The signed message is (m, t, e, y) and the verification checks if $y^e \stackrel{?}{=} x g_1^t g_2^{\mathsf{H}(m)}$. If n is an l'-bit number and H has l-bit output (i.e. security $l/2$ bits) then this variant is secure when t is an $l' + 3l/2$-bit number. Therefore it is less efficient than ACE-Sign.

Theorem 7.19. *The Camenisch-Lysyanskaya scheme is secure under the Strong RSA assumption, with not so tight reduction.*

Proof. The proof is very similar to the proof of Theorem 7.16.

- *FLEX forgery.* $\forall j, e \neq e_j$. The reduction wants to solve the flexible RSA problem for (n, u). It generates q_S random primes $e_j \in [2^l, 2^{l+1}]$ and random $\alpha, \beta \in [1...n^2]$, then deduces the elements of the public key $g_1 = u^{2\prod_j e_j}$, $g_2 = g_1^\beta$ and $x = g_1^\alpha$. It can answer the j-th oracle query for m_j by generating a random $t_j \in [0...2^{l'+3l/2}]$ and computing $y_j = u^{2(\alpha + t_j + \beta \mathsf{H}(m_j))\prod_{i \neq j} e_i}$. The forgery will give an e-th root of g_1, and therefore a non-trivial root of u.
- *GUESS forgery.* $\exists j_0 : e = e_{j_0}$. The reduction wants to find $u^{1/e'} \bmod n$. It chooses a random j_0 in $1...q_S$, sets $e_{j_0} = e'$ and generates $q_S - 1$ other random primes $e_j \in [2^l, 2^{l+1}]$. It generates random $\alpha, \beta \in [1...n^2]$ and $\gamma \in [0...2^{l'+3l/2}]$, then deduces the elements of the public key $g_1 = u^{2\prod_{j \neq j_0} e_j}$, $g_2 = g_1^\beta$ and $x = g_1^{\alpha e' - \gamma}$. It can answer the j-th oracle queries for m_j with $j \neq j_0$ by generating a random $t_j \in [0...2^{l'+3l/2}]$ and computing $y_j = u^{2(\alpha e' - \gamma + t_j + \beta \mathsf{H}(m_j))\prod_{i \neq j, j_0} e_i}$. It can answer the j_0-th oracle query for m_{j_0} with $y_{j_0} = g^\alpha$, $e_{j_0} = e'$ and $t_{j_0} = \gamma - \beta \mathsf{H}(m_{j_0})$. Owing to the condition $\gamma \in [0...2^{l'+3l/2}]$, the value t appears to be uniform in $[0...2^{l'+3l/2}]$. If indeed the forgery is such that $e = e'$, then $(y/y_{j_0})^e = g^{t - t_{j_0} + \beta \mathsf{H}(m) - \beta \mathsf{H}(m_{j_0})}$ and it gives the e-th root of u. This succeeds if j_0 was correctly guessed (probability $1/q_S$).

□

Two other variants, due to Fischlin [195], replace $g^{\mathsf{H}(m)}$ by $g_1^t g_2^{t+\mathsf{H}(m)}$ or by $g_1^t g_2^{t \oplus \mathsf{H}(m)}$. The signed message is (m, t, e, y) and the verification checks e.g. if $y^e \stackrel{?}{=} x g_1^t g_2^{t \oplus \mathsf{H}(m)}$. These variants are secure for l-bit t, so their efficiency is similar to ACE-Sign with the DL-based chameleon hash.

Theorem 7.20. *The Fischlin schemes are secure under the Strong RSA assumption, with not so tight reduction.*

Proof. The proof is very similar to the proofs of Theorem 7.16 and 7.19.

- *FLEX forgery.* $\forall j, e \neq e_j$. The reduction wants to solve the flexible RSA problem for (n, u). It generates q_S random primes $e_j \in [2^l, 2^{l+1}]$ and random $\alpha, \beta \in [1...n^2]$, then deduces the elements of the public key $g_1 = u^{2\prod_j e_j}$, $g_2 = g_1^\beta$ and $x = g_1^\alpha$. It can answer the j-th oracle query for m_j by generating a random $t_j \in [0...2^l]$ and computing $y_j = u^{2(\alpha + t_j + \beta(t_j \oplus \mathsf{H}(m_j)))\prod_{i \neq j} e_i}$. The forgery will give an e-th root of g_1, and therefore a non-trivial root of u.
- *GUESS/1 forgery.* $\exists j_0 : e = e_{j_0}$ and $t_{j_0} \neq t$. The reduction wants to find $u^{1/e'} \bmod n$. It chooses a random j_0 in $1...q_S$, sets $e_{j_0} = e'$ and generates $q_S - 1$ other random primes $e_j \in [2^l, 2^{l+1}]$. It generates random $\alpha, \beta \in [1...n^2]$ and $\gamma \in [0...2^l]$, then deduces the elements of the public key $g_1 = u^{2\prod_{j \neq j_0} e_j}$, $g_2 = g_1^{\beta e'}$ and $x = g_1^{\alpha e' - \gamma}$. It can answer the j-th oracle queries for m_j with $j \neq j_0$ by generating a random $t_j \in [0...2^l]$ and computing $y_j = u^{2(\alpha e' - \gamma + t_j + \beta e'(t_j \oplus \mathsf{H}(m_j)))\prod_{i \neq j, j_0} e_i}$. It can answer the j_0-th oracle query for m_{j_0} with $y_{j_0} = g_1^{\alpha + \beta(\gamma \oplus \mathsf{H}(m_{j_0}))}$, $e_{j_0} = e'$ and $t_{j_0} = \gamma$. If indeed the forgery is such that $e = e'$ and $t_{j_0} \neq t$, then $(g_1^{\beta(t_{j_0} \oplus \mathsf{H}(m_{j_0})) - \beta(t \oplus \mathsf{H}(m))} \cdot y/y_{j_0})^e = g^{t - t_{j_0}}$ and it gives the e-th root of u.

– *GUESS/2 forgery.* $\exists j_0 : e = e_{j_0}$ and $t_{j_0} \oplus \mathsf{H}(m_{j_0}) \neq t \oplus \mathsf{H}(m)$. Similar to the GUESS/1 forgery, with the public key $g_2 = u^{2\prod_{j \neq j_0} e_j}$, $g_1 = g_2^{\beta e'}$ and $x = g_2^{\alpha e' - \gamma}$. Also $t_{j_0} = \gamma \oplus \mathsf{H}(m_{j_0})$ and $y_{j_0} = g_2^{\alpha + \beta t_{j_0}}$.

□

7.3.4 Current standards

The US NIST (National Institute of Standards and Technology) issued in 1994 a FIPS (U.S. Government Federal Information Processing Standard) that described DSA (Digital Signature Algorithm) [387]. This standard has been revised twice: FIPS-186-1 [388] and FIPS-186-2 [389] added the ANSI X9.31 and X9.62 standards.

The ANSI (American National Standards Institute) issued in 1999 ANSI X9.31 [15], which is a partial domain hash RSA signature, and ANSI X9.62 [16], which is ECDSA.

The ISO (International Organisation for Standardisation) has published ISO-9796 [254, 255], ISO-14888 [261, 262] and ISO-15946-2 [263].

The IEEE P1363 group has published various standards on public key cryptography [252, 253] that go down to implementation details.

7.4 Digital signature schemes considered during Phase II

The complete description of a signature scheme needs to explain how to convert integers to and from byte strings or bit strings. Usually, this has no influence on the security and will not be mentioned here.

7.4.1 ECDSA

ECDSA was submitted by Certicom [268] and is a signature scheme based on the intractability of the discrete logarithm problem in an elliptic curve subgroup. It was first proposed in 1992 by Scott Vanstone [508] in response to NIST's request for public comments on their first proposal for DSS [387]. It is an ISO [262] standard since 1998, an ANSI [16] standard since 1999 and an IEEE [252] and NIST [389] standard since 2000. Interoperability between these standards is discussed in `http://www.certicom.com/resources/news/news_103000.html`. The version submitted to NESSIE is the ANSI X9.62 ECDSA.

7.4.1.1 The design

ECDSA is a special case of the family of DL-based signature schemes described in Sect. 7.3.2. It is defined on a prime order elliptic curve subgroup with ElGamal category, ECxq projection and type I hash.

– *Domain parameters.* The security parameter is an integer l, e.g. 160. Let E be an elliptic curve over the finite field $\mathbb{F}$ and $\langle G \rangle$ a subgroup of known prime

order $q \in [2^l, 2^{l+1}]$ and known generator G. We use the additive notation for this group.
$\mathbb{F}$ is either a prime field $GF(p)$ or a characteristic 2 field $GF(2^n)$.
Let H be a hash function with l bits of output (usually SHA-1) and $\mathsf{i}_{\mathbb{F}}$ a mapping from $\mathbb{F}$ to the set of integers modulo q, that does the conversion from $\mathbb{F}$ to $\mathbb{Z}$ as specified in ANSI X9.62 and then a reduction modulo q.

- *Key generation algorithm.* The key generation algorithm chooses a random $v \in (\mathbb{Z}/q\mathbb{Z})^\times$ and sets $\mathsf{pk} = V = v \cdot G$ and $\mathsf{sk} = v$.
- *Verification algorithm.* The verification algorithm on a signed message $(m, r, s) \in \mathcal{M} \times \mathbb{Z}/q\mathbb{Z} \times (\mathbb{Z}/q\mathbb{Z})^\times$ computes $h = \mathsf{H}(m)$ and $R = s^{-1} \cdot (h \cdot G + r \cdot V)$, and checks if $r \stackrel{?}{=} \mathsf{i}_{\mathbb{F}}(R_x)$.
- *Signing algorithm.* To sign the message m one takes a random invertible $k \in (\mathbb{Z}/q\mathbb{Z})^\times$, and computes $R = k \cdot G$, $r = \mathsf{i}_{\mathbb{F}}(R_x)$, $h = \mathsf{H}(m)$ and $s = k^{-1}(h + vr)$. If s is not invertible, another k is taken. The signed message is (m, r, s).

Parameter generation. The parameters for ECDSA consist mainly of the description of a suitable elliptic curve and of a base point that generates a subgroup with large prime order.

Depending on the variant of ECDSA, the elliptic curve and base point can be chosen from a table of suitable values [116] or can be taken at random subject to the base point having large prime order. Any underlying field can be chosen, but only prime fields or binary fields are usually considered.

The elliptic curve subgroup is not part of the public key, it is a system parameter common to all users. Therefore it is important that no weaknesses can be found in it.

Okeya *et al.* [422] proposed the use of elliptic curves with Montgomery form to protect against timing attacks. This leads to the OK-ECDSA variant, which was not submitted to NESSIE but is studied by CRYPTREC [421].

7.4.1.2 Security analysis (see also Sect. 7.3.2)

Attacks on ECDSA. Necessary conditions for the security of ECDSA are one-wayness and collision resistance of H and intractability of the (u, ru)-semilogarithm in the elliptic curve E, which implies intractability of the discrete logarithm.

Security proofs for ECDSA. This signature scheme has been published for a long time and various security arguments have been provided which show that the two necessary conditions above might be sufficient. However, none of those security arguments can be used to argue that ECDSA is an optimal design for a scheme based on the intractability of the elliptic curve discrete logarithm.

The first published arguments were proofs in the random oracle model that in fact apply to variants of ECDSA, e.g. to KCDSA (proof with idealised p) or to the Schnorr or PVSSR schemes (proof with idealised H). The proof that works in the generic group model applies to ECDSA itself, but it is also an example of

a case where a generic proof is explicitly invalidated by some specific properties of the components. [6]

Therefore one big concern about ECDSA is that it is probably not the best choice in the extensive family of DL-based signature schemes.

The hash function of ECDSA. The function H should be a one-way collision resistant hash function with output in a subset of $(\mathbb{Z}/q\mathbb{Z})^\times$, and the security of the scheme may be weakened if this output set has substantially fewer than q elements.

But the specifications of ECDSA say that H is the SHA-1 hash function, whose output is a 160-bit string, reduced mod$q$. Therefore it might not be an element of $(\mathbb{Z}/q\mathbb{Z})^\times$, because it can be zero. An additional property required for SHA-1 is *zero-finder resistance* [108], which means that a preimage of 0 should be hard to find.

Some realistic attacks on DSA and ECDSA rely on the ability to choose the parameters of the scheme after having studied some properties of the hash function [85, 510]. An easy protection is the inclusion in the input of H of some certification data depending on the parameters and the public key. It has been proposed for KCDSA and some other schemes [285, 369].

Other comments: parameter and key validation. Note that to prove that the parameters are not designed to correspond to a weak elliptic curve, ECDSA asks for a certification, which is the seed used for the generation of a random curve. This certification technique can be bypassed in characteristic 2, and should absolutely be improved [513].

Note also that the ECDSA submitted to NESSIE asks that the verification and the signing algorithms make sure that $r \neq 0$, while the description in this document does not make this verification. The rationale for this check on r is to protect against a very specific type of bad parameter. It is felt that a good algorithm for parameter validation would be preferred to ad hoc checks, and that it is harmful to make the verification and signing algorithms more complicated than necessary.

The random nonce. The random value k used in the signing algorithm should be *unpredictible*, otherwise the scheme can be attacked [182, 401, 402]. However, this value can be deterministically generated from the message and a secret value, like it is done for the FDH-D design (cf. Sect. 7.3.1.3).

Side-channel attacks. In elliptic curve cryptography, scalar point multiplication is a crucial operation. As mentioned in Sect. A.1 algorithms performing this operation are a particular target for side-channel attacks. In ECDSA, the task is to compute $k \cdot G$. Any information concerning the value k that can leak during the computation might be used by an attacker to compute the secret key. Different side-channel techniques exist to get the information and various countermeasures

[6] The elliptic curve subgroup can easily be distinguished from an ideal group because it has the trivial automorphism $(x, y) \mapsto (x, -y)$. In the case of ECDSA, the choice of p being the reduction of the x-coordinate allows one to use this automorphism to forge a new signature for an existing signed message.

have been proposed in the literature to defend against this kind of attack. For further discussion on this subject, the reader is referred to the Annex A and to the survey [424] by Oswald and Preneel.

Another side-channel technique that can be used is introducing faults during the computation of the signature (for a general introduction to fault attacks, see Sect. A.2). For example, and following the original idea of Bao *et al.* [26] against DSA, if an attacker is able to change one bit of the secret key v used by the signer, then the erroneous signature can be used to recover the original value of the bit. Indeed, the signer would output $s' = k^{-1}(h + vr \pm 2^i r)$, assuming the attacker changed the i-th bit of v, and depending on its original value. So the verification algorithm would give ${s'}^{-1}(h \cdot G + r \cdot V) = k \cdot G \pm ({2^i r s'}^{-1}) \cdot G$. Thus, computing all the possible values for $R = {s'}^{-1}(h \cdot G + r \cdot V) \pm ({2^i r s'}^{-1}) \cdot G$ and detecting if $r \stackrel{?}{=} \mathsf{i}_{\mathbb{F}}(R_x)$. the attacker will discover the original value of the bit that he flipped. In [50], Biehl *et al.* present another idea: faults are used to make the device apply the multiplication algorithm to a point that is actually on a different, probably cryptographically weak, elliptic curve. The result of this computation might be used to recover the secret key v. Countermeasures against fault attacks are necessary, e.g. checking the consistency of the output.

7.4.2 ESIGN

ESIGN was submitted by NTT [207] and can be viewed as a variant of RSA that has faster signing but relies on more demanding security assumptions.

7.4.2.1 The design

- *Domain parameters.* The security parameter is an integer l, e.g. 512. Let H be a hash function with $l - 1$ bits of output and e be a small integer.
  In the original submission of ESIGN to NESSIE [207] the requirements are $l \geq 352$ and $e \geq 8$, with recommended values $l = 384$ and $e = 1024$. In the specification of ESIGN-D [416] the requirements are $l \geq 342$ and $\frac{3l}{2} \leq e \leq 2^{l/4}$, with recommended values $l = 512$ or 1024 and $e = 65537$.
- *Key generation algorithm.* Let p and q be distinct primes from $[2^{l-1}, 2^l]$ such that $n = p^2 q \in [2^{3l-1}, 2^{3l}]$. The keys are $\mathsf{pk} = n$ and $\mathsf{sk} = (p, q)$.
- *Verification algorithm.* The verification of a signed message $(m, s) \in \mathcal{M} \times \mathbb{Z}/n\mathbb{Z}$ begins with $h = \mathsf{H}(m)$ and $x = s^e \bmod n$, and checks if $x \stackrel{?}{=} h \cdot 2^{2l} + w$ where $w \in [0, 2^{2l-1}]$.
- *Basic signing algorithm.* To sign the message m one takes a random $w \in [0, 2^{2l-1}]$ and computes $h = \mathsf{H}(m)$, $x = h \cdot 2^{2l} + w$ and $s = x^{1/e} \bmod n$. The signed message is (m, s).
- *ESIGN fast signing algorithm.* To sign the message m one computes $h = \mathsf{H}(m)$. Then one takes a random $r < pq$ and computes $u = h \cdot 2^{2l} - r^e \bmod n$, $v = \lceil \frac{u}{pq} \rceil$ and $w = v \cdot pq - u$ until $w < 2^{2l-1}$. Then $t = \frac{v}{e \cdot r^{e-1}} \bmod p$ and $s = r + t \cdot pq$. The signed message is (m, s).
  Both signing algorithms give the same output distribution.

7.4.2.2 Security analysis (see also Sect. 7.3.1)

Attacks on ESIGN and its variants. Necessary conditions for the security of ESIGN are one-wayness and collision resistance of H and intractability of the AER problem.

Security proofs for ESIGN and its variants. ESIGN is an FDH signature scheme based on the "truncated e-th power" function, which is defined by $\mathsf{f}(x) = \lfloor \frac{x^e \bmod n}{2^{2l}} \rfloor$. The original description of ESIGN [207] made the statement that the original security proof of FDH applies to ESIGN, but Stern *et al.* [497] noticed that f^{-1} is randomised, and therefore this proof only assesses the security against a SO-CMA attacker.

A variant named ESIGN-D has been described [230] and replaces the original submission. This variant uses the FDH-D technique described in Sect. 7.3.1.3. Another variant named ESIGN-R is described in the same document and uses PFDH with a seed of length at least $2\log_2 q_S$.

To assess the security of ESIGN-D and ESIGN-R, we need to study the properties of f.

- *Definitions.* The set $\mathcal{I}_l$ contains all pairs (n, η) with suitable $n = p^2 q$ and $\eta \in \mathbb{Z}/n\mathbb{Z}$. Let us define $\mathcal{X} = \{x \in \mathbb{Z}/n\mathbb{Z} | x = 0 \| h \| 0 \| w$ with $h \in [0, 2^{2l-1}]$ and $w \in [0, 2^{2l-1}]\}$. The input sets are $\mathcal{S} = \mathcal{T} = \{x \in \mathbb{Z}/n\mathbb{Z} | x^e \bmod n \in \mathcal{X}\}$ and the output set is $\mathcal{H} = [0, 2^{2l-1}]$.
  We define $\mathsf{f}(x) = \lfloor \frac{x^e \bmod n}{2^{2l}} \rfloor$ and $\mathsf{g}(x) = \lfloor \frac{\eta \cdot x^e \bmod n}{2^{2l}} \rfloor$.
- *Computational properties.* These properties (OW1, OW2, OW3, CF1, CF2, CF3, TR1 and TR2) are implied by the description of the scheme, the probability distribution on $\mathcal{S} = \mathcal{T}$ being uniform, with a sampling algorithm uniformly taking an element of $\mathbb{Z}/n\mathbb{Z}$ until it is in $\mathcal{S}$.
- *Statistical properties.* Property UN (almost uniformity of the output of f and g) can be proved [183, 417, 496].
  Property TR3 comes from the following facts. Both signing algorithms generate values with their e-th power uniform in $\mathcal{X}$. The e-th power function is a bijection between $\mathcal{S}$ and $\mathcal{X}$.
- *Intractability properties.* Preimage resistance (property OW4) holds if the AER problem is hard.
  Claw-freeness (property CF4) holds if the Claw-AER problem is hard.
  Second preimage resistance (property OW5) holds if the 2nd-AER problem is hard.

Therefore, the following security results have been proven.

- If the AER problem is hard with a security level of k bits, then both ESIGN-D and ESIGN-R have a proven (in the random oracle model) security level of $k - \log_2 q_H$ bits. Non-malleability requires also that the 2nd-AER problem is hard.
- If the Claw-AER problem is hard with a security level of k bits, then ESIGN-D also has a proven security level of $k - \log_2 q_S$ bits and ESIGN-R a proven security level of k bits. Non-malleability still requires that the 2nd-AER problem is hard.

The hash function of ESIGN and its variants. Both the original description of ESIGN and the tweak to ESIGN-D use a hash function H, based on SHA-1, which is common to all the public keys. It may be a better design to improve the security in the multi-key setting by including pk in the input of H.

In the original description of ESIGN, the value $\mathsf{H}(m)$ is defined to be the first bits of $\text{SHA-1}^{80}_{\sigma}(0\|m)\|\text{SHA-1}^{80}_{\sigma}(1\|m)\|...$, where $\text{SHA-1}^{80}_{\sigma}$ is a variant of SHA-1 with a different starting value and truncated to 80 bits. This is not an optimal design because it is too slow if m is a long message. The specification of ESIGN-D uses a better design: $\text{SHA-1}(\text{SHA-1}(m)\|0)\|\text{SHA-1}(\text{SHA-1}(m)\|1)\|....$

Side-channel attacks. For a general introduction to side-channel attacks, see the Annex A. We did not find any side-channel attack against ESIGN. The fact that it is a randomised algorithm protects ESIGN against some side-channel attacks.

ESIGN-D is a deterministic variant of ESIGN. We can use this to mount a side-channel attack that uses faults (see Sect. A.2). Imagine an attacker can disturb the signing algorithm by introducing faults during the computation of either u, v or t. This leads to an erroneous value $t' = t \pm \epsilon$, with $0 \leq \epsilon < p$. Thus, an erroneous signature s' is computed, with the following relation: $s' = s \pm \epsilon \cdot pq$, where $s = r + t \cdot pq$ is the correct signature. If an attacker can get both the correct signature and an erroneous one on the same message, then he gets ϵpq, which is not a multiple of N, so $\gcd(s - s', N) = pq$ and the modulus is factorised. This attack is particularly powerful because all it needs is a random error during the computation of the most time-consuming steps. So any implementation of ESIGN-D should be protected against fault attacks.

7.4.3 SFLASHv2

All signature schemes from the FLASH family are FDH-D schemes based on the intractability of variants of the HFE problem. These variants are named C^{*--} (see [429]).

7.4.3.1 The design

- *Domain parameters.* The parameters are four integers q, d, n and r, a security parameter l, a hash function H with output in $(\mathbb{F}_q)^{n-r}$ and a family of pseudo-random functions prf with an l-bit index, input in $(\mathbb{F}_q)^{n-r}$ and output in $(\mathbb{F}_q)^r$.
  The function $\mathbf{F}(x) = x^{q^d+1}$ is a bijection of the finite field $\mathbb{F}_{q^n}$ with inverse $\mathbf{F}^{-1}(x) = x^h$, where $h = (q^d+1)^{-1} \bmod (q^n-1)$. The function $\phi : \mathbb{F}_{q^n} \to (\mathbb{F}_q)^n$ fixes a representation of $\mathbb{F}_{q^n}$ as an $\mathbb{F}_q$-vector space. Let $\mathbf{f} = \phi^{-1} \circ \mathbf{F} \circ \phi$.
  For SFLASHv2 we have $l = 80$, $q = 128 = 2^7$, $d = 11$, $n = 37$ and $r = 11$ and H and prf based on SHA-1.
- *Key generation algorithm.* Two random affine bijections $\mathbf{s}$ and $\mathbf{t}$ of $(\mathbb{F}_q)^n$ and a random l-bit value Δ are generated. They are the private key.
  The public key is $(P_1, ..., P_{n-r})$ where $(P_1, ..., P_n)$ are the quadratic polynomials that describe $\mathbf{t} \circ \mathbf{f} \circ \mathbf{s}$.

– *Verification algorithm.* The verification of a signed message $(m, s_1, ..., s_n) \in \mathcal{M} \times (\mathbb{F}_q)^n$ begins with $(h_1, ..., h_{n-r}) = \mathsf{H}(m)$ and then computes $(y_1, ..., y_{n-r}) = (P_1, ..., P_{n-r})(s_1, ..., s_n)$ and checks if all $h_i \stackrel{?}{=} y_i$.
– *Signing algorithm.* To sign the message m one computes $(h_1, ..., h_{n-r}) = \mathsf{H}(m)$, then $(x_1, ..., x_r) = \mathsf{prf}_\Delta(h_1, ..., h_{n-r})$ and finally $(s_1, ..., s_n) = \mathbf{s}^{-1} \circ \mathbf{f}^{-1} \circ \mathbf{t}^{-1}(h_1, ..., h_{n-r}, x_1, ..., x_r)$. The signed message is $(m, s_1, ..., s_n)$.
The fact that $x \mapsto x^q$ is linear over $(\mathbb{F}_q)^n$ allows for some tricks that help to do a fast computation of $\mathsf{F}^{-1}(x) = x^h$ and therefore of $\mathbf{f}^{-1}$.

7.4.3.2 Security analysis (see also Sect. 7.3.1)

Attacks on SFLASHv2. The security of SFLASHv2 is exactly the security of an FDH-D scheme based on the one-wayness of the function $\mathsf{f} = (P_1, ..., P_{n-r}) : (\mathbb{F}_q)^n \to (\mathbb{F}_q)^{n-r}$, where the P_j are quadratic polynomials in $\mathbb{F}_q$ generated with a C^{*--} trapdoor.

Necessary conditions for the security of SFLASHv2 are one-wayness and collision resistance of H and one-wayness of f.

Security proof for SFLASHv2. The preimage resistance of the function f is not well defined, but two techniques to find preimages can be described. This is not sufficient for a high level of confidence in the intractability of the C^{*--} problem, but it is sufficient for short term security.

– *Attack on the C^{*--} structure.* This attack [429] requires $\mathcal{O}(q^r)$ operations, which is more than 2^{80} Triple-DES operations for the parameters of SFLASHv2.
– *Resolution of a random set of quadratic equations.* This is called the MQ problem and is NP-hard. However, Gröbner basis finding [110], XL or FXL algorithms [137] or the more sophisticated F_5 and $F_5/2$ algorithms of Faugère [189] are relatively efficient at solving systems of polynomial equations over finite fields.
For the parameters of SFLASHv2, they probably require more computational power than 2^{80} Triple-DES operations.

The hash function of SFLASHv2. In the specifications for SFLASHv2 the value $\mathsf{H}(m)$ is defined to be the first bits of SHA-1$(m)\|$SHA-1(SHA-1(m)). The output of H is clearly not uniform random in $(\mathbb{F}_q)^{n-r}$. A better design is to take the first bits of SHA-1(SHA-1$(m)\|0)\|$SHA-1(SHA-1$(m)\|1$).

Other comments about the design. The value $\mathsf{prf}_\Delta(h)$ is defined to be the first bits of SHA-1$(h\|\Delta)$. This is an acceptable family of pseudo-random functions. HMAC may be preferred.

The affine part of $\mathbf{s}$ and $\mathbf{t}$ can be recovered from the public key alone, and therefore appears to be useless [214, 215].

Side-channel attacks. For a general introduction to side-channel attacks, the reader is referred to the Annex. A. When a message is signed with SFLASHv2, most of the computations are performed in finite fields of characteristic 2. This makes it difficult to imagine, for example, a fault attack where one would change a bit of the secret key and try to use the erroneous signature to recover the original value of the bit. However, other methods might exist to gain information

about the secret key, for instance using differential power analysis (DPA) [495]. The submitters proposed in [8] to mask all the intermediate data with random values.

7.4.4 QUARTZ

QUARTZ is a CPC-D scheme based on the intractability of a variant of the HFE problem.

7.4.4.1 The design

– *Domain parameters.* The parameters are four integers d, n, v and r, a security parameter l, a hash function H with output in $\{0,1\}^{n-r}$ and a family of pseudo-random functions prf with l-bit index, input in $\{0,1\}^{n-r}$ and output in $\{0,1\}^{r+v}$.
The function $\phi : \mathbb{F}_{2^n} \to \{0,1\}^n$ fixes a representation of $\mathbb{F}_{2^n}$ as a $\{0,1\}$-vector space. For any function $\mathbf{F}$ on the finite field $\mathbb{F}_{2^n}$ let $\mathbf{f} = \phi^{-1} \circ \mathbf{F} \circ \phi$.
For QUARTZ we have $l = 80$, $d = 129$, $n = 103$, $v = 4$ and $r = 3$ and H and prf based on SHA-1.
– *Key generation algorithm.* A random affine bijection $\mathbf{s}$ of $\{0,1\}^{n+v}$, a random affine bijection $\mathbf{t}$ of $\{0,1\}^n$, a random family $(\mathbf{F}_V)_{V \in \{0,1\}^v}$ of polynomials over $\mathbb{F}_{2^n}$ and a random l-bit value Δ are generated. They are the private key.
Each polynomial $\mathbf{F}_V$ is randomly chosen from the polynomials of degree at most d whose monomials $\{x^{2^a+2^b}\}_{a,b \geq 0}$ have constant coefficient, $\{x^{2^a}\}_{a \geq 0}$ have coefficient linear in V, and x^0 has coefficient quadratic in V.
The public key is $P = (P_1, ..., P_{n-r})$ where $(P_1, ..., P_n)$ are the quadratic polynomials that describe the function$(\mathbf{t} \circ \mathbf{f}_V \circ \mathbf{s}) : \{0,1\}^{n+v} \to \{0,1\}^n$.
– *Signing and verification algorithms.* QUARTZ is exactly the CPC-D design with 4 rounds based on the trapdoor function $P : \{0,1\}^{n-r} \times \{0,1\}^{r+v} \to \{0,1\}^{n-r}$. The appendix is an element of $\{0,1\}^{n-r} \times (\{0,1\}^{r+v})^4$, with length $n + 3r + 4v = 128$ bits.
The signer can compute P^{-1} because the knowledge of $\mathbf{F}_V$ is sufficient to compute a preimage for $\mathbf{f}_V$ and $P^{-1}(x) \in \{\mathbf{s}^{-1} \circ \mathbf{f}_V^{-1} \circ \mathbf{t}^{-1}(x, R)$ for $R \in \{0,1\}^r$, $V \in \{0,1\}^v$ $\}$. The probability that the preimage does not exist for fixed R and V is approximately $\frac{1}{e}$, so the average number of attempts is e and the probability that the signing algorithm fails is $\frac{1}{e^{128}} \simeq 2^{-185}$.

7.4.4.2 Security analysis (see also Sect. 7.3.1)

General security analysis. The preimage resistance of the function P is not well defined, but two techniques to find preimages can be described.

– *Attack on the QUARTZ structure.* If the degree of $\mathbf{F}_V$ is not bounded, then the function P is a random set of quadratic polynomials, but the signing time is too long. To improve the efficiency of the signing algorithm (which is still very slow) the maximal degree in QUARTZ is set some value d. No known attack directly exploits this difference between the QUARTZ problem and the MQ problem.

– *Resolution of a random set of quadratic equations.* This is called the MQ problem and is NP-hard. However, Faugère [190] showed experimentally that the classical algorithm to solve a system of polynomial equations (Gröbner basis finding) is much more efficient on QUARTZ systems than on general systems.
The impact of this result has been studied by Courtois *et al.* [132] and their conclusion is that setting $d = 257$ increases the computational power of Faugère's attack to 2^{78}. The price to pay is a signing algorithm that is more than 3 times slower.

The hash function of QUARTZ. In the specifications for the first version of QUARTZ [134], the value $\mathsf{H}(m)$ is defined to be the first bits of $h_1 \| h_2 \| h_3$, where $h_1 = \text{SHA-1}(m)$, $h_2 = \text{SHA-1}(h_1)$ and $h_3 = \text{SHA-1}(h_2)$. The revised version of QUARTZ [136] uses the better design that takes the first bits of $\text{SHA-1}(h\|0)\|\text{SHA-1}(h\|1)\|\text{SHA-1}(h\|2)$, where $h = \text{SHA-1}(m)$.

Other comments about the design. The signature generation algorithm in the first version of QUARTZ only accepts the case where $\mathbf{F}_V^{-1}$ has a unique solution. The revised version also accepts the case where two solutions exist and chooses one in a deterministic way. The drawback of the first version is that the appendix space is restricted to the values where $\mathbf{F}_V$ has only one root, and that gives some information about $\mathbf{F}_V$ that might be usable for an attack.

Another drawback of Quartz is that the scheme is malleable (it is not strongly unforgeable) [276].

Side-channel attacks. Side-channel attacks are introduced in Annex A. The signing algorithm of QUARTZ does not use any of the classically vulnerable operations, and we did not find any side-channel attack against it.

7.4.5 RSA-PSS

RSA-PSS as submitted by RSA Labs [275] is based on the PSS design, and its security mainly relies on the intractability of the e-th root problem.

The differences between the submitted RSA-PSS and the generic design described in Sect. 7.3.1.5 of this document and in the original description of PSS [46] are justified in the submission and in supporting documents [275, 271].

7.4.5.1 The design

– *Domain parameters.* The parameters are the bitlength l of the modulus, the output size l' of a hash function $Hash$ and the public exponent $e \geq 3$.
– *Key generation algorithm.* Two random primes p and q are generated. The public key is $n = pq$ and the private exponent is $d = e^{-1} \bmod (p-1)(q-1)$.
– *Signing and verification algorithms.* The scheme uses the PSS design where the trapdoor function is $\mathsf{f}(x) = x^e \bmod n$ and where the two functions H and G are built from the common hash function $Hash$. A hash identifier may be used, and the complete definitions of H and G are given in Sect. 7.4.5.2 below. One consequence of this complete definition is that the trapdoor function f is used only on a subset of $\mathbb{Z}/n\mathbb{Z}$. More precisely, if we define $\mathcal{H}_0 =$

$\{x \in \mathbb{Z}/n\mathbb{Z} \,|\, x = a\|b\|0\text{xBC}\}$, $\mathcal{H}_{\mathsf{Id}} = \{x \in \mathbb{Z}/n\mathbb{Z} \,|\, x = a\|b\|\mathsf{Id}\|0\text{xCC}\}$, $\mathcal{H} = \mathcal{H}_0 \cup \bigcup_{\mathsf{Id}} \mathcal{H}_{\mathsf{Id}}$ and $\mathcal{S} = \{x^{1/e} \,|\, x \in \mathcal{H}\}$, then the NESSIE submission is a PSS construction based on the restriction $\mathsf{f} : \mathcal{S} \to \mathcal{H}$.

Parameter and key generation. While any exponent $e \geq 3$ can be used, the NESSIE submission proposes small values like 3, 17, or 65537. The advantage of choosing a small e is that the verification algorithm is much faster.

The generation of random prime numbers p and q of a given size is crucial to the security of the scheme. The NESSIE submission suggests an algorithm that does a probabilistic primality testing. Other algorithms may be used for key generation.

7.4.5.2 Security analysis (see also Sect. 7.3.1)

Security proof for RSA-PSS. Following the generic security proof for the PSS design, the security of RSA-PSS is based on the clawfreeness of f and $\mathsf{g}(x) = \eta \cdot x^e \bmod n$ for a random η. This is easily shown to be equivalent to the e-th root problem for η.

But because the NESSIE submission is based on a restriction of the function f, the efficiency of the security proof is slightly worse than for the generic PSS design. This is shown by Jonsson [271].

On the hash functions of RSA-PSS. The description of the function H depends on an option t which can be 1 or 2 and decides whether a hash identifier [7] is used or not. If $t = 1$ and $Hash$ is a hash function with $hLen$-byte output, then H has an output of $hLen + 1$ bytes and is defined by $\mathsf{H}(m\|r) = Hash(0_{(8 \text{ bytes})}\|Hash(m)\|r)\|0\text{xBC}$. If $t = 2$ and $Hash$ is a hash function with $hLen$-byte output and identifier Id, then H has an output of $hLen + 2$ bytes and is defined by $\mathsf{H}(m\|r) = Hash(0_{(8 \text{ bytes})}\|Hash(m)\|r)\|\mathsf{Id}\|0\text{xCC}$. The output of the function $\mathsf{G}(h)$ is defined to be the first $\lfloor l/8 \rfloor - hLen - t$ bytes of $Hash(h\|Hash(0_{(4 \text{ bytes})})\|Hash(h\|1_{(4 \text{ bytes})})\|...$ Moreover, the salt r being of length smaller than $\lfloor l/8 \rfloor - hLen - t$ bytes, a constant $\bar{m} = 0...01$ is used.

While this is very close to the generic PSS design, the fact that H and G have variable output length, depending on t, makes a new proof necessary and may introduce subtle weaknesses. Another very similar design is easier to provide with a security proof, if the function H has a fixed output length for each public key. For example, $\mathsf{H}(m\|r)$ can be defined to be the last bytes of $...\|Hash(1_{(8 \text{ bytes})}\|Hash(m)\|r)\|Hash(0_{(8 \text{ bytes})}\|Hash(m)\|r)\|\mathsf{Trail}$, where Trail is either 0x00BC or $\mathsf{Id}\|0\text{xCC}$. The security proof of this variant of RSA-PSS then makes the hypothesis that this function can be viewed as a random oracle (with output in $\mathcal{H}$), which is likely if $Hash$ is one-way and collision-resistant. [8]

Another improvement on the functions H and G can be suggested, which protects against parameter manipulation, especially against adversarial hashing [369]: the inclusion of some commitment to the parameters and the public key in the input of the hash functions, as suggested in Sect. 7.2.1.2.

[7] Some thoughts on hash identifiers have been published by Kaliski [281].

[8] This design may even be better than the simple design $\mathsf{H}(m\|r) = Hash(m\|r)$, which has the extensibility property [109].

One last improvement is the use of UOWHF (Universal One-Way Hash Functions) instead of CRHF (Collision-Resistant Hash Functions). The reason behind this is that there exist secure constructions of families of UOWHF, while no family of CRHF has been found.

RSA problem versus *e*-th root problem. It is important to notice that because the exponent e is a parameter of the scheme the security of RSA-PSS is provably based on the e-th root problem.

Some implementations of RSA-based schemes generate n before choosing e. If e is not randomly chosen from the numbers coprime to $\phi(n)$ (e.g. the implementation of GPG 1.2.1 `www.gnupg.org` chooses the smallest such e in the list 41, 257, 65537, 65539, ...) the security of the scheme is based on another intractability assumption.

However, flexibility in the choice of e can be useful for some RSA-based schemes, e.g. threshold RSA and mediated RSA [157, 94].

It is possible that the e-th root problem for small values of e like 3, 17, or 65537 has not the same intractability as the generic problem. It may be easier, or harder to solve. But small exponents make this problem easier to solve if some side-channel information can be obtained, e.g. some bits of the secret exponent d [96]. Moreover, some other theoretical arguments have been given that suggest that a prime $e \geq 65537$ is a conservative choice [98].

The random nonce. The random value r used in the signing algorithm should have *entropy*, otherwise the efficiency of the security proof decreases. If this value is deterministically generated, then the scheme is equivalent to a RSA-FDH scheme, and has not so tight reduction to the e-th root problem. Even if the deterministic generation of r is not pseudo-random, e.g. if $r = 0$, no concrete weakness of the resulting scheme is known.

Side-channel attacks. A general introduction to side-channel attacks can be found in Annex. A. In [93], Boneh *et al.* stressed how Chinese remainder based implementations of RSA signature schemes are vulnerable to faults. For efficiency, one would compute separately $S_p = x^d \bmod p$ and $S_q = x^d \bmod q$ and then use the Chinese remainder theorem to construct the signature $S = x^d \bmod n$. If an error occurs during only one of the two exponentiations, then an attacker who obtains this faulty signature and the correct one of the same message can factor the modulus. This attack has been improved by Joye *et al.* in [278]: one such erroneous signature and the corresponding plaintext x are enough to find out the entire secret exponent. This attack does not apply to RSA-PSS. Indeed, in this scheme, the message is at first encoded using the PSS encoding method, which introduces some random bits. Thus the user never signs the same message twice, and so the first version of the attack is avoided. Furthermore, given an erroneous signature, the full message x that has been signed cannot be recovered, so the second version of the attack does not work either.

For the same reason, it seems very hard to mount against RSA-PSS a fault based attack (based on [292]) that introduces faults in the secret exponent.

7.5 Digital signature schemes not selected for Phase II

7.5.1 ACE-Sign

7.5.1.1 The design

ACE-Sign is based on the merge of a RAND-secure scheme and of a secure chameleon hash, as described in Sect. 7.3.3.4.

- *Domain parameters.* The security parameters are two integers l (e.g. 160) and l' (e.g. 512). Let H be a hash function with an l-bit output.
- *Key generation algorithm.* Two l'-bit strong primes p and q are randomly chosen, and $n = pq$ is computed. h and x are randomly taken in QR_n and e' is a randomly chosen $(l+1)$-bit prime. The keys are $\mathsf{pk} = (n, h, x, e')$ and $\mathsf{sk} = (p, q)$.
- *Verification algorithm.* The verification of a signed message $(m, e, y, y') \in \mathcal{M} \times \mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ checks that e is an odd $(l+1)$-bit integer different from e'. Then it computes $x' = (y')^{e'} h^{-\mathsf{H}(m)}$ and checks if $x \stackrel{?}{=} y^e h^{-\mathsf{H}(x')}$.
- *Signing algorithm.* A random $y' \in QR_n$ and a random $(l+1)$-bit prime e are generated, and $x' = (y')^{e'} h^{-\mathsf{H}(m)}$ and $y = (x h^{\mathsf{H}(x')})^{1/e}$ are computed. The appendix is (e, y, y').

In fact, ACE-Sign as submitted to NESSIE does not use a collision resistant function H but a family of universal one-way hash functions (UOWHF). The index of the hash function used is added to the signed message. This increases its size but weakens the security hypothesis on the hash.

Moreover, the random prime e is generated by some specific randomised algorithm that output e together with some certification values. The properties of this algorithm are: the probability that the same e is generated twice is negligible, and it is intractable to generate e and the certification without using this algorithm. This modification increases the size of the signed message but allow to reduce the security of ACE-Sign to the RSA assumption in the random oracle model.

7.5.1.2 Security analysis (see also Sect. 7.3.3)

General security analysis. We can apply the general security result for this construction (cf. Sect. 7.3.3.4), but ACE-Sign is also provided with an explicit security proof, under the hypothesis that H is collision resistant (or taken from a family of UOWHF).

If the forgery is (m, e, y, y') we define $x' = (y')^{e'} h^{-\mathsf{H}(m)}$, and for all answers (m_i, e_i, y_i, y_i') to signature queries we define $x_i' = (y_i')^{e'} h^{-\mathsf{H}(m_i)}$. This proof defines three types of forgeries:

- In a type I forgery for some j we have $e = e_j$ and $x' = x_j'$. It leads to a solution of the RSA problem for (n, z, e') with tight reduction. This corresponds to a forgery of the chameleon hash. One can notice that $e = e_j$ is not used in the proof.

– In a type II forgery for some j we have $e = e_j$ but $x' \neq x'_j$. It leads to a solution of the RSA problem for (n, z, e_j) with not so tight reduction. This corresponds to the GUESS forgery of the RAND scheme.
– In a type III forgery for all i we have $e \neq e_i$. It leads to a solution of the strong RSA problem for (n, z) with tight reduction. This corresponds to the FLEX forgery of the RAND scheme.

The random nonces. The random value y' used in the signing algorithm and the randomness needed to generate e only need to be *unpredictible*. These values can be deterministically generated from the message and a secret value, like it is done for the FDH-D design (cf. Sect. 7.3.1.3).

Comments about the design. ACE-Sign is the first of a family of digital signature schemes which have a security proof in the real world and for which all the operations used for signature or verification are efficient. While this design is very promising, ACE-Sign has the drawback of having a not so tight reduction. This is also the case for all the variants of this scheme.

For this reason its advantage over RSA-PSS, namely the fact that the security proof holds in the real world, is only meaningful if parameter size is increased to counteract the non-optimal reduction. For example, to have 80-bit security with $q_S \simeq 2^{32}$, ACE-Sign would need a modulus of 4096 bits, where RSA-PSS only needs 1536 bits. The impact in terms of performance is important.

7.5.2 FLASH

7.5.2.1 The design

This scheme is very similar to SFLASHv2. The only change is in the parameter choice, where $q = 256 = 2^8$ instead of 2^7. See also [427].

7.5.2.2 Security analysis

There is no known difference in security between FLASH and SFLASHv2 and the performance is similar. However, the fact that 7 is prime implies that no other subfield can be found in $\mathbb{F}_{128}$ than $\mathbb{F}_2$, while the FLASH base field $\mathbb{F}_{256}$ can also be seen as a vector space over $\mathbb{F}_{16}$ or $\mathbb{F}_4$. This may introduce additional weaknesses in FLASH.

7.5.3 SFLASH

7.5.3.1 The design

This scheme is very similar to SFLASHv2. The only change is in the secret key choice, where all components of $\mathbf{s}$ and $\mathbf{t}$ are 0 or 1. The advantage of this restriction is that the keys are much shorter than for SFLASHv2.

7.5.3.2 Security analysis

Gilbert and Minier [220] have found how to use this special property to break SFLASH. Their attack is practical, so SFLASH is totally insecure.

Changes from version 1.0 to version 2.0 of the document

—	Typos have been corrected.
§7.1	More on the distinction between parameters and keys. Short reference to hash identifiers.
§7.2.1.1	SO-CMA and RAND are defined. Strong non-repudiation and basic non-repudiation of origin are distinguished.
§7.2.1.2	Reference to [369] is added.
§7.3.1.2	Theorem 7.1 added.
§7.3.2	Reference to extensibility of common hash functions and their suitability as random oracles added [109].
§7.3.2.4	"ElGamal projection" is renamed "Identity projection".
§7.3.2.6	"GPS-sign scheme" is added.
§7.3.2.11	Proofs based on "semilogarithm" are added. They are generalisations of Brown's results [107].
§7.3.3	Rewritten to be more explicit on the underlying design techniques.
§7.4.1.2	Partly rewritten.
§7.4.3.2	References to [214, 215] are added.
§7.4.4.2	Comments about malleability of Quartz corrected.
§7.4.5	Expanded, to answer Burt Kaliski's comments on version 1.0.
§7.5.1	The description is made more readable. Some considerations are moved to Sect. 7.3.3.4. CRHF and UOWHF variants mentioned.

8. Digital identification schemes

We present in this chapter asymmetric techniques to identify one entity (the prover, also called Alice) with regard to another (the verifier, also called Bob), in a way that any attacker (also called Charlie) would most certainly fail to substitute himself to Alice. We will review some standard techniques to achieve this goal, starting from the least secure.

8.1 Introduction

8.1.1 Identification through Password

The most widespread identification protocol is the identification through a password. For instance, under a UNIX system, it works as follows. Every user U chooses a password x_U and sends it to the server. The server applies a hash function f to compute $y_U = f(x_U)$ and then stores (U, y_U). Each time a user wants to access his account, he will send his password x to the server that will then check whether $y_U = f(x)$.

Unfortunately, this identification does not satisfy the modern security requirements. Indeed any eavesdropper will be able to recover the secret password (which is sent unencrypted) therefore causing the scheme to be totally vulnerable to a passive attack (cf. Definition 8.2).

8.1.2 Lamport's Protocol

This protocol is akin to the previous password protocol, with the difference that any password is only used once. The price to pay is a larger storage capacity, or an enhanced computational power. It works as follows.

– Initialisation
  1. Every user U chooses and saves a secret value $x_U = x_0$.
  2. He then computes $x_1 = f(x_0), x_2 = f(x_1), \ldots, x_{1000} = f(x_{999})$, where f is as before a one-way function.
  3. He then publishes $y_U = x_{1000}$ to the server (verifier) who stores (U, y_U).

– Identification

[0] Coordinators for this chapter: UCL — Mathieu Ciet and Francesco Sica

1. When first identifying himself, the prover sends, together with his identity, $x = x_{999}$. The verifier then checks that $f(x) = y_U$ and updates the value of y_U by setting $y_U = x$.
2. Successive identification are carried out as previously, with the prover sending $x_{998}, x_{997}, \ldots$ and so on. Since f is supposed one-way, only the prover is able to produce this sequence of values.

There are two problems with this kind of identification. Since successive values $x_{1000}, x_{999}, \ldots$ become public after doing several executions of the protocol, one cannot safely replay this protocol. In particular, in case of disk crash, if the last value sent (say x_{765}) was not stored, but say only up to x_{900} was stored, anyone having recorded any value between x_{899} and x_{765} will be able to identify himself instead of the prover if the server replays previous identification rounds.

Another issue lies in the fact that this identification scheme can only be used with one verifier (this is a problem for electronic transactions, for instance).

8.2 Security Requirements

8.2.1 Passive Attacks and Interactive Proofs

As we saw in the previous section, a major threat is represented by replay attacks, that are the most dangerous passive attacks. To thwart the possibility of passive attacks, one solution is to make the identification protocol interactive with a series of questions and answers between Alice and Bob. The minimal properties of this protocol against any attack are the completeness and the soundness. They are explained in the following.

Definition 8.1 (Goal of attacker). *An identification scheme is* totally broken *by an attacker if the attacker can impersonate the legitimate prover at will (to any verifier) using his public key.*

Definition 8.2 (Passive and active attacks). *A* passive attack *involves an adversary who tries to break the identification scheme by simply recording data exchanged between prover and verifier and thereafter analysing it. An* active attack *involves an adversary who can deviate from the verifier's protocol in order to extract more information about the secret key.*

Definition 8.3 (Completeness property). *An interactive proof is called* complete *if given a honest prover and a honest verifier, the verifier accepts the prover with overwhelming probability.*

Definition 8.4 (Soundness property). *An interactive proof is called* sound *if whenever Charlie tries to impersonate Alice during the identification protocol, he will fail with overwhelming probability.*

How do we build in practice such interactive proofs of knowledge? Asymmetric cryptography comes to our help: Alice will be the holder of a secret key S_A, with

corresponding public key I_A. This public key is a word of a language $L \in \mathsf{NP}$ (see start of Section 8.2.5 for a definition of NP). The secret key S_A *witnesses* that $I_A \in L$. One chooses L in such a way that finding such a witness is untractable. The identity proof consists in Alice trying to persuade Bob that she possesses such a witness S_A. Also if Charlie wants to successfully identify himself, he will have to know a witness.

We can reformulate the soundness property by making it more precise.

Definition 8.4b (Soundness property). *An interactive proof is called* sound *if there exists an expected polynomial-time algorithm M with the following property: if an attacker impersonating the prover can, with non-negligible probability, successfully execute the protocol and be accepted by the verifier then M can recover a prover's witness using the attacker as a subalgorithm.*

We will now present a few problems on which such interactive proofs are based.

8.2.2 Trusted Hard Mathematical Problems

These problems can be divided into two classes, namely the popular number-theoretic problems (RSA, discrete logarithm, ...) which are not NP-complete but on which much can be written, and proven NP-complete problems (PKP, SD, ..., see below). NP-complete languages L are the natural candidates for identification tasks, because by definition if $L \notin \mathsf{P}$ (supposing $\mathsf{P} \neq \mathsf{NP}$) then checking whether a word x belongs to L cannot be done in polynomial time, whereas the knowledge of a witness (certificate) y allows to do so.

Let us describe these problems (cf Section 6.2.3).

- Let g be an element of a group, and let g have order q. The discrete logarithm problem (DLP) is the problem of finding a when given (g, g^a). The DLP assumption is that the DLP cannot be solved by a polynomial-time (with respect to q) algorithm. We will use the version where the group is the set $(\mathbb{Z}/n\mathbb{Z})^\times$ of invertible residues modulo n.
- Short exponent problem: this is a sub-instance of the discrete logarithm problem. Given two coprime integers n, g, an integer $S << \varphi(n)$ and $g^s \mod n$ the problem consists of recovering the exponent s, knowing that $s \leq S$.
- An RSA key is a pair (n, e) where $n = pq$ with p and q primes, and $1 \leq e < n$ with $g.c.d.(n, e) = 1$. The RSA problem is the problem of finding an integer $1 \leq x \leq n$ such that $x^e = y$ when given an RSA key (n, e) and an integer $1 \leq y \leq n$.
- The e^{th} root problem is similar to the RSA problem but the RSA key is thought of a parameter to the system, i.e. it is the problem of finding, for some fixed RSA key (n, e), an integer $1 \leq x \leq n$ such that $x^e = y$ when given an integer $1 \leq y \leq n$.
- RSA-omi (one more inversion) problem: given two integers n, e such that $\gcd(e, \varphi(n)) = 1$, a challenge oracle outputting a random $y \in (\mathbb{Z}/n\mathbb{Z})^\times$ for each query and a inversion oracle which on input $y \in (\mathbb{Z}/n\mathbb{Z})^\times$ outputs x such

that $y \equiv x^e \pmod{n}$, the problem is for an adversary making t queries to the challenge oracle and getting $y_1, \ldots, y_t$ to find $x_1, \ldots, x_t$ such that $y_i = x_i^e \pmod{n}$ for $i = 1, \ldots, t$ with *at most* $t-1$ queries to the inversion oracle.
- One-more discrete logarithm (omdl) problem: this is defined similarly to the RSA-omi, where the inversion oracle is replaced by a discrete logarithm oracle.
- PKP (Permuted Kernel Problem): Given an $m \times n$ matrix A with entries in $\mathbb{Z}/p\mathbb{Z}$ and a vector $(v_1, \ldots, v_n) \in (\mathbb{Z}/p\mathbb{Z})^n$, find a permutation σ (if it exists) on the set $\{1, \ldots, n\}$ such that $(v_{\sigma(1)}, \ldots, v_{\sigma(n)})$ lies in the kernel of A.
- SD (Syndrome Decoding problem): Given a $(n, n-k)$-linear binary code with parity matrix H and a k-bit vector i, find a minimum-weight solution to the linear equation $H(s) = i$.
- CLE (Constrained Linear Equations problem): This is the problem of finding solutions to a system of linear equations modulo a small prime, when imposing that the solutions must be taken from a specified set of residues.
- PPP (Permuted Perceptrons Problem): Define an ϵ-matrix (resp. vector) to have all entries equal to ± 1. Let A be an ϵ-matrix of size $m \times n$ and S be a set of m nonnegative integers. The problem consists in finding an ϵ-vector V of size n such that the set of the components of the vector AV is equal to S.

8.2.3 Protection against Active Attacks

The reason to base interactive protocols on hard problems is to offer resistance to the more pernicious active attacks. We can adapt a methodology similar to other asymmetric schemes proofs in order to show that a given scheme is secure under active attacks. The following definitions are adaptations of similar Definition 6.2 and the definition found at the beginning of Section 7.2.3.

We precise that an active attacker will usually act as the verifier to Alice for a number of times and then try to impersonate her successfully with other honest verifiers.

Definition 8.5. *A* (t, ε)-solver *for a problem is a probabilistic Turing Machine* $\mathcal{A}$ *that runs in time bounded above by* t *and outputs a solution for the problem with probability at least* ε.

A (t, ε, q_I)-impersonator *for a digital identification scheme is a probabilistic Turing Machine* $\mathcal{A}$ *that runs in time bounded above by* t*, makes at most* q_I *challenges to the prover and succeeds in breaking the scheme with probability at least* ε.

Definition 8.6. *A* proof of security *is the description of a (randomised) algorithm called* reduction algorithm. *This algorithm is a* (t', ε')*-solver for some mathematical problem and interacts with a* (t, ε, q_I)*-impersonator for the identification scheme. A proof of security explains how a solver, using a* (t, ε, q_I)*-impersonator as a subalgorithm, can solve the underlying mathematical problem in time* t' *with probability* ε'*. The proof must relate* t *to* t' *and* ε *to* ε'*. In particular,* $\varepsilon' \leq \varepsilon$ *and* $t' \geq t$.

Thus if the underlying mathematical problem is hard, so that there are no (t', ε')-solver for it, then there cannot exist any (t, ε, q_I)-impersonator of the identification scheme. We say that the identification scheme is (t, ε, q_I)-secure.

Usually $q_I \ll t$, and we will require $k = \log_2(t/\varepsilon) = 80$ (the attacker's computing power is estimated to 2^{80} triple-DES) and $\log_2 q_I = 30$ (the attacker cannot require more than a billion executions of the identification protocol).

However, some algorithms until recently did not have any security proof according to this paradigm, and even the known proofs [43] use some strong underlying assumption, like the RSA-omi for GQ2 or the one-more discrete logarithm for Schnorr.

This is the reason the zero-knowledge property proposed by Goldwasser, Micali and Rackoff [228] is rather used as a shield against active attacks.

8.2.4 Zero-Knowledge

8.2.4.1 The Power of Zero-Knowledge Interactive Proofs

Is it possible for Alice to convince Bob that she knows a secret S_A without revealing anything other than this fact? The theory of zero-knowledge answers in the affirmative. In other words, during a zero-knowledge identification protocol, Alice is effectively sending Bob only one bit of information, namely that she is indeed who she claims she is.

This section is largely inspired from Goldreich's book [222, Chapter 6], to which we refer for more technical details and proofs. Following is a formalisation of Alice and Bob and some rephrasing of concepts already seen. Note that one can view, in light of the preceding sections, an interactive identification protocol as a proof that some word x belongs to a language L.

Definition 8.7 (Loose formalisation of prover and verifier). *Alice and Bob are* interactive Turing machines *(ITM), denoted respectively P and V. Each of these Turing machines has a certain number of tapes, most important of which are a common input tape (read-only), auxiliary input tapes (read-only and specific to each), random seed or coin tapes (read-only and specific to each), work tapes (read-write and specific) and two communication tapes (shared, on one of them P can read and V can write, on the other one V can read and P can write).*

Also, P and V have two states (active and idle) and they cannot be both active (they are both idle only before and after the identification protocol).

Definition 8.8 (Time complexity of ITMs). *Let A and B be two linked ITMs and $t\colon \mathbb{N} \to \mathbb{N}$ a non-decreasing function. Then A has* time complexity t *if for any B and any bit string x, machine A on interacting with B with common input x always (i.e. for any distribution of outputs of tapes specific to B) halts within time $t(|x|)$, where $|x|$ denotes the bit length of x.*

We say A has polynomial-time *complexity if t can be chosen to be a polynomial.*

Definition 8.9. *We denote by* $\langle A(y), B(z)\rangle(x)$ *the random variable consisting of the local output of* B *after interacting with* A*, over all choices of the random seed tapes, with common input* x *and auxiliary inputs* y *for* A *and* z *for* B*.*

Definition 8.10 (Completeness and soundness revisited). *The identification protocol between* P *and* V *which is an interactive proof system for a language* L *is*

– complete *if for every common input* $x \in L$*, there exists* y *such that for any* z

$$\text{Prob}(\langle P(y), V(z)\rangle(x) = 1) \geq \frac{2}{3} \ , \tag{8.1}$$

– sound *if for every* $x \notin L$*, for any ITM* B *for any bit string* y, z

$$\text{Prob}(\langle B(y), V(z)\rangle(x) = 1) \leq \frac{1}{3} \ .$$

Remark. *The numbers* $2/3$ *and* $1/3$ *can be equivalently replaced in this definition by* $1-\epsilon$ *and* ϵ *for any* $0 < \epsilon < 1/2$*. Also, given* $x \in L$*, we denote* $P_L(x)$ *to be the set of* y *satisfying* (8.1)*.*

We come to the main definition of this section.

Definition 8.11 (Zero-knowledge property). *Let* (P, V) *be an interactive proof for a language* L*. We say* (P, V) *is* auxiliary-input zero-knowledge *if for every probabilistic polynomial time interactive machine* V^* *there exists a probabilistic algorithm* $M^*(\cdot,\cdot)$*, running in polynomial time with respect to its first input, so that the following two distributions are indistinguishable:*

– $\{\langle P(y), V^*(z)\rangle(x)\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$ *and*
– $\{M^*(x, z)\}_{x\in L, z\in\{0,1\}^*}$*.*

In this case, M^* *is called a* simulator *of the interaction of* V^* *with* P*.*

What this says is that the interaction of P and V can be simulated polynomially by an algorithm controlled by V, with the same inputs (to V). It is important that this simulation be polynomial-time, as we will later see. Also note that an eavesdropper seeing only the interaction between P and V will be unable to tell whether he is witnessing a real interaction (variable $\langle P(y), V^*(z)\rangle(x)$) or a fake one ($M^*(x, z)$).

In this definition, one has to give a meaning to the term indistinguishable. If the two distributions are truly the same, the property is called *perfect* zero-knowledge. But in practice, giving the limited power an attacker can have, the weaker notion of *computational* zero-knowledge offers enough security.

Definition 8.12 (Computational indistinguishability). *The distributions defined by the random variables* $\{\langle P(y), V^*(z)\rangle(x)\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$ *and* $\{M^*(x, z)\}_{x\in L, z\in\{0,1\}^*}$ *are computationally indistinguishable if for every probabilistic algorithm* $D(\cdot,\cdot,\cdot)$ *running in polynomial time with respect to its first input, every polynomial* p*, all sufficiently long* $x \in L$*, all* $y \in P_L(x)$ *and* $z \in \{0,1\}^*$*, one has*

$$|\text{Prob}(D(x,z,\langle P(y),V^*(z)\rangle(x))=1)-\text{Prob}(D(x,z,M^*(x,z))=1)|<\frac{1}{p(|x|)}\ .$$

Remark. *By zero-knowledge we will henceforth mean computational zero-knowledge.*

Note that for the time complexity of ITMs and hence M^* and D we require polynomial-time only in the common input x, not in the auxiliary inputs z or y which can be very large. If we did, then we would allow for instance D's that could run in exponential time and worse, in practice shredding the notion of computational indistinguishability.

The interesting property of zero-knowledge proofs is that they remain zero-knowledge after polynomially many repetitions of the protocol.

Theorem 8.1 (Closure under sequential composition). *Let (P,V) be an interactive auxiliary-input zero-knowledge proof for a language L. Let Q be a polynomial. Define P_Q to be the sequential running of P with same x as common input $Q(|x|)$ times and independent random tapes (similarly for V_Q). Then (P_Q,V_Q) is an auxiliary-input zero-knowledge proof for L. Moreover if (P,V) is perfect auxiliary-input zero-knowledge, then so is (P_Q,V_Q).*

Remark. *Actually, V plays no role here, because it is clear that the zero-knowledge property is a property of the prover P only.*

One should be aware here that the assertions of the theorem are not necessarily valid if one uses alternative definitions of zero-knowledge, as the original definition of Goldwasser, Micali and Rackoff [228]. In particular, allowing auxiliary inputs is essential in order to get closure under sequential composition.

This theorem is used in practice to construct protocols where the probability of cheating is as low as one desires. For instance, in the Fiat-Shamir protocol (see Section 8.4.1) a legitimate prover is always accepted, but a cheater can be accepted with probability 1/2. Therefore by repeating the basic 3-round protocol a number k of times, one can ensure that a cheater will only be accepted with probability $1/2^k$, while retaining the zero-knowledge character of the protocol.

A very nice result is that all languages in NP have a zero-knowledge proof provided one-way function exist.

Theorem 8.2. *Suppose one-way functions exist. Then every language in* NP *has an auxiliary-input zero-knowledge proof system. Furthermore, the prescribed prover in this system can be implemented in probabilistic polynomial-time, provided it gets the corresponding* NP *witness as auxiliary input.*

8.2.4.2 Negative Results on Zero-Knowledge

The theory of zero-knowledge does not solve all our problems, since we will see that to obtain adequate security, zero-knowledge protocols have to be repeated sufficiently many times and thus lose in efficiency.

We begin by recalling the definition of the complexity class BPP.

Definition 8.13 (BPP). *The class* BPP *consists of all languages L for which there exists a randomised algorithm V such that for all x*

$$\text{Prob}(V(x) = 1 \mid x \in L) \geq \frac{2}{3}$$

and

$$\text{Prob}(V(x) = 1 \mid x \notin L) \leq \frac{1}{3} \ .$$

As before, the numbers appearing in the right-hand sides can be made arbitrarily close to 1 (resp. 0). It is clear that languages in this class have trivial interactive proofs of knowledge, in the sense that the verifier has no interaction with the prover.

Definition 8.14 (Black-box zero-knowledge). *The prover P for the language L is* black-box zero-knowledge *if there exists a universal probabilistic polynomial-time algorithm M which uses any verifier V^* as a black box, such that $\{\langle P(y), V^*(z)\rangle(x)\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$ and $\{M^{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*}$ are computationally indistinguishable.*

Hence the main difference with traditional zero-knowledge definitions is that the simulator here is universal and does not depend on the verifier V^*. In practice, all known proof of zero-knowledge proceed by exhibiting a universal M, so that in the end, one always demonstrates black-box zero-knowledge properties. However this theoretical definition is important because of the following result.

Theorem 8.3. *Suppose that (P, V) is an interactive proof system for the language L with negligible error probability. Suppose also that there exists a constant ρ such that for every $x \in L$, on input x the prover P sends at most ρ messages (constant round), the messages sent by the verifier are* predetermined *consecutive segments of its random tape (public coins or Arthur-Merlin game) and that P is black-box zero-knowledge. Then $L \in$ BPP.*

Thus, except for "trivial" languages, if (P, V) satisfy all the properties above, it must be that the error probability is not negligible, hence to make small, you must increase the number of sequential runs of the protocol. Another conclusion is that in order to construct low error probability zero-knowledge protocols it is wise to allow the verifier to use secret coins.

In the cases that we will consider, namely 3-round protocols, the distinction between public and secret tosses of a coin does not exist (since there is only one random toss), hence we get that there are no 3-round black-box zero-knowledge identification systems with negligible error probability. The number of rounds here is believed to be optimal in the sense that there exist 4-round black-box zero-knowledge proofs for languages believed to be outside BPP and assuming the existence of claw-free permutations (see 7.3.1.1) there exist 5-round zero-knowledge interactive proofs for all languages in NP.

There is one important consequence of this fact. If we consider *parallel* executions of the protocol, then the error probability can be made negligible (for

instance less than $1/2^m$ for m parallel executions of Fiat-Shamir). If (P, V) play an Arthur-Merlin game, then the same is true for the parallel version and the number of rounds stays the same. Hence the theorem implies that this parallel version cannot be black-box zero-knowledge. Although it may still be zero-knowledge, the fact that all known zero-knowledge proofs are actually black-box zero-knowledge proofs is bad omen. Indeed the following was proved.

Theorem 8.4 (Non-closure under parallel composition). *There exist two zero-knowledge provers P_1 and P_2 such that the prover P which runs both of them in parallel leaks knowledge.*

This and the fact that zero-knowledge protocols are expensive (in terms of performance) has led to consider a larger class of protocols that includes zero-knowledge protocols but can still be used for secure identification.

8.2.5 Witness Indistinguishability

Let L be a language of NP. By definition, this means that there exists a binary relation, called *witness relation*, R_L such that $(x, y) \in R_L$ implies $|y| \leq p_L(|x|)$ for some polynomial p_L. Also, $(x, y) \in R_L$ can be checked in polynomial-time and

$$L = \{x \colon \exists y \colon (x, y) \in R_L\} \ .$$

We call y a witness to $x \in L$. In general, $x \in L$ may have more than one witness, so that the cardinality of $R_L(x) = \{y \colon (x, y) \in R_L\}$ is larger than 1. We say an interactive proof for L is witness-indistinguishable if after running the protocol, the verifier cannot tell which witness the prover used as auxiliary input. Formally,

Definition 8.15 (Witness indistinguishability/independence). *Let $L \in \mathsf{NP}$ with a witness relation R_L, (P, V^*) an interactive proof for L, where V^* is any polynomial-time ITM . Denote by* $\mathrm{view}^{P(y)}_{V^*(z)}(x)$ *a random variable describing the contents of the random-tape of V^* and the messages V^* reads on the communication tape (written by P), on common input x, auxiliary input y (for P) and z (for V^*). We say P is* witness-indistinguishable *if for every V^* and every two sequences $(w^1_x)_{x\in L}$ and $(w^2_x)_{x\in L}$ such that $w^i_x \in R_L(x)$, the following two distributions are computationally indistinguishable*

- $\{x, \mathrm{view}^{P(w^1_x)}_{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*}$ *and*
- $\{x, \mathrm{view}^{P(w^2_x)}_{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*}$.

If the two distributions are the same, then we use the term witness-independent.

Note that any zero-knowledge proof for a language in NP is witness-indistinguishable, since the view corresponding to each witness can be approximated by the same simulator. Likewise, perfect zero-knowledge proofs are witness independent.

Theorem 8.5 (Closure under sequential and parallel composition). *The sequential composition of witness-indistinguishable (resp. witness-independent) provers is witness-indistinguishable (resp. witness-independent).*

Let $L \in \mathsf{NP}$, R_L a witness relation, P a witness indistinguishable (resp. witness-independent) prover for R_L that runs in probabilistic polynomial time. Let Q be a polynomial and P_Q denote the ITM that on common input $x_1, \ldots, x_{Q(n)} \in \{0,1\}^n$ and auxiliary input $y_1, \ldots, y_{Q(n)} \in \{0,1\}^*$ invokes P in parallel $Q(n)$ times, so that in the i^{th} copy P is invoked with common input x_i and auxiliary input y_i. Then P_Q is witness-indistinguishable (resp. witness-independent) for the relation R_{L^Q} such that $\big((u_1, \ldots, u_{Q(n)}), (v_1, \ldots, v_{Q(n)})\big) \in R_{L^Q}$ iff $(u_1, v_1) \in R_L, \ldots, (u_{Q(n)}, v_{Q(n)}) \in R_L$.

Remark. *It is important that P be probabilistic polynomial-time.*

Theorems 8.2 and 8.5 together with the observation that zero-knowledge implies witness-indistinguishability can be put together to arrive to the following result.

Theorem 8.6. *Assume there exist one-way functions, then every language in* NP *has a* constant round *witness-indistinguishable proof system with* negligible error probability*. In fact, the error probability can be made exponentially small.*

8.2.6 Resettable Zero-Knowledge Proofs

The notion of resettable zero-knowledge was introduced in [113] to provide natural solutions to physical problems in the implementation of zero-knowledge protocols, for instance when the prover is implemented on a smart card that can be reset by simply disconnecting its power supply.

Also this stronger property offers security (is closed) under concurrent (parallel) executions of the protocol, something we have seen to be false for the plain zero-knowledge property.

Finally, this notion provides an alternative way of constructing identification schemes that are fundamentally different from the ones constructed following the Fiat-Shamir paradigm, see Section 8.3.1.

We will not discuss at length this property, since it is not satisfied by any of the submissions or the existing standards. We mention it for completeness and because of its practical relevance.

Loosely speaking, a resettable zero-knowledge proof is an interactive proof where the prover is forced to use a random but *fixed* coin in a polynomial number of executions, each time interacting with the verifier in the usual fashion. The verifier can of course send messages that are related from one execution to another. If the output of the verifier is indistinguishable from a polynomial-time simulator (that depends only on the common input, previously denoted by x), we say the proof is *resettable zero-knowledge* (rZK).

Analogously, one can define *resettable witness-indistinguishability* (rWI). The main results are the following.

Theorem 8.7. *Under the DLP assumption, there is a non-constant round (resp. constant round) rZK (resp. rWI) proof for* NP.

Hence we can do, under classical assumptions, everything previously written in a resettable fashion.

8.2.7 Classification of Attacks

In the following, we give the list of possible attacks against identification schemes. Notice that the attacks are similar to those for message authentication codes.

Distinction is sometimes made between adversaries based on the type of information available to them.

Definition 8.16. *An* outsider *is an adversary who has no special knowledge beyond that generally available, e.g. by eavesdropping on protocol messages over open channels.*

An insider *is an adversary with access to additional information, (e.g. commitment of prover, see Section 8.3.1). He can be a* one-time insider*, if he obtains such information at one point in time for use at a subsequent time or a* permanent insider *if he has continual access to privileged information.*

We list some particular active attacks on identification schemes.

Concurrent attack. A concurrent attack is an active attack whereby the attacker interacts with several copies of the same prover (same secret key) using different commitments g in the notation of Section 8.3.1. This attack is realistic when identification protocols are used in the context of Internet.

Man-in-the-middle attack. A man-in-the-middle attack is an attack whereby an intruder will communicate anonymously with both the verifier and the prover in such a way as to cut off the direct information exchange between them. As a result the (honest) verifier and prover will think to have successfully executed the identification protocol while in fact they will have leaked privileged information that may prove fatal to the prover.

Replay attack. A replay attack is an attack whereby the attacker is able to achieve his goal by using some previous honest execution of the protocol, either with the same verifier or a different one.

Interleaving attack. An interleaving attack is an attack whereby the attacker uses a selective combination of information from several previous honest protocol executions. A replay attack is a type of interleaving attack.

Reflection attack. A reflection attack is an interleaving attack whereby the attacker passes all queries from the verifier to the originator of the honest information and forwards all replies from this entity back to the verifier.

Forced delay attack. This attack occurs when an attacker intercepts a message and relays it at some later point in time.

Chosen text attack. This is an attack whereby an attacker impersonates the verifier and chooses messages in such a way as to obtain information about the prover's key.

Reset attack. In this type of attack the attacker is able to reset the prover to a previous state, for instance by forcing it to always use the same random seed (coin). This is a very powerful physical attack that can be implemented easily on smart cards deprived of internal power supply. Note that resistance against reset attacks, in a slight generalisation of Section 8.2.6 implies resistance to concurrent attacks defined above.

8.2.8 Assessment Process

The submitted asymmetric identification scheme is assessed with respect to generic common identification schemes and specific attacks. To assess the security of an asymmetric identification scheme which is a zero-knowledge protocol we will follow the Feige-Fiat-Shamir methodology in proving completeness, soundness and the zero-knowledge property.

8.3 Overview of Common Designs

8.3.1 Interactive 3-round Identification Protocols

An identification protocol relying on a zero-knowledge proof of knowledge usually follows the four steps which we will describe. This paradigm is often called the Fiat-Shamir paradigm, since it was first described in the Fiat-Shamir protocol (see Section 8.4.1) for knowledge of a square root modulo n.

This kind of protocols is commonly called interactive 3-round identification protocol.

- Commitment: the prover randomly selects a *commitment* g (also called *coin*) and sends it under a mask x to the verifier.
- Challenge: the verifier chooses a random *challenge* c and sends it to the prover.
- Response: the prover uses his secret key together with g to compute a response value y, which is sent back to the verifier.
- Verification: the verifier uses y together with x to check the identity of the prover. He may either accept or reject.

There is also a need when using an asymmetric proof of identity to get assurance of the validity of a public key of the prover. This is the role of the *trusted authority* (TA) which is always supposed honest: the prover chooses a secret key and computes the related public key and the TA certifies his choice. Some of the schemes such as GQ, GPS and Feige-Fiat-Shamir (but not GQ2 or Schnorr) can be made *identity-based*, meaning that the public key is related very closely to the prover's identity (e.g. his email address) and his secret key is then computed by the TA (who hold of some super-secret unknown to all provers) who then passes it on to the prover.

Note that it is obvious in all the following protocols that these are vulnerable to reset attacks and there is nothing we can do about it, since this is a common weakness to all zero-knowledge proofs of knowledge (by definition, which we do not recall here, see [222, Chapter 6]).

8.3.2 Current standards

– ISO/IEC 9798-5 specifies three identification techniques: one family based on the RSA problem, of which Fiat-Shamir and GQ are instances, one family based on the discrete logarithm problem, of which Schnorr is an example and a mechanism based on an asymmetric encryption scheme, derived from the Brandt-Damgaard-Landrock-Pedersen scheme [104].

8.4 Digital identification schemes considered during Phase II

8.4.1 Fiat-Shamir

We describe the Fiat-Shamir protocol [193], historically the first zero-knowledge protocol and an example emulated by later schemes.

– Initialisation
  1. The TA chooses a large $N = pq$, with p and q primes of size $n/2$ and publishes N.
  2. Alice chooses her secret key $S \in (\mathbb{Z}/N\mathbb{Z})^*$ and computes the public key $I = S^2 \bmod N$.
– Identification
  1. Alice chooses a random number $r \in_R (\mathbb{Z}/N\mathbb{Z})^*$, computes $x = r^2 \bmod N$ and sends x to Bob.
  2. Bob chooses a random bit $b \in_R \{0, 1\}$ and sends it to Alice.
  3. Alice then computes $y = rS^b \bmod N$ and sends it to Bob.
  4. Bob checks whether $y^2 = xI^b \bmod N$.

As explained after Theorem 8.1, one has to repeat this protocol sequentially k times to obtain a cheating probability of $1/2^k$. Hence it is important that k grows faster than any expression $C \log n$ for any constant $C > 0$ to achieve super-polynomial security, but must remain less than $n^{C'}$ for some $C' > 0$, in order to preserve the zero-knowledge property (see Theorem 8.1).

Theorem 8.8. *The sequential Fiat-Shamir protocol is an interactive proof of knowledge of a square root of I. It is sound, complete and zero-knowledge.*

8.4.2 Schnorr

The prototype of many identification schemes is Schnorr's identification scheme, which we will describe briefly. A TA publishes

– two primes p, q, such that q divides $p-1$ and such that the discrete log problem is difficult to solve in $\mathbb{F}_p^*$ (typically $\log_2 p = 1024$ and $\log_2 q = 160$),
– an element $g \in \mathbb{F}_p^*$ of order q and
– a security parameter t such that $q > 2^t$.

Each prover chooses $s \in \{1, \ldots, q-1\}$ at random and computes $I = g^{-s} \pmod{p}$, publishing I through the TA as their public key. Let us examine a protocol round.

1. *Commitment*: the prover picks $r \in \{0, \ldots, q-1\}$ at random and sends to the verifier $x = g^r \pmod{p}$.
2. *Challenge*: the verifier chooses a random $c \in \{1, \ldots, 2^t\}$ and sends it to the prover.
3. *Response*: the prover computes $y = r + sc \pmod{q}$ and sends it to the verifier.
4. *Verification*: the verifier computes $z = g^y I^c \pmod{p}$ and accepts the prover if and only if $z = x$.

Theorem 8.9. *The Schnorr protocol is an interactive proof of knowledge of the discrete logarithm of I to the base g in $\mathbb{Z}/p\mathbb{Z}$. It is sound, complete and zero-knowledge for fixed t.*

Note that trivially if Charlie tries to impersonate Alice, his probability of fooling Bob is at most $1/2^t$. However, if one lets t grow super-polynomially, then the protocol cannot be simulated in polynomial time and thus cannot be zero-knowledge. Hence to decrease the error probability while keeping adequate security against active attacks, one must play this protocol several times sequentially.

8.4.3 GPS

GPS is the only identification scheme submitted to NESSIE. The scheme has good performance with high security. The submitted documents contained some minor flaws in the specifications, but these were corrected at the beginning of phase II.

8.4.3.1 The Design

GPS scheme consists of an interactive zero-knowledge identification scheme, that combines provable security based on integer factorisation and computing discrete logarithms modulo prime numbers, identity-based short key and minimal on-line computation.

It is essentially a modified version of the well-known Schnorr identification scheme (see Section 8.4.2). Unlike the Schnorr scheme, GPS uses a generator g with unknown order and the exponent is calculated in $\mathbb{Z}$ rather than modulo p.

The GPS identification scheme consists of ℓ iterations of an identification round. This ℓ is part of the security parameters of the scheme. We now describe one round of the GPS identification scheme. Let A, B, S be parameters with $|A| \geq |S| + |B| + 80$, $|B| = 32$ and $|S|$ greater than 140 bits. It would be better if $|S| = 180$ meaning A is approximately a 300-bit number. Let n be a RSA modulus ($n = pq$ where p, q are 512-bit primes).

Prover		**Verifier**
choose $r \in \{0, \ldots, A-1\}$ compute $x = g^r \pmod n$	$\xrightarrow{x}$	
	$\xleftarrow{c}$	choose $c \in \{0, \ldots, B-1\}$
check $c \in \{0, \ldots, B-1\}$ compute $y = r + cs$	$\xrightarrow{y}$	check $y \in \{0, \ldots, A + (B-1)(S-1) - 1\}$ accept if $g^y I^c = x \pmod n$

This scheme is proved complete, sound (a prover accepted with probability greater than $1/B^\ell$ must know the discrete logarithm of I), and perfectly zero-knowledge and honest-verifier zero-knowledge if B is not too large, see Section 8.4.3.2.

GPS is designed to be used in situations where authentication has to be done "on the fly" with smart cards.

8.4.3.2 The Security of GPS

The submitters show that GPS is complete, sound and zero-knowledge. More precisely they prove the following three assertions [438].

Theorem 8.10 (Completeness). *The execution of the protocol between a prover who knows the secret key corresponding to his public key and a verifier is always successful.*

Theorem 8.11 (Soundness). *Assume some adversary is accepted in polynomial time with non-negligible probability by honest verifiers, that* $\log(|n|) = o(\ell \cdot |B|)$ *and that* ℓ *and* B *are polynomial in* $|n|$*. Then there exists a polynomial-time algorithm that solves the discrete log with short exponent problem.*

Theorem 8.12 (Zero-knowledge). *The GPS protocol is computationally zero-knowledge if* ℓ *and* B *are polynomial in* $|n|$ *and* $\ell SB/A$ *is negligible. As a consequence it is also honest-verifier computationally zero-knowledge under the same assumptions.*

In [439], the authors actually relate the soundness of the protocol to the factorisation of the modulus n, under the same hypothesis.

It may be remarked that the proof of the zero-knowledge property assumes that $|B|$ is constant, as in the proof for the original Schnorr protocol. On the other hand, Pointcheval [434] proves that under some hypothesis on n and g, assuming $|B| > 2 \operatorname{ord}(g)$, GPS still retains its witness indistinguishable property, and active attacks are then related to the factorisation of n. This leads to more efficient and secure identification protocol since one can then safely take $\ell = 1$.

8.4.3.3 Various Attacks on GPS [478]

Verifier Cheating over Value of *c*. In the original version of GPS the response did not include any check that $c < B$. If the verifier is dishonest then he could send $c = A$ as a challenge to the prover. The prover then computes $y = r + sA$ and sends this value to the verifier. Since (assuming the prover is following the protocol correctly) $r < A$, the verifier can easily compute

$$s = \left\lfloor \frac{y}{A} \right\rfloor \quad .$$

This problem was pointed out by Daniel Bleichenbacher (see NESSIE forum) and the protocol was consequently modified.

Breaking the Pseudo-Random Number Generator. Suppose that the pseudo-random number generator used to generate r in the commitment step is weak, so that given k random values $r_1, r_2, \ldots, r_k$ we can predict r_{k+1}. Then the verifier can break the system by sending $c = 0$ k times in order to find the values $r_1, r_2, \ldots, r_k$ and thus predict the value of r_{k+1}. On the $(k+1)$-th application of the algorithm, the verifier can send an arbitrary value of c and then compute the secret key s which is given by

$$s = \frac{y - r_{k+1}}{c} \quad .$$

Alternatively, suppose that the prover is dishonest and wishes to pass himself off as the holder of private/public key pair (s, I) without actually knowing the private key s. If such a dishonest prover could break the pseudo-random number generator used to generate the challenge c then he could fool the verifier as follows:

- the dishonest prover sends the commitment $x = g^{r+c} I^c$ to the verifier. The verifier sends the (not so) random challenge c to the prover.
- The dishonest prover sends $y = r + c$ to the verifier. The verifier checks that $x = g^y I^c = g^{r+c} I^c$ and that $y \in [0, A + (B-1)(S-1)]$.

Both of these checks will be accepted as correct by the verifier who will then assume that the dishonest prover is in fact the holder of the private key s.

Therefore GPS is only as safe as long as the pseudo-random number generator used is unbroken.

8.4.3.4 Fault and Chosen-Modulus Attacks on GPS [174, 173]

The first fault attack targets the secret key and requires the following fault model: bit flip fault model, complete control on the number of faulty bits induced and complete or loose control on the fault location. The fault can be induced before the computation starts. The idea is to flip exactly one bit of the secret key used by the prover. With his response to the challenge, an attacker can recover the original value of the bit he flipped. Repeating this operation, he can recover all the bits of the secret key. This attack has been introduced by [93] and is fully described in [174]. If the attacker does not know exactly which bit of the secret key he flipped, he can make several tries and find out which one it was and its original value. Obviously the attack is then less efficient.

The second fault attack targets an intermediate value and works in the following model: bit flip fault model, complete control on the number of faulty bits induced, and complete or loose control on the fault location. The value targeted is namely the random value r chosen by the prover at the first step. The prover has to store this value as he is going to use it again when responding to the verifier's challenge. If an attacker is able to swap exactly one bit of this value while

the prover is waiting for the challenge, and to repeat this operation several times sending the same challenge to the prover, the attacker can recover the secret key s. The attacker must have at least the following control on the fault location: the fault concerns the value r. If he has complete control, so that he knows exactly which bit has been flipped, the attack is more efficient. This attack has been first described against Schnorr's identification scheme in [93], and the version against GPS is described in [174].

GPS is also vulnerable to a chosen-modulus attack. Indeed, the modulus n is needed by the prover, so a modification of this value can lead to an attack. An attacker who is able to replace this value by a value of his choice can recover the secret key. The first step of the protocol consists in the prover choosing a commitment r and sending the value $g^r \mod n$ to the verifier. If the attacker has replaced the original modulus by a number so that he can easily solve the discrete logarithm in the new field, he can thus find the value r and so the secret key.

8.4.4 GQ

In this section we sum up what the research community already knows about the security of the GQ protocol. We will see that GQ satisfies the following properties: completeness, soundness, perfect zero-knowledge and security against concurrent attacks under the RSA-omi assumption. We will then discuss more practical aspects of the security of GQ, namely we see what attacks on RSA apply to GQ, then we notice that GQ is immune from side-channel attacks on exponents.

8.4.4.1 The Design

The GQ protocol was first published in [234]. It is of the same design as the Fiat-Shamir identification scheme. It provides security based on the RSA assumption, identity-based public keys, and allows the use of the same modulus by multiple users. It needs a trusted authority to generate some of the parameters.

Public Parameters. Let n be a RSA modulus, and v a *prime* RSA exponent. These parameters can be shared by several users.

Parameters of the authority. The factorisation of n and the integer s such that $v.s \equiv 1 \pmod{\varphi(n)}$ are kept secret by the authority.

Parameters of the users. Each user's private key is an integer Q, and the corresponding public key is an integer G verifying $GQ^v \equiv 1 \pmod{n}$ (the equation $G \equiv Q^v \pmod{n}$ can also be used).

Remark. *The use of the second equation only affects the verification step.*

A round of GQ.

Prover		Verifier
choose $r \in (\mathbb{Z}/n\mathbb{Z})^*$ compute $R = r^v \pmod n$	$\xrightarrow{R}$	
	$\xleftarrow{d}$	choose $d \in \{0, \ldots, v-1\}$
check $d \in \{0, \ldots, v-1\}$ compute $D = rQ^d \pmod n$	$\xrightarrow{D}$	check $D \in (\mathbb{Z}/n\mathbb{Z})^*$ accept if $R \equiv D^v G^d \pmod n$ (or $RG^d \equiv D^v \pmod n$ if second equation is used)

8.4.4.2 The Security of GQ

The following properties of GQ have been proved.

Theorem 8.13 (Completeness). *The execution of the protocol between a prover who knows the secret key corresponding to his public key and a verifier is always successful (it is obvious, see [234]).*

Theorem 8.14 (Soundness). *Assume some adversary knows a commitment such that he can succeed in an identification with probability greater than v^{-1}, then this cheater can recover the real prover's private key in polynomial time. This proof was first published in [233].*

Theorem 8.15 (Zero-knowledge). *The GQ protocol is perfect zero-knowledge if v is polynomial in n (this was shown in [111]). As a consequence it is also honest-verifier computationally zero-knowledge under the same assumption.*

Theorem 8.16. (Security against Impersonation under Concurrent Attacks). *The GQ protocol is secure against impersonation under concurrent and active attacks, if the RSA-omi problem is hard. This result was published in [43].*

8.4.4.3 Practical security problems

RSA-related security problems. The GQ protocol requires the use of a RSA modulus, and the private GQ key (or its inverse, depending upon which version is being used) is the RSA signature of the public GQ key. This implies that the choice of the modulus n and the private exponent s must be done with the same care than in RSA.

- The modulus n must be large enough to prevent elliptic curve factorisation
- One can wonder if Wiener's attack [520], Boneh-Durfee's attack [95], or Blömer-May's attack [87] apply to GQ. It could in theory, but it will not work in practice. These attacks work on small secret exponents (i.e., $s < n^{\frac{1}{4}}$ for Wiener's attack, $s < n^{0.292}$ for Boneh-Durfee's attack and $s < n^{0.29}$ for Blömer-May's attack). But the secret exponent cannot be small in GQ because $v.s > \varphi(n)$ and v is small. For example, if we take $|n| = 1024$ and $|v| = 16$, we have $|s| > 1000$.
- Since the factorisation of n is not known by the prover, she can't use the CRT function, and the fault attack on chinese remainder based implementation [93] will not work on GQ.

A remark about side channel attacks. An attacker might try to find some information on the private key by using a side channel attack to recover one or both of the exponents used by the prover. In this paragraph, we notice that even if the attacker recovers both of the exponents it doesn't give him any advantage.

Let us give at look at the computations made by the prover during a proof (notice that the first computation can be made in advance).

- *Commitment :* The prover calculates $R = r^v \bmod n$. The exponent used here, v, is public.
- *Answer :* The prover calculates $D = rQ^d \bmod n$. The exponent used here, d, is the challenge and is not secret (more precisely, an eavesdropper can see it during the proof).

Recovering one of the exponents, or both of them, will not provide an attacker any additional information about the private key.

Changes from version 1.0 to version 2.0 of the document

—	The first part of the chapter was completely rewritten. Sections 8.1.1 through 8.2.6 reflect this new material. Older sections were incorporated into Sect. 8.2.1 and Sect. 8.2.3 (definitions).
§8.2.2	expanded to include proven NP-complete problems PKP, SD, CLE, PPP as well as omdl (one more discrete log problem).
§8.2.7	Added reset attacks.
§8.3.1	one paragraph removed. Clarification that ZK proofs of knowledge are susceptible to reset attacks.
§8.3.2	developed.
§8.4.1	added: description of Fiat-Shamir identification scheme.
§8.4.3.2	first two paragraphs omitted and writing of new security considerations after the three theorems.
§8.4.3.4	is new material.
—	Typos were corrected and description of protocols were uniformised to some extent.

A. Side-channel attacks

Ever since cryptography became part of our every day's life, not only the used cryptographic algorithms became subject to attacks, but also their implementations in hard- or software. The traditional cryptographic model however, does not take the aspect of implementation attacks into account. In the traditional scenario, Alice and Bob secure their communication by some mathematical function, namely the cryptographic algorithm. The adversary Eve, is only assumed to have knowledge about this mathematical function and some plain- and ciphertext pairs. Consequently, security proofs only involve these components. Despite the given proofs of security for any cryptographic algorithm in any theoretical model, a communications system based on this algorithm can still be vulnerable to quite a number of other attacks. A very dangerous class of such attacks is commonly referred to as side-channel attacks and this appendix is devoted to the consideration of the NESSIE primitives towards their vulnerabilities to such attacks and their properties and abilities to help defeating such attacks.

Currently there is very little theoretical framework in which one can assess such attacks. The NESSIE PIB members have stated that only if a side-channel attack applies, regardless of implementation, should this be used as a selection criterion. Unfortunately, some of the currently known side-channel attacks work *regardless* of the specific implementation, and even worse, they do not assume an attacker to have knowledge about the attacked device. Furthermore these attacks work also for all currently known algorithms. Apparently, the selection criterion should not be the applicability of these types of attacks. On the contrary, an algorithms abilities to counteract them should be a selection criterion instead. This is also what we concentrated on in our research towards side-channel attacks within the NESSIE project. The results of this research has been collected in several survey articles In section A.1 we deal with the passive types of side-channel attacks which are also well known as information leakage attacks. Such attacks do not require to actively manipulate the computation, but only monitor the side-channel leakage during the computation. First, we introduce the currently exploitable side-channels in section A.1.1. Section A.1.2 deals with simple side-channel attacks on implementations of symmetric and asymmetric primitives, while section A.1.3 discusses differential side-channel attacks on implementations of symmetric and asymmetric schemes. In section A.2 we deal with active side-channel attacks. As can be deduced from their name, this type of side-channel attacks assumes an

[0] Coordinator for this appendix: KUL — Elisabeth Oswald

attacker actively manipulating the execution of a cryptographic algorithm. Both sections conclude our research and give some general recommendations about selecting and implementing algorithms. Information which is specifically related to the NESSIE primitives can be found directly in the sections devoted to the individual primitives, in the main part of this document.

A.1 Passive Side-Channel Attacks

Passive side-channel attacks were published by Kocher in [312] for the first time. In this first article, timing information was used to gain knowledge about the secret key of implementations of the RSA, DSS, and other cryptosystems. The attack described in this article required an attacker to be able to simulate or predict the timing behaviour of the attacked device rather accurately. The second article by Kocher *et al.* [313] presented a similar, but far more dangerous attack. This second article introduced the usage of power consumption information to determine the secret key. Due to its statistical nature, one of the attacks in [313] is called *differential* power analysis. Another type of side-channel information was introduced only recently. The first articles, [445] and [213], about the usage of electromagnetic emanations were presented in 2000 and published in 2001.

A.1.1 Types of Information Leakage

We shortly discuss the different types of information leakage that have been exploited by attacks which have been published in the open literature so far.

Execution Time Leakage. Often, a device takes slightly different amounts of time to execute an algorithm. Explanations for this behaviour include different input data which might cause some instructions to take different amounts of time for their executions, performance optimisations or branching instructions. Practical implementations for attacks using this kind of information leakage, such as [165] and [250], indicate that such attacks are challenging to realize in practice due to the difficulty of measuring the real execution time. In many modern processors, even on smartcards, instructions can be cached and so the execution time is more and more related to other influences.

Countermeasures appear to be easy to implement, and to work efficiently in practice. Since their first introduction, most work has been dedicated to the exploitation of side-channels with a higher amount of information.

Power Consumption Leakage. Most commonly used cryptographic devices are implemented in CMOS (complimentary metal-oxide semiconductor) logic. The power consumption characteristics of CMOS circuits can be summarised shortly as follows. Whenever a circuit is clocked, the circuits gates change their states simultaneously. This leads to a charging and discharging, resp., of the internal capacitors and this in turn results in a current flow which is measurable at the outside of the device. The measurements can be conducted easily. One needs

either a data acquisition card or a digital oscilloscope to acquire the measurements. The current flow can be measured directly with a current probe, or by putting a small resistor in series with the devices ground-input or power-input. Power analysis attacks are the most popular attacks at the time of writing due to their effectiveness and simplicity. While the first publication [313] was mainly concerned with power-analysis attacks on secret-key cryptosystems, Messerges *et. al.* [368] presented such an attack for public-key cryptosystems. In [48] and [7], the methods of some power-analysis attacks are refined. An introduction to the practical aspects of power-analysis attacks can be found in [6].

Electromagnetic Radiation Leakage. The same charging and discharging which occurs whenever a circuit is clocked creates besides the current flow also a certain electromagnetic (short EM) field. *Direct emanations* are caused by intentional current flow which is caused by the execution of an algorithm. *Unintentional emanations* are caused by the miniaturisation and complexity of modern CMOS devices. This miniaturisation and complexity results in coupling effects between components in close proximity. EM attacks are becoming more and more popular at the time of writing because of the high amount of information of this side-channel and because due to the fact that the information can be exploited also in a larger distance to the attacked device [5].

Error Message Leakage. An error message attack usually targets a device implementing a decryption scheme. We make the assumption that there is a one-bit feedback from the device to tell whether or not the message has been successfully decrypted. If the attacker can somehow know the reason why the decryption operation failed, he might gain some information about the secret key or a plaintext by sending well chosen ciphertexts to the device. An attack exploiting this side-channel has been published in [512].

Combining Side-Channels. Not much research has been done on this topic so far. However, the following simple observation has been used for attacks. Timing attacks can suffer from the difficulty of obtaining precise measurements. Attacks are even more difficult whenever only one intermediate operation is targeted. In such a case, power measurements lead directly and more precisely to the timing of the intermediate operation if this intermediate operation is visible in the power consumption trace. Besides in [463], such an attack is also presented in [515].

A.1.2 Simple Side-Channel Attacks

All attacks presented in this section have been performed with power consumption leakage information so far. A *trace* refers to a measurement taken for one execution of the attacked cryptographic operation. In a simple side-channel attack, only one single measurement is used to gain information about the devices secret key. Obviously, to perform such an attack, the side-channel information needs to be strong enough to be directly visible. Additionally, the secret key needs to have some simple, exploitable relationship with the operations visible in the side-channel trace. Such an attack typically targets implementations which use key dependent branching in the implementation.

A.1.2.1 Attacking Implementations of Symmetric Schemes

A special kind of simple power-analysis attacks, the so called *Hamming weight attacks* exploit a strong relationship between the Hamming weight and the power-consumption trace. In [72] such an attack is presented on an implementation of the DES algorithm and in [347] such an attack is presented on an implementation of the AES algorithm. For these types of attacks it is vital that the implementation is based on relatively small data-words such as for example, in an 8-bit implementation. Usually, this type of attack is applied to implementations of ciphers with a simple key schedule. For implementations that try to achieve a protection against first-order differential power-analysis attacks, this method can be used to determine information on the used mask. *Collision Attacks* on implementations of the DES algorithm have been examined in [468]. Internal collisions are detected by their power trace in these attacks.

A.1.2.2 Securing Implementations of Symmetric Schemes

To counteract the type of simple power-analysis attack that uses Hamming weight information, a designer has to assure that the Hamming weight information which is leaked is not correlated with the intermediate values that are processed. In dedicated hardware implementations this can be achieved by using a special logic-style or by masking intermediate values (this can be achieved by bus encryption as well as by masking the operations of the algorithm in general). In software implementations the intermediate values have to be masked. It is imperative to implement a decent masking scheme to counteract attacks such as presented in [120] and [126]. Probably a good noise generator on chip can also help to counteract such attacks, at least in the case of a power attack.

A.1.2.3 Attacking Implementations of Asymmetric Schemes

Scenarios in which simple side-channel attacks are a possible threat have been considered in [424] and can be summarised as follows. If a *multiplication of a known and a secret value* has to be calculated, then a simple side-channel attack is theoretically possible, but unlikely to work in practice. An *exponentiation of a known with a secret value* is also in principle vulnerable to simple side-channel attacks. The practical feasibility of the attack is heavily dependent on the implementation. Unprotected *Scalar multiplications* of a known elliptic curve point by an unknown scalar are highly vulnerable to this kind of attack regardless of the underlying hardware. Also, implementations based on addition-subtraction chains can leak enough information to recover the private key [423]. Considerations of the security of implementations of S-Flash and Quartz can be found in [8]. In Klima *et al.* [292] a variant of a Hamming weight attack is applied on RSA.

A.1.2.4 Securing Implementations of Asymmetric Schemes

Counteracting attacks on a multiplication can be achieved by switching multiplier and multiplicand. To protect implementations of modular exponentiations, an always-square-and-multiply approach can be helpful. The same is valid for implementations of the scalar point-multiplication on elliptic curves. In general,

there are more implementation options to secure elliptic curve cryptosystems. An overview of countermeasures published in the open literature is given in [424].

A.1.2.5 Conclusions and Recommendations

Hamming weight attacks are a practical threat to all (unprotected) software implementations of symmetric algorithms. Countermeasures in hard- and software have been published in the open literature (see [425] for a survey). Since the efficient implementation of countermeasures is most important, we recommend to choose algorithms which allow such efficient implementations. Summarising our considerations in [425] we can say, that ciphers without too many different algebraic structures are easier to protect. Rijndael, Khazad and Camellia are algorithms which are favourable from our point of view.

Simple side-channel attacks can be applied in practice on (unprotected) software implementations of a modular exponentiation and on potentially all kinds of (unprotected) implementations of elliptic-curve scalar point-multiplications. Such attacks are less likely to be realizable in practice for implementations of modular exponentiations than on implementations of scalar point-multiplications. But, there are much more possibilities to counteract such attacks for implementations of the scalar point-multiplication.

A.1.3 Differential Side-Channel Attacks

Differential side-channel attacks exploit the correlation between the processed data and the instantaneous side-channel leakage of the attacked cryptographic device. Due to the fact that this correlation is usually very small, statistical methods must be used to exploit it efficiently. In a differential side-channel attack the output(s) of the real physical device and the output of a hypothetical model of the device (working with a hypothetical key) are compared. Only if the hypothetical key equals the real key, the output of the hypothetical model is correlated to the output of the real device. By comparing the two outputs, the secret key can be determined. If the hypothetical model only outputs a single value (i.e. it predicts for example, the power consumption of the real device for only one moment in time), then the attack is called *first-order* differential side-channel attack. If a model can output more values for the same side-channel then such an attack is called *higher-order* differential side-channel attack. For example, if two output-values are used in an attack, then one usually refers to the attack as a second-order differential side-channel attack. If only the term differential side-channel attack is used, then it refers to a first-order differential side-channel attack.

A.1.3.1 Attacking Implementations of Symmetric Schemes

The quality of the hypothetical model of the attacker influences the strength of the attack to a large extent. Dedicated hardware implementations Feistel ciphers without an initial bit-wise addition of the key, allow the implementation of a very powerful hypothetical model. The statistical qualities of the S-boxes also influence the strength of an attack. However, none of the block ciphers which we considered in [425] showed in this case special properties.

A.1.3.2 Securing Implementations of Symmetric Schemes

As we already pointed out in the previous sections, software countermeasures are usually based on masking the data and the key during the computation. Ciphers which allow the cheap implementation of masking schemes are certainly preferable. Also for hardware countermeasures, the cheap realization is mostly important. In case of using a special logic style it is certainly an advantage if only very few different types of gates have to be designed in this logic style. The simpler a cipher can be described, the more suited it is for such an implementation.

A.1.3.3 Attacking Implementations of Asymmetric Schemes

Typical targets for an attack are again implementations of the modular exponentiation and implementations of scalar point-multiplication. Three different types of differential side-channel attacks have been introduced. Two of them, the SEMD (Single-Exponent Multiple-Data) and the MESD (Multiple-Exponent Single-Data) attack, do not require the attacker to have knowledge or a model for the attacked device. The ZEMD (Zero-Exponent Multiple-Data) attack which is essentially the same attack as proposed in [128], assumes that the attacker has a model and can predict intermediate values of the computation. Several refinements for the basic ideas behind differential side-channel attacks have been presented. An overview can be found in [424].

A.1.3.4 Securing Implementations of Asymmetric Schemes

There have been significantly less articles published dealing with countermeasures for implementations of the modular exponentiation than for countermeasures for implementations of the scalar point-multiplication on an elliptic curve (see [424] for a survey). Countermeasures for the scalar point-multiplication include randomising points, randomising curves, randomising the scalar and randomising the algorithms for the scalar point-multiplication. Since practical realizations of elliptic curve cryptosystems are software implementations (which probably make use of some accelerator unit) anyway, most of these countermeasures are cheap to implement and to combine with each other.

A.1.3.5 Conclusions and Recommendations

Summarising our considerations in [425] we can say, that ciphers without too many different algebraic operations are easier to protect. AES, Khazad and Camellia are algorithms which are favourable from our point of view. We did not consider any hash function, stream cipher or MAC algorithm in detail. However, the attack techniques and countermeasures would be exactly the same as for block ciphers.

Because of the variety of available countermeasures for elliptic curve cryptosystems, they seem to be favourable in the case of asymmetric schemes.

Hardware countermeasures suffer from the same drawbacks as already stated in A.1.2.5. Another difficulty in the implementation of asymmetric schemes is usually the inherent complexity of their implementation. Dedicated hardware implementations of asymmetric schemes are significantly larger than such implementations of symmetric schemes. This amplifies the difficulties for the application of such countermeasures.

A.1.4 Error Message Attacks

This kind of attack has been first introduced by Bleichenbacher in [86] where a chosen ciphertext attack against the RSA encryption standard PKCS#1 is described. In this standard, the decryption operation fails if the result of the RSA decryption is not in the correct format (more precisely, the first two bytes are fixed). The attack demonstrated that it is then possible to compute the RSA decryption of any ciphertext, by sending well chosen "ciphertexts" to the device and using it as an oracle to know if the corresponding plaintext is in the right format. There may be other integrity checks applied, in addition to the format checking, but the attack is still reliable if the different failures can be separated. This is the case, for instance, if different error messages are sent, or if the whole verification process takes more time to achieve than the first failure condition (and in that case, the attack will be combined with a passive side-channel technique, see Sect. A.1).

Other error message attacks are mentioned in the literature: [348] against RSA-OAEP, [159] against the NESSIE candidate EPOC-2 (this one recovers the secret key). This shows that no information about the reasons why a decryption failed should leak from the device, and how important it is to completely obscure this information.

A.1.5 Consideration of Hash-function, MACs and Stream Ciphers

There has been no research on attacks of MAC primitives and hash functions. Only in the case of the stream cipher SOBER a timing attack is known [328]. The attack-techniques however, are essentially the same as the techniques which were developed for block ciphers. Scenarios in which hash functions or MACs would be subject to attacks include constructions in which these primitives are used keyed.

A.2 Active Side-Channel Attacks

In a passive side-channel attack, the attacker only eavesdrops on some side-channel information, which is analysed afterwards to reveal some secret information. An active side-channel attack involves an attacker that takes *active* part in the attack: he somehow interacts with the device and tries to gain additional information by analysing its reactions. Some passive side-channel techniques seen in the previous section can be used to determine these reactions. This kind of attack requires the attacker to be able to deviate the device from its normal behaviour. This can be done, for instance, by modifying some internal data used by the device.

In the following we describe the most popular active side-channel attack: fault attacks. We will give examples of how successfully they have been applied, and see how they can be avoided.

A.2.1 Fault Attacks

When an attacker has physical access to a cryptographic device, he may try to force it to malfunction. A fault attack is an attack in which information about the message or the secret key is leaked from the output of erroneous computations. This kind of attack can be applied to symmetric cryptosystems as well as to asymmetric cryptosystems, and has been first introduced in [93].

There are several ways to introduce an error during the computation performed by the cryptographic device. Though the description of these practical means is beyond the scope of this introduction, we cite some non-invasive methods:

- spike attacks consist in deviating the external power supply more than can be tolerated by the device. This will surely lead to a wrong computation.
- glitch attacks are similar to spike attacks, but target the clock contact of the integrated circuit.
- optical attacks consist in focusing flash-light on the device in order to set or reset bits. It seems that very precise faults can be induced with this technique, as shown by Anderson *et al.* in [12].

We will now focus on what the attacker is able to do (i.e. the attack model) instead of considering the practical means to achieve this attack model. We will first sort these attacks according to two criteria. The first one is the attack model: how does the attacker modify the value? Which are exactly the assumptions about his capabilities? The second one concerns the value targeted: which data used by the device does the attacker modify?

A.2.1.1 Attack models

There exist a lot of different fault attacks. Most of the time, they differ by the assumptions made about the attackers capabilities: the way he can access and modify the memory, the power he has upon the fault occurrence time, ... In Blömer *et al.* [88], the authors characterise fault attacks according to different criteria:

- control on the fault location;
- control on the fault occurrence time;
- control on the number of faulty bits induced;
- fault model.

On the three first items, an attacker can have no control, loose control or precise control. We have to clarify the fault models we will consider. In [88] different models are proposed, we selected the following ones: random fault model, bit flip model, bit set or reset model.

The authors precise that the bit flip and bit (re)set models can be achieved with complete control on fault location and precise timing using optical attacks. In that case, an attacker can mount what we will call a *chosen value/modulus attack*: he can replace a value used by the device by a value of his choice. The fact that the attacker knows the original value does not matter. This kind of attack is described in Sect A.2.2.

A.2.1.2 Target values

We now differentiate the different values that can be the target of a fault attack.

In asymmetric cryptography, one party owns some secret and the other party only knows public values. Let Bob be the one who possesses the secret (the decrypter, the signer or the prover, depending on the type of primitive considered), and Alice the one that knows only the public values. First we should separate into two parts the set of data that can be made public. The *parameters* are the set of public data which are initially chosen and which define the general setting (e.g., the characteristics of the elliptic curve in elliptic curve cryptography). The *public key* is then chosen among all the public keys that the parameters permit. This public key can be modified without changing the parameters. We call the *private key* the whole set of data needed by Bob to perform his computation part that changes when we change the public key. So, the public data that is not modified when the public key changes are part of the parameters. We call the *secret key* the subset of the private key that has to be kept secret. Note that the private key depends on implementation choices. For instance, if a public modulus $n = pq$ is needed, one can choose to store n, or to store the values p and q, and to compute n. The secret key may also depend on some choices.

In symmetric cryptography, the private key is reduced to the secret key and if public data is needed, they are parameters (e.g., constant words, S-boxes).

So, in order to perform his part of the computation, Bob needs the private key (which may contain a part of the public key) and maybe some parameters. Thus a modification of any part of this set of data can possibly lead to an attack giving some information. This is important to notice because the data which does not need to be kept secret might be stored in unprotected memory location, so that it is easy to modify it. If a modification of some public data permits an attack giving information about the secret data, this data should be protected as well as the secret ones, and some additional countermeasures might be useful.

Another kind of data that can be the target of a fault attack is intermediate data. An attacker could introduce faults in the registers of the device while they are holding some intermediate values.

A.2.1.3 Published attacks

Introducing faults in the secret key of asymmetric schemes. This kind of attack has been applied by Bao *et al.* [26] to the RSA decryption (or signature) scheme, to the El Gamal, Schnorr and DSA signature schemes. It has been extended to various RSA-type signature schemes in [27], to the encryption scheme RSA-KEM in Klíma *et al.* [292], and to ACE-KEM, ECIES-KEM, PSEC-KEM in [164]. It works efficiently in the following model: bit flip fault model, complete control on the number of faulty bits induced, complete control on the location. For the timing, the fault must occur before the critical computation, so we need only loose control on it.

The idea is to flip one bit of the secret key and to use the erroneous computation of the device to get the value of this bit. This is particularly easy for discrete logarithm based schemes because we can use the simple relation between the secret and the public keys to successively guess the bits of the secret key.

Introducing faults in registers. Here, faults are introduced in an intermediate value stored in the registers of the device. This kind of attack has been applied to the Fiat-Shamir and the Schnorr identification schemes in Boneh *et al.* [93], where the random value r chosen by the prover is the target of the fault introduction. This attack works against GPS, as shown in [174]. The prover has to store this value r, as he is going to use it again when responding to the verifier's challenge. The idea is to swap exactly one bit of r while the prover is waiting for the challenge. Several such erroneous computations are used to recover the secret key.

Random faults. This is the most powerful attack model, as the fault model and the number of faulty bits induced are random, and controls on the location and the timing are loose. The well known Bellcore attack [93] against RSA using the Chinese Remainder Theorem belongs to this category. This attack has been improved by Joye *et al.* [278]. The NESSIE candidate ESIGN-D (see Sect. 7.4.2) is also vulnerable to an attack of this type [173]. Here, a random error occurs during some (more or less long) step of the computation, and the erroneous output completely reveals the secret key.

Fault attacks against elliptic curve cryptosystems. The use of elliptic curves can lead to specific fault attacks. A fault attack of this type is described in Biehl *et al.* [50]. Here, the idea is to disturb the point multiplication step such that the resulting point is on a cryptographically weak curve, where we can solve the discrete logarithm, and thus find out the secret key. First, the authors present an attack in the bit flip fault model, with complete control on the number of faulty bits induced, loose control on the location and complete control on the timing. Exactly one bit of the input point is flipped, exactly at the beginning of the multiplication process, but the attacker does not know which bit has been modified. Then the constraint on the timing is relaxed, and faults are introduced at random moments during the multiplication process.

Fault attacks against symmetric cryptosystems. Fault attacks as they have been introduced in [93] use algebraic properties of the asymmetric cryptosystems. In Biham *et al.* [71], the authors first applied fault attacks to symmetric cryptosystems, introducing *differential fault analysis*, where statistical method are to be used. Various fault models are considered, and several cryptosystems are attacked, among them the full DES. Fault attacks against the AES are considered by Blömer *et al.* in [88].

A.2.2 Chosen Modulus Attacks

Chosen modulus attacks can be viewed as a particular kind of fault attack. In a chosen modulus attack, the attacker replaces a value used by the device to perform its cryptographic computation. This can be done, for instance, by applying several bit sets/resets at a precise memory location of the device in order to replace a (possibly unknown) value by another value, this one known and chosen. The target value is more likely to be a public one (either a part of the public key,

or some parameter of the scheme), as it may not be as well protected as a secret value.

The schemes vulnerable to this kind of attacks are typically the ones where a modular exponentiation with secret exponent is performed. Usually, the (public) modulus will be replaced by a value well chosen by the attacker, enabling him to recover the secret exponent. This is done against RSA-KEM in [292]. The authors explain how to recover the secret exponent using the decrypting device with a Trojan modulus. A chosen modulus attack against GPS is described in [173].

A.2.3 Preventing fault attacks

As we have seen, fault attacks are very powerful attacks that may permit the cryptanalysis of theoretically secure schemes. Several software countermeasures have been proposed, among them:

- Double computation: for encryption schemes, this could be a solution. However, it doubles the computational time, and does not protect against permanent faults.
- Checking the output: this can be done quite efficiently with signature and identification schemes. However, it assumes that the device contains the whole public key, and this is not always the case.
- Randomisation: here random bits are introduced in the computation. They are either XOR-ed to sensitive data to blind them, or appended to the message, as in the signature scheme RSA-PSS.

In [280], Joye *et al.* show that some countermeasures can sometimes help the attacker. However, we feel confident that hardware countermeasures are also used in combination with the software countermeasures in order to avoid the large range of existing fault attacks.

References

1. M. Abdalla, M. Bellare, and P. Rogaway, "DHAES: An encryption scheme based on the Diffie-Hellman problem." Technical report, Submitted to IEEE P1363, 1998. Available from `http://grouper.ieee.org/groups/1363/P1363a/Encryption.html`. [p. 301]
2. M. Abdalla, M. Bellare, and P. Rogaway, "The Oracle Diffie-Hellman assumptions and an analysis of DHIES." 2001. Available at `http://www-cse.ucsd.edu/users/mihir/papers/dhies.html`. Earlier version in [1]. [p. 191]
3. M. Abdalla and L. Reyzin, "A new forward-secure digital signature scheme." in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 116–129, Springer-Verlag, 2000. Full version available at `http://eprint.iacr.org/2000/002/`. [p. 214]
4. M. Abe and T. Okamoto, "A signature scheme with message recovery as secure as discrete logarithms." in *Proceedings of Asiacrypt'99* (K.-Y. Lam, E. Okamoto, and C. Xing, eds.), no. 1716 in Lecture Notes in Computer Science, pp. 378–389, Springer-Verlag, 1999. [p. 234, 242]
5. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)." in *Proceedings of CHES'02* (B. S. Kaliski, Çetin Kaya Koç, and C. Paar, eds.), no. 2535 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 291]
6. M. Aigner and E. Oswald, "Power analysis tutorial." Technical report, IAIK, TU-GRAZ, 2002. Available at `http://www.iaik.tugraz.at/aboutus/people/oswald/papers/dpa_tutorial.pdf`. [p. 291]
7. M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible." in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 489–502, Springer-Verlag, 2000. [p. 291]
8. M.-L. Akkar, N. T. Courtois, R. Duteuil, and L. Goubin, "A fast and secure implementation of SFLASH." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 267–278, Springer-Verlag, 2003. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 261, 292]
9. Algorithms group of the EESSI-SG, "Elliptic Curve German Digital Signature Algorithm." Described in [263], 1999. [p. 241]
10. J. H. An, Y. Dodis, and T. Rabin, "On the security of joint signature and encryption." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 83–107, Springer-Verlag, 2002. [p. 213]
11. R. J. Anderson, "Two remarks on public key cryptology (invited lecture)." in *Conference on Computer and Communications Security – CCS'97*, 1997. Summary available at `http://www.cl.cam.ac.uk/users/rja14/`. [p. 214]
12. R. J. Anderson and S. Skorobogatov, "Optical fault induction attacks." in *Proceedings of CHES'02* (B. S. Kaliski, Çetin Kaya Koç, and C. Paar, eds.), no. 2535 in Lecture Notes in Computer Science, Springer-Verlag, 2002. Also available from `http://www.cl.cam.ac.uk/~sps32/`. [p. 296]

13. ANSI X9.19, "Financial institution retail message authentication." 1996. [p. 153]
14. ANSI X9.30.2, "Public key cryptography for the financial services industry — Part 2: The Secure Hash Algorithm (SHA-1)." 1997. [p. 132]
15. ANSI X9.31, "Digital signatures using reversible public key cryptography for the financial services industry (rDSA)." 1998. [p. 132, 254]
16. ANSI X9.62, "Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)." 1999. [p. 254]
17. ANSI X9.71, "Keyed hash message authentication code (mac)." 2000. [p. 153]
18. A. Antipa, D. R. L. Brown, A. Menezes, R. Struik, and S. A. Vanstone, "Validation of elliptic curve public keys." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 211–223, Springer-Verlag, 2003. [p.]
19. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-bit block cipher suitable for multiple platforms." Primitive submitted to NESSIE by NTT Corp., Sept. 2000. See also `http://info.isl.ntt.co.jp/camellia/` and [20]. [p. 4, 55, 57]
20. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-bit block cipher suitable for multiple platforms – design and analysis." in *Proceedings of Selected Areas in Cryptography – SAC'00* (D. R. Stinson and S. E. Tavares, eds.), no. 2012 in Lecture Notes in Computer Science, pp. 39–56, Springer-Verlag, 2001. [p. 57, 302]
21. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, "A practical and provably secure coalition-resistant group signature scheme." in *Proceedings of Crypto'00* (M. Bellare, ed.), no. 1880 in Lecture Notes in Computer Science, Springer-Verlag, 2000. Also available at `http://www.ics.uci.edu/~gts/paps/acjt00.pdf`. [p.]
22. S. Babbage, "Cryptanalysis of the LILI-128 stream cipher." in *Proceedings of the Second NESSIE Workshop*, 2001. `NES/DOC/EXT/WP3/001`. [p. 121]
23. S. Babbage, "Simple attack on BMGL faster than exhaustive key search." 23 Jan. 2002. NESSIE discussion forum. [p. 112]
24. S. Babbage and J. Lano, "Probabilistic factors in the SOBER-t stream ciphers." in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 118]
25. F. Bahr, J. Franke, and T. Kleinjung, "2.953+ c158 is factorized.". See `http://www.loria.fr/~zimmerma/records/gnfs158`. [p. 222]
26. F. Bao, R. H. Deng, Y. Han, A. B. Jeng, A. D. Narasimhalu, and T.-H. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults." in *Proceedings of Security Protocols Workshop 1997* (B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, eds.), no. 1361 in Lecture Notes in Computer Science, pp. 115–124, Springer-Verlag, 1998. [p. 257, 297]
27. F. Bao, R. H. Deng, M. Joye, and J.-J. Quisquater, "RSA-type signatures in the presence of transient faults." in *Proceedings of Cryptography and Coding – CC'97* (M. Darnell, ed.), no. 1355 in Lecture Notes in Computer Science, pp. 155–160, Springer-Verlag, 1997. [p. 297]
28. E. Barkan and E. Biham, "In how many ways can you write Rijndael?." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 160–175, Springer-Verlag, 2002. `NES/DOC/TEC/WP5/025`. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 68]
29. P. S. L. M. Barreto and V. Rijmen, "The Anubis block cipher." Primitive submitted to NESSIE, Sept. 2000. [p. 4, 82, 83, 100]
30. P. S. L. M. Barreto and V. Rijmen, "The Khazad legacy-level block cipher." Primitive submitted to NESSIE, Sept. 2000. [p. 4, 37, 39, 90]
31. P. S. L. M. Barreto and V. Rijmen, "The Whirlpool hashing function." Primitive submitted to NESSIE, Sept. 2000. [p. 5, 132, 133]
32. P. S. L. M. Barreto and V. Rijmen, "Recursive S-boxes for Anubis, Khazad, and Whirlpool." Private report, NESSIE, 2001. [p. 82]

33. P. S. L. M. Barreto, V. Rijmen, J. Nakahara, Jr, B. Preneel, J. Vandewalle, and H. Y. Kim, "Improved SQUARE attacks against reduced-round HIERO-CRYPT." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 165–173, Springer-Verlag, 2001. NES/DOC/KUL/WP3/005. [p. 27, 78, 79, 84, 85, 99, 100]
34. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Q. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay, "Report on the AES candidates." in *Proceedings of the Second Advanced Encryption Standard Conference*, pp. 53–67, NIST, 1999. Available at http://csrc.nist.gov/encryption/aes/round1/conf2/papers/baudron1.pdf. [p. 21, 63]
35. M. Bellare, A. Boldyreva, and S. Micali, "Public-key encryption in a multi-user setting: Security proofs and improvements." in *Proceedings of Eurocrypt'00* (B. Preneel, ed.), no. 1807 in Lecture Notes in Computer Science, pp. 259–274, Springer-Verlag, 2000. Full paper available at http://www-cse.ucsd.edu/users/mihir/papers/musu.html. [p. 215]
36. M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 1–15, Springer-Verlag, 1996. Also available at http://www-cse.ucsd.edu/users/mihir/papers/hmac.html. [p. 152, 163, 324]
37. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes." in *Proceedings of Crypto'98* (H. Krawczyk, ed.), no. 1462 in Lecture Notes in Computer Science, pp. 26–45, Springer-Verlag, 1998. Also available at http://www-cse.ucsd.edu/users/mihir/papers/relations.html. [p. 172]
38. M. Bellare and S. Micali, "How to sign given any trapdoor function." in *Proceedings of Crypto'88* (S. Goldwasser, ed.), no. 403 in Lecture Notes in Computer Science, pp. 200–215, Springer-Verlag, 1988. Also appeared in [39, 40]. [p. 227]
39. M. Bellare and S. Micali, "How to sign given any trapdoor function (extended abstract)." in *Proceedings of Symposium on Theory of Computing – STOC'88*, ACM Press, 1988. Full paper in [40]. [p. 303]
40. M. Bellare and S. Micali, "How to sign given any trapdoor function." *Journal of the ACM*, vol. 39, pp. 214–233, Jan. 1992. Also available at http://www-cse.ucsd.edu/users/mihir/papers/ds.html. [p. 303]
41. M. Bellare and S. Miner, "A forward-secure digital signature scheme." in *Proceedings of Crypto'99* (M. J. Wiener, ed.), no. 1666 in Lecture Notes in Computer Science, pp. 431–448, Springer-Verlag, 1999. Also available at http://www-cse.ucsd.edu/users/mihir/papers/fsig.html. [p. 214]
42. M. Bellare and C. Namprempre, "Authenticated encryption: relations among notions and analysis of the generic composition paradigm." in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 531–545, Springer-Verlag, 2000. Full paper available at http://eprint.iacr.org/2000/025/. [p. 213]
43. M. Bellare and A. Palacio, "GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks." in *Proceedings of Crypto'02* (M. Yung, ed.), no. 2442 in Lecture Notes in Computer Science, pp. 162–177, Springer-Verlag, 2002. Full paper available at http://www-cse.ucsd.edu/users/mihir/papers/gq.html. [p. 273, 286]
44. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols." in *Proceedings of Conference on Computer and Communications Security – CCS'93*, pp. 62–73, ACM Press, Nov. 1993. Full paper available at http://www-cse.ucsd.edu/users/mihir/papers/ro.html. [p. 176, 224, 227, 228, 230, 231]
45. M. Bellare and P. Rogaway, "Optimal asymmetric encryption – how to encrypt with RSA." in *Proceedings of Eurocrypt'94* (A. De Santis, ed.), no. 950 in Lecture

Notes in Computer Science, pp. 92–111, Springer-Verlag, 1994. Also available at http://www-cse.ucsd.edu/users/mihir/papers/oaep.html. [p. 208, 209, 224]
46. M. Bellare and P. Rogaway, "The exact security of digital signature – how to sign with RSA and Rabin." in *Proceedings of Eurocrypt'96* (U. Maurer, ed.), no. 1070 in Lecture Notes in Computer Science, pp. 399–416, Springer-Verlag, 1996. Revised version available at http://www-cse.ucsd.edu/users/mihir/papers/exactsigs.html. [p. 224, 229, 232, 262]
47. D. J. Bernstein, "Circuits for integer factorization." Web page. http://cr.yp.to/nfscircuit.html. [p. 222]
48. R. Bevan and E. Knudsen, "Ways to enhance differential power analysis." in *Proceedings of ICISC'02* (K. Kim, ed.), no. 2587 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 291]
49. A. Bibliowicz, P. Cohen, and E. Biham, "A system for assisting analysis of some block ciphers." Internal report, NESSIE, 2002. NES/DOC/TEC/WP2/007. [p. 26]
50. I. Biehl, B. Meyer, and V. Müller, "Differential fault attacks on elliptic curve cryptosystems." in *Proceedings of Crypto'00* (M. Bellare, ed.), vol. 1880 of *Lecture Notes in Computer Science*, pp. 131–146, Springer-Verlag, 2000. [p. 257, 298]
51. E. Biham, "New types of cryptoanalytic attacks using related keys." in *Proceedings of Eurocrypt'93* (T. Helleseth, ed.), no. 765 in Lecture Notes in Computer Science, pp. 398–409, Springer-Verlag, 1993. [p. 19, 20]
52. E. Biham, "How to forge DES-encrypted messages in 2^{28} steps." Technical report, Comp. Sci. Dept., Technion, 1996. Available from http://www.cs.technion.ac.il/Reports/. [p. 54, 92]
53. E. Biham, "Cryptanalysis of Ladder-DES." in *Proceedings of Fast Software Encryption – FSE'97* (E. Biham, ed.), no. 1267 in Lecture Notes in Computer Science, pp. 134–138, Springer-Verlag, 1997. [p. 20]
54. E. Biham, "Observations on the relations between the bit-functions of many S-boxes." in *Proceedings of the Third NESSIE Workshop*, 2002. NES/DOC/TEC/WP5/027. [p. 23, 39, 45, 52, 54, 66, 70, 79, 88, 89, 93]
55. E. Biham, "Optimization of IDEA." in *Proceedings of the Third NESSIE Workshop*, 2002. NES/DOC/TEC/WP6/026. [p. 33]
56. E. Biham and A. Biryukov, "An improved Davis' attack on DES." *Journal of Cryptology*, vol. 10, no. 3, pp. 195–205, 1997. [p. 20]
57. E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials." in *Proceedings of Eurocrypt'99* (J. Stern, ed.), no. 1592 in Lecture Notes in Computer Science, pp. 12–23, Springer-Verlag, 1999. [p. 16, 46, 95]
58. E. Biham, A. Biryukov, and A. Shamir, "Miss in the middle attacks on IDEA, Khufu, and Khafre." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 124–138, Springer-Verlag, 1999. [p. 35, 36, 90]
59. E. Biham, O. Dunkelman, V. Furman, and T. Mor, "Preliminary report on the NESSIE submissions: Anubis, Camellia, Khazad, IDEA, Misty1, NIMBUS, and Q." Public report, NESSIE, 2001. NES/DOC/TEC/WP3/011. [p. 39, 59, 95]
60. E. Biham, O. Dunkelman, and N. Keller, "Linear cryptanalysis of reduced round Serpent." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 16–27, Springer-Verlag, 2001. [p. 58]
61. E. Biham, O. Dunkelman, and N. Keller, "The rectangle attack – rectangling the Serpent." in *Proceedings of Eurocrypt'01* (B. Pfitzmann, ed.), no. 2045 in Lecture Notes in Computer Science, pp. 340–357, Springer-Verlag, 2001. [p. 58]
62. E. Biham, O. Dunkelman, and N. Keller, "Enhancing differential-linear cryptanalysis." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 254–266, Springer-Verlag, 2002. NES/DOC/TEC/WP5/017. [p. 17, 92]

63. E. Biham and V. Furman, "Differential cryptanalysis of Nimbus." Public report, NESSIE, 2001. NES/DOC/TEC/WP3/009. [p. 80, 99, 314]
64. E. Biham and V. Furman, "Differential cryptanalysis of Q." Internal report, NESSIE, 2001. NES/DOC/TEC/WP3/010. [p. 87, 100]
65. E. Biham, V. Furman, M. Misztal, and V. Rijmen, "Differential cryptanalysis of Q." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 174–186, Springer-Verlag, 2001. NES/DOC/TEC/WP3/012. [p. 86]
66. E. Biham and N. Keller, "Cryptanalysis of reduced variants of Rijndael." in *Proceedings of the Third Advanced Encryption Standard Conference*, NIST, Apr. 2000. [p. 39, 64, 66, 90, 96]
67. E. Biham, N. Keller, and O. Dunkelman, "Differential-linear cryptanalysis of Serpent." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. NES/DOC/TEC/WP5/032. [p. 17]
68. E. Biham, N. Keller, and O. Dunkelman, "Rectangle attacks on SHACAL-1." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. NES/DOC/TEC/WP5/031. [p. 76, 98, 141]
69. E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems." in *Proceedings of Crypto'90* (A. Menezes and S. A. Vanstone, eds.), no. 537 in Lecture Notes in Computer Science, pp. 2–21, Springer-Verlag, 1990. [p. 15]
70. E. Biham and A. Shamir, *Differential cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993. [p. 127]
71. E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems." in *Proceedings of Crypto'97* (B. S. Kaliski, Jr, ed.), no. 1294 in Lecture Notes in Computer Science, pp. 513–525, Springer-Verlag, 1997. [p. 298]
72. E. Biham and A. Shamir, "Power analysis of the key scheduling of the AES candidates." in *Proceedings of the Second Advanced Encryption Standard Conference*, NIST, 1999. [p. 292]
73. A. Biryukov, "Analysis of involutional ciphers: Khazad and Anubis." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. [p. 23, 40]
74. A. Biryukov and C. De Cannière, "Block ciphers and systems of quadratic equations." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 22, 38, 41, 47, 59, 64, 67]
75. A. Biryukov, C. De Cannière, and G. Dellkrantz, "Cryptanalysis of Safer++." Internal report, NESSIE, 2003. NES/DOC/KUL/WP5/028. [p. 70, 71, 96]
76. A. Biryukov and E. Kushilevitz, "Improved cryptanalysis of RC5." in *Proceedings of Eurocrypt'98* (K. Nyberg, ed.), no. 1403 in Lecture Notes in Computer Science, pp. 85–99, Springer-Verlag, 1998. [p. 60]
77. A. Biryukov, J. Nakahara, Jr, B. Preneel, and J. Vandewalle, "New weak-key classes of IDEA." in *Proceedings of ICICS'02* (R. H. Deng, S. Qing, F. Bao, and J. Zhou, eds.), no. 2513 in Lecture Notes in Computer Science, pp. 315–326, Springer-Verlag, 2002. NES/DOC/KUL/WP5/019. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 35, 37, 90]
78. A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers." in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 1–13, Springer-Verlag, 2000. [p. 104]
79. A. Biryukov and A. Shamir, "Structural cryptanalysis of SASAS." in *Proceedings of Eurocrypt'01* (B. Pfitzmann, ed.), no. 2045 in Lecture Notes in Computer Science, pp. 394–405, Springer-Verlag, 2001. [p. 20, 21]

80. A. Biryukov and D. Wagner, "Slide attacks." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 245–259, Springer-Verlag, 1999. [p. 20, 46]
81. J. Black and P. Rogaway, "Ciphers with arbitrary finite domains." in *Proceedings of CT-RSA'02* (B. Preneel, ed.), no. 2271 in Lecture Notes in Computer Science, pp. 114–130, Springer-Verlag, 2002. Also available from `http://www.cs.ucdavis.edu/~rogaway/papers/subset.htm`. [p. 248]
82. J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from PGV." in *Proceedings of Crypto'02* (M. Yung, ed.), no. 2442 in Lecture Notes in Computer Science, pp. 320–335, Springer-Verlag, 2002. Also available at `http://www.cs.colorado.edu/~jrblack/papers.html`. [p. 125, 135]
83. S. Blake-Wilson and A. J. Menezes, "Unknown key-share attacks on the station-to-station (sts) protocol." in *Proceedings of Public Key Cryptography – PKC'99* (H. Imai and Y. Zheng, eds.), no. 1560 in Lecture Notes in Computer Science, pp. 154–170, Springer-Verlag, 1999. [p. 215]
84. M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener, "Minimal key lengths for symmetric ciphers to provide adequate commercial security." Jan. 1996. Available at `http://www.counterpane.com/keylength.html`. [p. 221]
85. D. Bleichenbacher, "Generating ElGamal signatures without knowing the secret key." in *Proceedings of Eurocrypt'96* (U. Maurer, ed.), no. 1070 in Lecture Notes in Computer Science, pp. 10–18, Springer-Verlag, 1996. [p. 216, 256]
86. D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1." in *Proceedings of Crypto'98* (H. Krawczyk, ed.), no. 1462 in Lecture Notes in Computer Science, pp. 1–12, Springer-Verlag, 1998. [p. 295]
87. J. Blömer and A. May, "Low secret exponent RSA revisited." in *Cryptography and Lattices - Proceedings of CaLC'01* (J. H. Silverman, ed.), vol. 2146 of *Lecture Notes in Computer Science*, pp. 4–19, Springer-Verlag, 2001. [p. 286]
88. J. Blömer and J.-P. Seifert, "Fault based cryptanalysis of the Advanced Encryption Standard (AES)." in *Proceedings of Financial Cryptography – FC'03*, Lecture Notes in Computer Science, Springer-Verlag, 2003. Available at `http://eprint.iacr.org/2002/075/`. [p. 296, 298]
89. L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudorandom number generator." *SIAM Journal on Computing*, vol. 15, no. 2, pp. 364–383, May 1986. [p. 109]
90. M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits." *SIAM Journal on Computing*, vol. 15, no. 2, pp. 850–864, May 1986. [p. 111]
91. M. Blunden and A. Escott, "Related key attacks on reduced round KASUMI." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), Lecture Notes in Computer Science, pp. 277–285, Springer-Verlag, 2001. [p. 91]
92. D. Boneh, "Simplified OAEP for the RSA and Rabin functions." in *Proceedings of Crypto'01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 275–291, Springer-Verlag, 2001. Also available at `http://crypto.stanford.edu/~dabo/papers/saep.ps`. [p. 197, 210]
93. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults." in *Proceedings of Eurocrypt'97* (W. Fumy, ed.), no. 1233 in Lecture Notes in Computer Science, pp. 37–51, Springer-Verlag, 1997. [p. 264, 284, 285, 286, 296, 298]
94. D. Boneh, X. Ding, G. Tsudik, and B. Wong, "Instantaneous revocation of security capabilities." in *Proceedings of USENIX Security Symposium 2001*, Aug. 2001. [p. 264]

95. D. Boneh and G. Durfee, "Cryptanalysis of RSA with private key d less than $n^{0.292}$." *IEEE Transactions on Information Theory*, vol. IT-46, no. 4, pp. 1339–1349, July 2000. [p. 286]
96. D. Boneh, G. Durfee, and Y. Frankel, "An attack of RSA given a small fraction of the private key bits." in *Proceedings of Asiacrypt'98* (K. Ohta and D. Pei, eds.), no. 1514 in Lecture Notes in Computer Science, pp. 25–34, Springer-Verlag, 1998. Also available from `http://crypto.stanford.edu/~dabo/abstracts/bits_of_d.html`. [p. 264]
97. D. Boneh, G. Durfee, and N. Howgrave-Graham, "Factoring $n = p^r q$ for large r." in *Proceedings of Crypto'99* (M. J. Wiener, ed.), no. 1666 in Lecture Notes in Computer Science, pp. 326–337, Springer-Verlag, 1999. [p. 175, 196, 220]
98. D. Boneh and R. Venkatesan, "Breaking RSA may not be equivalent to factoring." in *Proceedings of Eurocrypt'98* (K. Nyberg, ed.), no. 1403 in Lecture Notes in Computer Science, pp. 59–71, Springer-Verlag, 1998. Full paper available from `http://crypto.stanford.edu/~dabo/abstracts/no_rsa_red.html`. [p. 218, 264]
99. N. Borisov, M. Chew, R. Johnson, and D. Wagner, "Multiplicative differentials." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 17–33, Springer-Verlag, 2002. [p. 35, 81]
100. J. Borst, "The block cipher Grand Cru." Primitive submitted to NESSIE, Sept. 2000. [p. 4, 83, 84]
101. J. Borst, L. R. Knudsen, and V. Rijmen, "Two attacks on reduced IDEA (extended abstract)." in *Proceedings of Eurocrypt'97* (W. Fumy, ed.), no. 1233 in Lecture Notes in Computer Science, pp. 1–13, Springer-Verlag, 1997. [p. 34, 36]
102. J. Borst, B. Preneel, and J. Vandewalle, "Linear cryptanalysis of RC5 and RC6." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 16–30, Springer-Verlag, 1999. [p. 63]
103. J. Brandt, I. Damgård, P. Landrock, and T. P. Pedersen, "Zero-knowledge authentication scheme with secret key exchange." in *Proceedings of Crypto'88* (S. Goldwasser, ed.), no. 403 in Lecture Notes in Computer Science, pp. 583–588, Springer-Verlag, 1988. [p. 307]
104. J. Brandt, I. Damgård, P. Landrock, and T. P. Pedersen, "Zero-knowledge authentication scheme with secret key exchange." *Journal of Cryptology*, vol. 11, no. 3, pp. 147–159, 1998. Journal version of [103]. [p. 281]
105. E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung, "Design validations for discrete logarithm based signature schemes." in *Proceedings of Public Key Cryptography – PKC'00* (H. Imai and Y. Zheng, eds.), no. 1751 in Lecture Notes in Computer Science, pp. 276–292, Springer-Verlag, 2000. Also available at `http://www.di.ens.fr/~pointche/pub.php?reference=BrPoVaYu00`. [p. 214, 237, 239, 241, 243, 244, 247]
106. D. R. L. Brown, "The exact security of ECDSA." Available at `http://grouper.ieee.org/groups/1363/Research/contributions/ECDSASec.ps`, improved in [108], 2000. [p. 224]
107. D. R. L. Brown, "A security analysis of the elliptic curve digital signature algorithm." in *Proc. 5th Workshop on Elliptic Curve Cryptography (ECC'01)*, 2001. Abstract available at `http://www.cacr.math.uwaterloo.ca/conferences/2001/ecc/abstracts.html`. [p. 246, 267]
108. D. R. L. Brown, "Generic groups, collision resistance, and ECDSA." Available at `http://eprint.iacr.org/2002/026/`, 2002. [p. 125, 224, 245, 247, 256, 307]
109. D. R. L. Brown, "How much "provable security" does ECDSA have?." 12 Dec. 2002. NESSIE discussion forum. [p. 236, 263, 267]

110. B. Buchberger, "Gröbner bases : an algorithmic method in polynomial ideal theory." in *Multidimensional systems theory* (N.-K. Bose, ed.), no. 16 in Mathematics and its Applications, ch. 6, pp. 184–232, D. Reidel Pub. Co., 1985. [p. 260] Based on his PhD thesis: *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, U. Innsbruck, Austria, 1965.
111. M. Burmester, "An almost-constant round interactive zero-knowledge proof." *Information Processing Letters*, vol. 42, no. 2, pp. 81–87, 1992. [p. 286]
112. J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols." in *Proceedings of SCN'02* (S. Cimato, C. Galdi, and G. Persiano, eds.), vol. 2576 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002. Also available at `http://theory.lcs.mit.edu/~cis/pubs/lysyanskaya/cl02b.ps.gz`. [p. 252]
113. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, "Resettable Zero-Knowledge." *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 42, 1999. Also available from `ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1999/TR99-042/index.html`. [p. 278]
114. R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited." in *Proceedings of Symposium on Theory of Computing – STOC'98*, pp. 209–218, ACM Press, 1998. Also available at `http://theory.lcs.mit.edu/~oded/rom.html`. [p. 176, 225]
115. Certicom Research, "Standards for Efficient Cryptography - SEC 1: Elliptic curve cryptography." 2000. Available at `http://www.secg.org/`. [p. 190, 221]
116. Certicom Research, "Standards for Efficient Cryptography - SEC 2: Recommended elliptic curve domain parameters." 2000. Available at `http://www.secg.org/`. [p. 255]
117. F. Chabaud and A. Joux, "Differential collisions in SHA-0." in *Proceedings of Crypto'98* (H. Krawczyk, ed.), no. 1462 in Lecture Notes in Computer Science, pp. 56–71, Springer-Verlag, 1998. [p. 74, 139]
118. J. H. Cheon, M. Kim, K. Kim, J.-Y. Lee, and S. Kang, "Improved impossible differential cryptanalysis of Rijndael and Crypton." in *Proceedings of ICISC'01* (K. Kim, ed.), no. 2288 in Lecture Notes in Computer Science, pp. 39–49, Springer-Verlag, 2001. [p. 66]
119. M. Ciet, G. Martinet, and F. Sica, "ACE: Advanced cryptographic engine." Public report, NESSIE, 2001. `NES/DOC/ENS/WP3/008`. [p. 188]
120. C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures." in *Proceedings of CHES'00* (Çetin Kaya Koç and C. Paar, eds.), no. 1965 in Lecture Notes in Computer Science, pp. 252–263, Springer-Verlag, 2000. [p. 292]
121. S. Contini, R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin, "The security of the RC6 block cipher." Technical report, RSA Laboratories, Aug. 1998. Available at `http://www.rsa.com/rsalabs/aes/`. [p. 17, 61, 62, 63, 95]
122. D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two." *IEEE Transactions on Information Theory*, vol. IT-30, no. 4, pp. 587–594, 1984. [p. 175, 220]
123. D. Coppersmith, "Impact of Courtois and Piepryzk results." NIST AES Discussion Forum, Sept. 2002. Available from `http://www.nist.gov/aes/`. [p. 67]
124. D. Coppersmith, "Personal communication to the authors of [375]." Apr. 2002. [p. 64, 67]
125. D. Coppersmith, S. Halevi, and C. S. Jutla, "Cryptanalysis of stream ciphers with linear masking." in *Proceedings of Crypto'02* (M. Yung, ed.), no. 2442 in Lecture Notes in Computer Science, pp. 515–532, Springer-Verlag, 2002. Also available at `http://eprint.iacr.org/2002/020/`. [p. 113]
126. J.-S. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis." in *Proceedings of CHES'00* (Çetin Kaya Koç and C. Paar,

eds.), no. 1965 in Lecture Notes in Computer Science, pp. 231–237, Springer-Verlag, 2000. [p. 292]
127. J.-S. Coron, "On the exact security of Full Domain Hash." in *Proceedings of Crypto'00* (M. Bellare, ed.), no. 1880 in Lecture Notes in Computer Science, pp. 229–235, Springer-Verlag, 2000. [p. 224, 228]
128. J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems." in *Proceedings of CHES'99* (Çetin Kaya Koç and C. Paar, eds.), no. 1717 in Lecture Notes in Computer Science, pp. 292–302, Springer-Verlag, 2000. Also available at `http://www.gemplus.com/smart/r_d/publi_crypto/download/515698_1290115520.pdf`. [p. 294]
129. J.-S. Coron, "Optimal security proofs for PSS and other signature schemes." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 272–287, Springer-Verlag, 2002. Also available at `http://eprint.iacr.org/2001/062/`. [p. 229, 231, 232]
130. N. T. Courtois, "Higher order correlation attacks, XL algorithm, and cryptanalysis of Toyocrypt." in *Proceedings of ICISC'02* (K. Kim, ed.), no. 2587 in Lecture Notes in Computer Science, Springer-Verlag, 2002. Also available at `http://eprint.iacr.org/2002/087/`. [p. 105]
131. N. T. Courtois, "Generic attacks and the security of Quartz." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 351–364, Springer-Verlag, 2003. Also available at `http://www.minrank.org/quartzbounds.pdf`. An earlier version appeared in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 234]
132. N. T. Courtois, M. Daum, and P. Felke, "On the security of HFE, HFEv- and Quartz." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 337–350, Springer-Verlag, 2003. Also available at `http://eprint.iacr.org/2002/138/`, and also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 220, 262]
133. N. T. Courtois, L. Goubin, W. Meier, and J.-D. Tacier, "Solving underfined systems of multivariate quadratic equations." in *Proceedings of Public Key Cryptography – PKC'02* (D. Naccache and P. Paillier, eds.), no. 2274 in Lecture Notes in Computer Science, pp. 211–227, Springer-Verlag, 2002. [p. 64, 67]
134. N. T. Courtois, L. Goubin, and J. Patarin, "Quartz, an asymmetric signature scheme for short signatures on PC (first version)." Primitive submitted to NESSIE, Sept. 2000. See also `http://www.minrank.org/quartz/` or [135]. [p. 262]
135. N. T. Courtois, L. Goubin, and J. Patarin, "Quartz, 128-bit long digital signature." in *Proceedings of CT-RSA'01* (D. Naccache, ed.), no. 2020 in Lecture Notes in Computer Science, pp. 282–297, Springer-Verlag, 2001. [p. 309]
136. N. T. Courtois, L. Goubin, and J. Patarin, "Quartz, an asymmetric signature scheme for short signatures on PC (second revised version)." Primitive submitted to NESSIE, Sept. 2001. See also `http://www.minrank.org/quartz/`. [p. 5, 229, 234, 262]
137. N. T. Courtois, A. Klimov, J. Patarin, and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations." in *Proceedings of Eurocrypt'00* (B. Preneel, ed.), no. 1807 in Lecture Notes in Computer Science, pp. 392–407, Springer-Verlag, 2000. [p. 22, 64, 67, 105, 260]
138. N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 267–287, Springer-Verlag, 2002. Different version of the preprint [139]. [p. 22, 29, 38, 41, 59, 64, 67, 137]
139. N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations." Available at `http://eprint.iacr.org/2002/044/`, 2002. [p. 309]

140. R. Cramer and I. B. Damgård, "New generation of secure and practical RSA-based signatures." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 173–185, Springer-Verlag, 1996. [p. 212]
141. R. Cramer and V. Shoup, "Signature schemes based on the strong RSA assumption." in *Proceedings of Conference on Computer and Communications Security – CCS'99*, pp. 46–52, ACM Press, 1999. [p. 251, 252]
142. R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack." 2001. Available at `http://www.shoup.net/papers/cca2.ps`. [p. 178, 184, 185, 186, 188]
143. P. Crowley and S. Lucks, "Bias in the LEVIATHAN stream cipher." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 211–218, Springer-Verlag, 2001. [p. 120]
144. J. Daemen, *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis*. Doctoral dissertation, K. U. Leuven, Mar. 1995. [p. 37]
145. J. Daemen, R. Govaerts, and J. Vandewalle, "Cryptanalysis of 2.5 rounds of IDEA (extended abstract)." Technical report 93/1, Department of Electrical Engineering, ESAT–COSIC, Mar. 1993. [p. 36]
146. J. Daemen, R. Govaerts, and J. Vandewalle, "Weak keys for IDEA." in *Proceedings of Crypto'93* (D. R. Stinson, ed.), no. 773 in Lecture Notes in Computer Science, pp. 224–231, Springer-Verlag, 1994. [p. 35, 37]
147. J. Daemen, L. R. Knudsen, and V. Rijmen, "The block cipher Square." in *Proceedings of Fast Software Encryption – FSE'97* (E. Biham, ed.), no. 1267 in Lecture Notes in Computer Science, pp. 149–165, Springer-Verlag, 1997. [p. 20, 21, 63, 64, 65]
148. J. Daemen, L. R. Knudsen, and V. Rijmen, "Linear frameworks for block ciphers." *Designs, Codes, and Cryptography*, vol. 22, no. 1, pp. 65–87, 2001. [p. 20, 21]
149. J. Daemen, M. Peeters, G. van Assche, and V. Rijmen, "Noekeon." Primitive submitted to NESSIE, Sept. 2000. [p. 4, 85, 86]
150. J. Daemen and V. Rijmen, "AES proposal: Rijndael." Selected as the Advanced Encryption Standard. Available from `http://www.nist.gov/aes`. [p. 21, 63, 65, 83, 96]
151. J. Daemen and V. Rijmen, "The wide trail design strategy." in *Proceedings of Cryptography and Coding – CC'01* (B. Honary, ed.), no. 2260 in Lecture Notes in Computer Science, pp. 222–238, Springer-Verlag, 2001. [p. 63]
152. I. B. Damgård and M. Koprowski, "Generic lower bounds for root extraction and signature schemes in general groups." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 256–271, Springer-Verlag, 2002. Also available at `http://eprint.iacr.org/2002/013/`. [p. 252]
153. D. W. Davies and S. Murphy, "Pairs and triplets of DES S-Boxes." *Journal of Cryptology*, vol. 8, pp. 1–25, 1995. [p. 20]
154. E. Dawson, W. Millan, L. Penna, L. Simpson, and J. D. Golic, "LILI-128." Primitive submitted to NESSIE, Sept. 2000. [p. 5]
155. C. De Cannière, "Guess and determine attack on SOBER." Public report, NESSIE, 2001. `NES/DOC/KUL/WP5/010`. [p. 118]
156. C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle, "Distinguishing attacks on Sober-t32." in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 117]
157. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely." in *Proceedings of Symposium on Theory of Computing – STOC'94*, pp. 522–533, ACM Press, 1994. [p. 264]
158. H. Demirci, "Square-like attacks on reduced rounds of IDEA." in *Proceedings of Selected Areas in Cryptography – SAC'02* (K. Nyberg and H. Heys, eds.), no. 2595 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 36, 90]

159. A. W. Dent, "An evaluation of EPOC-2." Public report, NESSIE, 2001. NES/DOC/RHU/WP5/017. [p. 195, 295]
160. A. W. Dent, "ACE-KEM and the general KEM-DEM structure." Internal report, NESSIE, 2002. NES/DOC/RHU/WP5/023. [p. 181, 182, 188, 201]
161. A. W. Dent, "Adapting the weaknesses of the random oracle model to the generic group model." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 100–109, Springer-Verlag, 2002. NES/DOC/RHU/WP5/021. Also available at http://eprint.iacr.org/2002/086/. [p. 176, 225]
162. A. W. Dent, "A designer's guide to KEMs." Public report, NESSIE, 2002. NES/DOC/RHU/WP5/029. Also available at http://eprint.iacr.org/2002/174/. [p. 181, 193, 210]
163. A. W. Dent, "An implementation attack against the EPOC-2 public-key cryptosystem." *Electronics Letters*, vol. 38, no. 9, p. 412, 2002. [p. 197]
164. A. W. Dent and E. Dottax, "An overview of side-channel attacks on the NESSIE asymmetric encryption primitives." Internal report, NESSIE, 2002. NES/DOC/RHU/WP5/020. [p. 177, 190, 194, 197, 200, 201, 297]
165. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack." in *Proceedings of CARDIS'98* (J.-J. Quisquater and B. Schneier, eds.), no. 1820 in Lecture Notes in Computer Science, pp. 167–182, Springer-Verlag, 1998. Also available at http://www.dice.ucl.ac.be/crypto/techreports.html. [p. 290]
166. M. Dichtl and M. Schafheutle, "Linearity properties of SOBER-t32 key loading." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 225–230, Springer-Verlag, 2002. [p. 108, 118]
167. M. Dichtl and P. Serf, "About the NESSIE submission "Using the general next bit predictor like an evaluation criteria"." in *Proceedings of the Second NESSIE Workshop*, 2001. NES/DOC/SAG/WP3/019. [p. 5]
168. W. Diffie and M. E. Hellman, "New directions in cryptography." *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, 1976. [p. 167, 225]
169. distributed.net, "RC5-64 has been solved.". See http://distributed.net/pressroom/news-20020926.html. [p. 222]
170. H. Dobbertin, "RIPEMD with two-round compress function is not collision-free." *Journal of Cryptology*, vol. 10, no. 1, pp. 51–69, 1997. [p. 128]
171. H. Dobbertin, "Cryptanalysis of MD4." *Journal of Cryptology*, vol. 11, no. 4, pp. 253–271, 1998. [p. 128]
172. Y. Dodis and L. Reyzin, "On the power of claw-free permutations." in *Proceedings of SCN'02* (S. Cimato, C. Galdi, and G. Persiano, eds.), vol. 2576 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002. Also available at http://eprint.iacr.org/2002/103/. [p. 228, 230, 231]
173. E. Dottax, "Fault and chosen modulus attacks on some NESSIE asymmetric primitives." Public report, NESSIE, 2002. NES/DOC/ENS/WP5/035. [p. 177, 201, 284, 298, 299]
174. E. Dottax, "Fault attacks on NESSIE signature and identification schemes." Public report, NESSIE, 2002. NES/DOC/ENS/WP5/031. [p. 284, 285, 298]
175. E. Dottax, "Three asymmetric encryption schemes based upon the factoring assumption: EPOC-2, HIME(R) and Rabin-SAEP." Public report, NESSIE, 2002. NES/DOC/ENS/WP5/028. [p. 197]
176. O. Dunkelman, "Comparing MISTY1 and KASUMI." Public report, NESSIE, Sept. 2002. NES/DOC/TEC/WP5/029. [p. 45]
177. O. Dunkelman and N. Keller, "Boomerang and rectangle attacks on SC2000." in *Proceedings of the Second NESSIE Workshop*, 2001. NES/DOC/TEC/WP3/014. [p. 88, 100]

178. C. Dwork and M. Naor, "An efficient existentially unforgeable signature scheme and its applications." in *Proceedings of Crypto'94* (Y. Desmedt, ed.), no. 839 in Lecture Notes in Computer Science, pp. 218–238, Springer-Verlag, 1994. [p. 212]
179. P. Ebinger and E. Teske, "Factoring $n = pq^2$ with the elliptic curve method." in *Proceedings of Algorithmic Number Theory Seminar – ANTS-V* (C. Fieker and D. R. Kohel, eds.), vol. 2369 of *Lecture Notes in Computer Science*, pp. 475–490, Springer-Verlag, 2002. [p. 175, 220]
180. P. Ekdahl and T. Johansson, "SNOW." Primitive submitted to NESSIE, Sept. 2000. [p. 5, 112]
181. P. Ekdahl and T. Johansson, "Distinguishing attacks on SOBER-t16 and t32." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 210–224, Springer-Verlag, 2002. [p. 115, 117]
182. E. El Mahassni, P. Q. Nguyen, and I. Shparlinsky, "The insecurity of Nyberg-Rueppel and other DSA-like signature schemes with partially known nonces." in *Cryptography and Lattices - Proceedings of CaLC'01* (J. H. Silverman, ed.), vol. 2146 of *Lecture Notes in Computer Science*, pp. 97–109, Springer-Verlag, 2001. [p. 256]
183. E. El Mahassni and I. Shparlinski, "On some uniformity of distribution properties of ESIGN." in *Proc. Intern. Workshop on Coding and Cryptography*, pp. 189–196, INRIA, 2001. Also available at `http://www.cs.mq.edu.au/~igor/ESIGN.ps`. [p. 258]
184. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms." *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 469–472, July 1985. [p. 225, 241, 312]
185. T. ElGamal, "A public-key cryptosystem and signature scheme based on discrete logarithms." in *Proceedings of Crypto'84* (G. R. Blakley and D. Chaum, eds.), no. 196 in Lecture Notes in Computer Science, pp. 10–18, Springer-Verlag, 1985. Full version in [184]. [p. 179]
186. A. Enge and P. Gaudry, "A general framework for subexponential discrete logarithm algorithms." *Acta Arithmetica*, vol. 102, no. 1, pp. 83–103, 2002. Also available at `http://www.lix.polytechnique.fr/Labo/Andreas.Enge/vorabdrucke/subexp.ps.gz`. [p. 175, 220]
187. S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital schemes." in *Proceedings of Crypto'89* (G. Brassard, ed.), no. 435 in Lecture Notes in Computer Science, pp. 263–275, Springer-Verlag, 1989. [p. 312]
188. S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital signatures." *Journal of Cryptology*, vol. 9, no. 1, pp. 35–67, 1996. Also available at `http://www.wisdom.weizmann.ac.il/~oded/PS/egm.ps`. Preliminary version in [187]. [p. 249]
189. J.-C. Faugère, "A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)." in *Proceedings of International Symposium on Symbolic and Algebraic Computation – ISSAC'02* (T. Mora, ed.), pp. 75–83, ACM Press, 2002. Also available at `http://www-calfor.lip6.fr/~jcf/Papers/@papers/f5/pdf`. [p. 260]
190. J.-C. Faugère, "Report on a successful attack of HFE challenge 1 with Gröbner bases algorithm F5/2." Announcement that appeared in `sci.crypt`, 19 Apr. 2002. [p. 220, 262]
191. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 213–230, Springer-Verlag, 2000. [p. 64, 65, 66, 96]
192. N. Ferguson, R. Schroeppel, and D. Whiting, "A simple algebraic representation of Rijndael." in *Proceedings of Selected Areas in Cryptography – SAC'01* (S. Vau-

denay and A. M. Youssef, eds.), no. 2259 in Lecture Notes in Computer Science, pp. 103–111, Springer-Verlag, 2001. [p. 64, 67]
193. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems." in *Proceedings of Crypto'86* (A. M. Odlyzko, ed.), no. 263 in Lecture Notes in Computer Science, pp. 186–194, Springer-Verlag, 1987. [p. 224, 281]
194. M. Fischlin, "A note on security proofs in the generic model." in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 458–469, Springer-Verlag, 2000. [p. 176, 225]
195. M. Fischlin, "The Cramer-Shoup Strong-RSA signature scheme revisited." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 116–129, Springer-Verlag, 2003. Also available at `http://eprint.iacr.org/2002/017/`. [p. 253]
196. S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4." in *Proceedings of Selected Areas in Cryptography – SAC'01* (S. Vaudenay and A. M. Youssef, eds.), no. 2259 in Lecture Notes in Computer Science, pp. 1–24, Springer-Verlag, 2001. [p. 122]
197. S. R. Fluhrer and D. A. McGrew, "Statistical analysis of the alleged RC4 stream cipher." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 19–30, Springer-Verlag, 2000. [p. 122]
198. P.-A. Fouque, J. Stern, and S. Vaudenay, "CS-cipher." Primitive submitted to NESSIE by CS Communication & Systèmes, Sept. 2000. [p. 4, 77, 78, 99]
199. G. Frey, M. Müller, and H.-G. Rück, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems." *IEEE Transactions on Information Theory*, vol. IT-45, no. 5, pp. 1717–1719, 1999. [p. 219]
200. G. Frey and H.-G. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves." *Mathematics of Computation*, vol. 62, no. 206, pp. 865–874, Apr. 1994. [p. 175, 220]
201. E. Fuijsaki, "Chosen ciphertext security of EPOC-2." Technical report, NTT Corporation, 2001. [p. 196]
202. E. Fujisaki, T. Kobayashi, H. Morita, H. Oguro, T. Okamoto, S. Okazaki, and D. Pointcheval, "PSEC: Provably secure elliptic curve encryption scheme." Primitive submitted to NESSIE by NTT Corp., Sept. 2000. See also `http://info.isl.ntt.co.jp/psec/`. [p. 5, 205, 206, 208]
203. E. Fujisaki, T. Kobayashi, H. Morita, H. Oguro, T. Okamoto, S. Okazaki, D. Pointcheval, and S. Uchiyama, "EPOC: Efficient probabilistic public-key encryption." Primitive submitted to NESSIE by NTT Corp., Sept. 2000. See also `http://info.isl.ntt.co.jp/epoc/`. [p. 5, 202, 203, 204]
204. E. Fujisaki and T. Okamoto, "How to enhance the security of public-key encryption at minimum cost." in *Proceedings of Public Key Cryptography – PKC'99* (H. Imai and Y. Zheng, eds.), no. 1560 in Lecture Notes in Computer Science, pp. 53–68, Springer-Verlag, 1999. [p. 202, 206, 208]
205. E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes." in *Proceedings of Crypto'99* (M. J. Wiener, ed.), no. 1666 in Lecture Notes in Computer Science, pp. 535–554, Springer-Verlag, 1999. [p. 196]
206. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption." in *Proceedings of Crypto'01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 260–274, Springer-Verlag, 2001. `NES/DOC/ENS/WP3/004`. [p. 209]
207. E. Fujiskai, T. Kobayashi, H. Morita, H. Oguro, T. Okamoto, and S. Okazaki, "ESIGN: Efficient digital signature scheme (submission to NESSIE)." Primitive submitted to NESSIE by NTT Corp., Sept. 2000. See also `http://info.isl.ntt.co.jp/esign/`. [p. 5, 257, 258]

208. J. Fuller and W. Millan, "On linear redundancy in S-boxes." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. Earlier version available at `http://eprint.iacr.org/2002/111/`. [p. 22, 23, 39, 59, 64, 66, 79]
209. V. Furman, "Differential cryptanalysis of Nimbus." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 187–195, Springer-Verlag, 2001. See also [63]. [p. 80, 81]
210. S. Furuya and V. Rijmen, "Observations on Hierocrypt-3/L1 key-scheduling algorithms." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 79, 99]
211. S. D. Galbraith, J. Malone-Lee, and N. P. Smart, "Public key signatures in the multi-user setting." *Information Processing Letters*, vol. 83, no. 5, pp. 263–266, 2002. [p. 215]
212. R. P. Gallant, R. J. Lambert, and S. A. Vanstone, "Improving the parallelized Pollard lambda search on binary anomalous curves." *Mathematics of Computation*, vol. 69, pp. 1699–1705, 2000. [p. 175, 220]
213. K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results." in *Proceedings of CHES'01* (Çetin Kaya Koç, D. Naccache, and C. Paar, eds.), no. 2162 in Lecture Notes in Computer Science, pp. 251–261, Springer-Verlag, 2001. [p. 290]
214. W. Geiselmann, R. Steinwandt, and T. Beth, "Attacking the affine parts of SFLASH." in *Proceedings of Cryptography and Coding – CC'01* (B. Honary, ed.), no. 2260 in Lecture Notes in Computer Science, pp. 355–359, Springer-Verlag, 2001. Also in *Proceedings of the Second NESSIE Workshop*, 2001. Part of these results were presented in [216]. [p. 260, 267]
215. W. Geiselmann, R. Steinwandt, and T. Beth, "Revealing 441 key bits of SFLASH^{v2}." in *Proceedings of the Third NESSIE Workshop*, 2002. Part of these results were presented in [216]. [p. 260, 267]
216. W. Geiselmann, R. Steinwandt, and T. Beth, "Revealing the affine parts of SFLASH^{v1}, SFLASH^{v2}, and FLASH." in *Actas de la VII Reunión Española de Criptologia y Seguridad de la Información. Tomo I* (S. G. Jiménez and C. M. López, eds.), pp. 305–314, 2002. [p. 314]
217. R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle." in *Proceedings of Eurocrypt'99* (J. Stern, ed.), no. 1592 in Lecture Notes in Computer Science, pp. 123–139, Springer-Verlag, 1999. [p. 250]
218. H. Gilbert, H. Handschuh, A. Joux, and S. Vaudenay, "A statistical attack on RC6." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 64–74, Springer-Verlag, 2000. [p. 63]
219. H. Gilbert and M. Minier, "A collision attack on seven rounds of Rijndael." in *Proceedings of the Third Advanced Encryption Standard Conference*, pp. 230–241, NIST, Apr. 2000. [p. 21, 64, 65, 71, 79, 96]
220. H. Gilbert and M. Minier, "Cryptanalysis of SFLASH." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 288–298, Springer-Verlag, 2002. [p. 266]
221. M. Girault, G. Poupard, and J. Stern, "Some modes of use of the GPS identification scheme." in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 242]
222. O. Goldreich, "Foundations of cryptography." Available at `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`. Three volumes: basic tools (published [224]), basic applications (in preparation), and beyond the basics (in planning). [p. 3, 11, 112, 213, 249, 273, 280]
223. O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. No. 17 in Algorithms and Combinatorics, Springer-Verlag, 1999. [p. 112]
224. O. Goldreich, *Foundations of Cryptography — Basic Tools*, vol. 1. Cambridge University Press, Aug. 2001. [p. 10, 314]

225. O. Goldreich and L. A. Levin, "A hard core predicate for any one way function." in *Proceedings of Symposium on Theory of Computing – STOC'89*, pp. 25–32, ACM Press, 1989. [p. 111, 112]
226. S. Goldwasser and M. Bellare, "Lecture notes on cryptography." Available at `http://www-cse.ucsd.edu/users/mihir/papers/gb.html`. [p. 3, 8, 11, 13]
227. S. Goldwasser and S. Micali, "Probabilistic encryption." *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984. [p. 170]
228. S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems." in *Proceedings of Symposium on Theory of Computing – STOC'85*, pp. 291–304, ACM Press, 1985. [p. 273, 275]
229. S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen message attacks." *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988. [p. 212, 213, 214, 248]
230. L. Granboulan, "How to repair ESIGN." in *Proceedings of SCN'02* (S. Cimato, C. Galdi, and G. Persiano, eds.), vol. 2576 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002. `NES/DOC/ENS/WP5/019`. Also available at `http://eprint.iacr.org/2002/074/` and also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 215, 218, 229, 258]
231. L. Granboulan, "RSA hybrid encryption schemes." Public report, NESSIE, 2002. `NES/DOC/ENS/WP5/012`. [p. 185, 201]
232. L. Granboulan, "Short signatures in the random oracle model." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 364–378, Springer-Verlag, 2002. `NES/DOC/ENS/WP5/021`. [p. 224, 232, 233]
233. L. C. Guillou and J.-J. Quisquater, "A "paradoxical" identity-based signature scheme resulting from zero-knowledge." in *Proceedings of Crypto'88* (S. Goldwasser, ed.), no. 403 in Lecture Notes in Computer Science, pp. 216–231, Springer-Verlag, 1988. [p. 286]
234. L. C. Guillou and J.-J. Quisquater, "A practical Zero-Knowledge protocol fitted to security microprocessor minimizing both transmission and memory." in *Proceedings of Eurocrypt'88* (C. Günther, ed.), vol. 330 of *Lecture Notes in Computer Science*, pp. 123–128, Springer-Verlag, 1988. [p. 285, 286]
235. S. Halevi, D. Coppersmith, and C. S. Jutla, "Scream: A software-efficient stream cipher." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 195–209, Springer-Verlag, 2002. [p. 110]
236. H. Handschuh, L. R. Knudsen, and M. J. B. Robshaw, "Analysis of SHA-1 in encryption mode." in *Proceedings of CT-RSA'01* (D. Naccache, ed.), no. 2020 in Lecture Notes in Computer Science, pp. 70–83, Springer-Verlag, 2001. [p. 75]
237. H. Handschuh and D. Naccache, "Shacal." Primitive submitted to NESSIE by Gemplus, Sept. 2000. [p. 5, 73, 75, 140, 141]
238. R. Harasawa, J. Shikata, J. Suzuki, and H. Imai, "Comparing the MOV and FR reductions in elliptic curve cryptography." in *Proceedings of Eurocrypt'99* (J. Stern, ed.), no. 1592 in Lecture Notes in Computer Science, pp. 190–205, Springer-Verlag, 1999. [p. 220]
239. C. Harpes, G. G. Kramer, and J. L. Massey, "A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma." in *Proceedings of Eurocrypt'95* (L. C. Guillou and J.-J. Quisquater, eds.), no. 921 in Lecture Notes in Computer Science, pp. 24–38, Springer-Verlag, 1995. [p. 35]
240. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo random number generators from any one-way function." *SIAM Journal on Computing*, vol. 28, pp. 1364–1396, 1999. [p. 112]
241. J. Håstad and M. Näslund, "BMGL: Synchronous key-stream generator with provable security." Primitive submitted to NESSIE, Sept. 2000. [p. 5, 110]
242. J. Håstad and M. Näslund, "A generalized interface for the NESSIE submission BMGL." Internal document, NESSIE, 2001. [p. 112]

243. Y. Hatano, H. Sekine, and T. Kaneko, "Higher order differential attack of Camellia (II)." in *Proceedings of Selected Areas in Cryptography – SAC'02* (K. Nyberg and H. Heys, eds.), no. 2595 in Lecture Notes in Computer Science, pp. 39–56, Springer-Verlag, 2002. [p. 95]
244. P. Hawkes, "Differential-linear weak key classes of IDEA." in *Proceedings of Eurocrypt'98* (K. Nyberg, ed.), no. 1403 in Lecture Notes in Computer Science, pp. 112–126, Springer-Verlag, 1998. [p. 35, 36, 37, 90]
245. P. Hawkes and G. G. Rose, "SOBER." Primitive submitted to NESSIE by Qualcomm International, Sept. 2000. [p. 5, 114, 116, 118]
246. P. Hawkes and G. G. Rose, "Guess-and-determine attacks on SNOW." in *Proceedings of Selected Areas in Cryptography – SAC'02* (K. Nyberg and H. Heys, eds.), no. 2595 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 113]
247. P. Hawkes and G. G. Rose, "On the applicability of distinguishing attacks against stream ciphers." in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 118]
248. Y. He and S. Qing, "Square attack on reduced Camellia cipher." in *Proceedings of ICICS'01* (S. Qing, T. Okamoto, and J. Zhou, eds.), no. 2229 in Lecture Notes in Computer Science, pp. 238–245, Springer-Verlag, 2001. [p. 59, 95]
249. J. C. Hernández, J. M. Sierra, J. C. Mex-Perera, D. Borrajo, A. Ribagorda, and P. Isasi, "Using the general next bit predictor like an evaluation criteria." Methodology submitted to NESSIE, Sept. 2000. [p. 5]
250. A. Hevia and M. A. Kiwi, "Strength of two data encryption standard implementations under timing attacks." *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, pp. 416–437, 1999. [p. 290]
251. P. Horster, M. Michels, and H. Petersen, "Meta-ElGamal signature schemes." in *Proceedings of Conference on Computer and Communications Security – CCS'94*, ACM Press, 1994. Full paper available at `http://www.geocities.com/CapeCanaveral/Lab/8967/TR-94-5.ps.gz`. [p. 239]
252. IEEE 1363-2000, "Standard Specifications for Public-Key Cryptography." Aug. 2000. Available from `http://standards.ieee.org/catalog/olis/busarch.html`. [p. 254]
253. IEEE P1363 working group, "Standard Specifications for Public-Key Cryptography.". Main web page at `http://grouper.ieee.org/groups/1363/`. [p. 254]
254. International Organization for Standardization, "ISO/IEC 9796-2: Information Technology - Security Techniques - Digital signature schemes giving message recovery - Part 2: Integer factorization based mechanisms." 2002. [p. 254]
255. International Organization for Standardization, "ISO/IEC 9796-3: Information Technology - Security Techniques - Digital signature schemes giving message recovery - Part 3: Discrete logarithm based mechanisms." 2000. [p. 254]
256. International Organization for Standardization, "ISO/IEC 9797-1: Information Technology - Security Techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher." 1999. [p. 151, 153]
257. International Organization for Standardization, "ISO/IEC 9797-2: Information Technology - Security Techniques - Message Authentication Codes (MACs) - Part 2: Mechanisms using a dedicated hash-function." 2002. [p. 152, 153]
258. International Organization for Standardization, "ISO/IEC 10118-2: Information Technology - Security Techniques - Hash-functions - Part 2: Hash-functions using an n-bit block cipher." 2000. [p. 129, 132]
259. International Organization for Standardization, "ISO/IEC 10118-3: Information Technology - Security Techniques - Hash-functions - Part 3: Dedicated hash-functions." 1998. [p. 132]
260. International Organization for Standardization, "ISO/IEC 10118-4: Information Technology - Security Techniques - Hash-functions - Part 4: Hash-functions using modular arithmetic." 1998. [p. 130, 132]

261. International Organization for Standardization, "ISO/IEC 14888-2: Information Technology - Security Techniques - Digital Signatures with Appendix - Part 2: Identity-based Mechanisms." 1998. [p. 254]
262. International Organization for Standardization, "ISO/IEC 14888-3: Information Technology - Security Techniques - Digital Signatures with Appendix - Part 3: Certificate-based Mechanisms." 1998. [p. 254]
263. International Organization for Standardization, "ISO/IEC 15946-2: Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 2: Digital signatures." 2002. [p. 254, 301]
264. G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying." in *Proceedings of Crypto'01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 332–354, Springer-Verlag, 2001. [p. 214]
265. T. Jakobsen, "Cryptanalysis of block ciphers with probabilistic non-linear relations of low degree." in *Proceedings of Crypto'98* (H. Krawczyk, ed.), no. 1462 in Lecture Notes in Computer Science, pp. 212–223, Springer-Verlag, 1998. [p. 19]
266. T. Jakobsen and L. R. Knudsen, "The interpolation attack on block ciphers." in *Proceedings of Fast Software Encryption – FSE'97* (E. Biham, ed.), no. 1267 in Lecture Notes in Computer Science, pp. 28–40, Springer-Verlag, 1997. [p. 19]
267. T. Jakobsen and L. R. Knudsen, "Attacks on block ciphers of low algebraic degree." *Journal of Cryptology*, vol. 14, pp. 197–210, 2001. [p. 19]
268. D. B. Johnson and S. Blake-Wilson, "ECDSA." Primitive submitted to NESSIE by Certicom, Sept. 2000. [p. 5, 234, 241, 254]
269. D. B. Johnson and S. Blake-Wilson, "ECIES." Primitive submitted to NESSIE by Certicom, Sept. 2000. [p. 5]
270. F. Jönsson and T. Johansson, "A fast correlation attack on LILI-128." Technical report, Lund University, 2001. Available at `http://www.it.lth.se/thomas/papers/paper140.ps`. [p. 121]
271. J. Jonsson, "Security proof for the RSA-PSS signature scheme." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 262, 263]
272. J. Jonsson, "An OAEP variant with a tight security proof." Available at `http://eprint.iacr.org/2002/034/`, Mar. 2002. [p. 224]
273. J. Jonsson and B. S. Kaliski, Jr, "RC6 block cipher." Primitive submitted to NESSIE by RSA, Sept. 2000. [p. 5, 60, 62, 72, 95]
274. J. Jonsson and B. S. Kaliski, Jr, "RSA-OAEP." Primitive submitted to NESSIE by RSA, Sept. 2000. [p. 5, 200, 208]
275. J. Jonsson and B. S. Kaliski, Jr, "RSA-PSS." Primitive submitted to NESSIE by RSA, Sept. 2000. [p. 5, 262]
276. A. Joux and G. Martinet, "Weaknesses in Quartz signature scheme." Public report, NESSIE, 2002. `NES/DOC/ENS/WP5/026`. [p. 262]
277. A. Joux and K. Nguyen, "Separating decision Diffie-Hellman from Diffie-Hellman in cryptographic groups." Available at `http://eprint.iacr.org/2001/003/`, 2001. [p. 219]
278. M. Joye, A. K. Lenstra, and J.-J. Quisquater, "Chinese remaindering based cryptosystems in the presence of faults." *Journal of Cryptology*, vol. 12, no. 4, pp. 241–245, 1999. Also available from `http://www.geocities.com/CapeCanaveral/Launchpad/9160/publications.html`. [p. 264, 298]
279. M. Joye and P. Paillier, "Constructive methods for the generation of prime numbers." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 217]
280. M. Joye, J.-J. Quisquater, S.-M. Yen, and M. Yung, "Observability analysis: Detecting when improved cryptosystems fail." in *Proceedings of CT-RSA'02* (B. Preneel, ed.), no. 2271 in Lecture Notes in Computer Science, pp. 17–29, Springer-Verlag, 2002. [p. 299]
281. B. S. Kaliski, Jr, "On hash function firewalls in signature scheme." in *Proceedings of CT-RSA'02* (B. Preneel, ed.), no. 2271 in Lecture

Notes in Computer Science, pp. 1–16, Springer-Verlag, 2002. Earlier version available at `http://grouper.ieee.org/groups/1363/Research/contributions/HashFunctionFirewalls.pdf`. [p. 212, 263]

282. B. S. Kaliski, Jr and M. J. B. Robshaw, "Linear cryptanalysis using multiple approximations." in *Proceedings of Crypto'94* (Y. Desmedt, ed.), no. 839 in Lecture Notes in Computer Science, pp. 26–39, Springer-Verlag, 1994. [p. 17]
283. B. S. Kaliski, Jr and Y. L. Yin, "On differential and linear cryptanalsis of the RC5 encryption algorithm." in *Proceedings of Crypto'95* (D. Coppersmith, ed.), no. 963 in Lecture Notes in Computer Science, pp. 171–184, Springer-Verlag, 1995. [p. 60]
284. T. Kawabata and T. Kaneko, "A study on higher order differential attack of Camellia." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 59]
285. KCDSA Task Force Team, "The Korean Certificate-based Digital Signature Algorithm." in *Proceedings of Asiacrypt'98* (K. Ohta and D. Pei, eds.), no. 1514 in Lecture Notes in Computer Science, pp. 175–186, Springer-Verlag, 1998. Also available at `http://grouper.ieee.org/groups/1363/P1363a/PSSigs.html` as an IEEE P1363a submission. [p. 216, 234, 241, 256]
286. L. Keliher, H. Meijer, and S. E. Tavares, "High probability linear hulls in Q." in *Proceedings of the Second NESSIE Workshop*, 2001. Available from `http://stimpy.mta.ca/faculty/lkeliher/publications.html`. [p. 86, 87, 100]
287. L. Keliher, H. Meijer, and S. E. Tavares, "Improving the upper bound on the maximum average linear hull probability for Rijndael." in *Proceedings of Selected Areas in Cryptography – SAC'01* (S. Vaudenay and A. M. Youssef, eds.), no. 2259 in Lecture Notes in Computer Science, pp. 112–128, Springer-Verlag, 2001. [p. 64, 66]
288. J. Kelsey, B. Schneier, and D. Wagner, "Key-schedule cryptanalysis of 3-WAY, IDEA, G-DES, RC4, SAFER, and Triple-DES." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 237–251, Springer-Verlag, 1996. [p. 19, 54, 92]
289. J. Kelsey, B. Schneier, and D. Wagner, "Mod n cryptanalysis, with applications against RC5P and M6." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 139–155, Springer-Verlag, 1999. [p. 18, 63]
290. J. Kim, D. Moon, W. Lee, S. Hong, S. Lee, and S. Jung, "Amplified boomerang attack against reduced-round SHACAL." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 243–253, Springer-Verlag, 2002. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 76, 98, 140, 141]
291. A. Klapper and M. Goresky, "Feedback shift registers, 2-adic span, and combiners with memory." *Journal of Cryptology*, vol. 10, no. 2, pp. 111–147, 1997. [p. 25, 106]
292. V. Klíma and T. Rosa, "Further results and considerations on side channel attacks on RSA." in *Proceedings of CHES'02* (B. S. Kaliski, Çetin Kaya Koç, and C. Paar, eds.), no. 2535 in Lecture Notes in Computer Science, Springer-Verlag, 2002. Also available at `http://eprint.iacr.org/2002/071/`. [p. 201, 210, 264, 292, 297, 299]
293. L. R. Knudsen, "Cryptanalysis of LOKI'91." in *Proceedings of Auscrypt'92* (J. Seberry and Y. Zheng, eds.), no. 718 in Lecture Notes in Computer Science, pp. 196–208, Springer-Verlag, 1993. [p. 19, 20]
294. L. R. Knudsen, "Truncated and higher order differentials." in *Proceedings of Fast Software Encryption – FSE'94* (B. Preneel, ed.), no. 1008 in Lecture Notes in Computer Science, pp. 196–211, Springer-Verlag, 1994. [p. 16, 46]
295. L. R. Knudsen, "A key-schedule weakness in SAFER K-64." in *Proceedings of Crypto'95* (D. Coppersmith, ed.), no. 963 in Lecture Notes in Computer Science, pp. 271–286, Springer-Verlag, 1995. [p. 19, 30, 50, 51, 68, 69]

296. L. R. Knudsen, "DEAL - a 128-bit block cipher." Technical report 151, Dept. of Informatics, University of Bergen, Norway, 1998. [p. 16, 46, 58]
297. L. R. Knudsen, "Contemporary block ciphers." in *Lectures on Data Security. Modern Cryptology in in Theory and Practice, LNCS Tutorial 1561* (I. B. Damgård, ed.), pp. 105–126, Springer-Verlag, 1999. Also available from `http://www.ii.uib.no/~larsr/papers/survey98.ps`. [p. 7, 15]
298. L. R. Knudsen, "A detailed analysis of SAFER K." *Journal of Cryptology*, vol. 13, pp. 417–436, 2000. [p. 30, 50, 68]
299. L. R. Knudsen, "The number of rounds in block ciphers." Internal report, NESSIE, 2000. `NES/DOC/UIB/WP3/003`. [p. 28]
300. L. R. Knudsen, "Analysis of RMAC." Public report, NESSIE, 2002. `NES/DOC/UIB/WP5/024`. [p. 162, 164]
301. L. R. Knudsen, "Correlations in RC6 on 256-bit blocks." Internal report, NESSIE, Sept. 2002. `NES/DOC/UIB/WP5/022`. [p. 21, 72, 98]
302. L. R. Knudsen, "Non-random properties of reduced-round Whirlpool." Public report, NESSIE, 2002. `NES/DOC/UIB/WP5/016`. [p. 136]
303. L. R. Knudsen, "Quadratic relations in Khazad and Whirlpool." Public report, NESSIE, 2002. `NES/DOC/UIB/WP5/017`. [p. 17, 137]
304. L. R. Knudsen and T. A. Berson, "Truncated differentials of SAFER." in *Proceedings of Fast Software Encryption – FSE'96* (D. Gollmann, ed.), no. 1039 in Lecture Notes in Computer Science, pp. 15–26, Springer-Verlag, 1996. [p. 50, 69]
305. L. R. Knudsen and W. Meier, "Improved differential attacks on RC5." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 216–228, Springer-Verlag, 1996. [p. 60]
306. L. R. Knudsen and W. Meier, "Correlations in RC6 with a reduced number of rounds." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 94–108, Springer-Verlag, 2000. [p. 21, 63, 95]
307. L. R. Knudsen and H. Raddum, "Recommendation to NIST for the AES." Public report, NESSIE, May 2000. `NES/DOC/UIB/WP3/005`. [p. 64, 67]
308. L. R. Knudsen and H. Raddum, "On Noekeon." in *Proceedings of the Second NESSIE Workshop*, 2001. `NES/DOC/UIB/WP3/009`. [p. 85, 86, 100]
309. L. R. Knudsen and V. Rijmen, "Truncated differentials of IDEA." Technical report 97/1, Department of Electrical Engineering, ESAT–COSIC, 1997. [p. 36]
310. L. R. Knudsen and M. J. B. Robshaw, "Non-linear approximations in linear cryptanalysis." in *Proceedings of Eurocrypt'96* (U. Maurer, ed.), no. 1070 in Lecture Notes in Computer Science, pp. 224–236, Springer-Verlag, 1996. [p. 17]
311. L. R. Knudsen and D. Wagner, "Integral cryptanalysis (extended abstract)." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 112–127, Springer-Verlag, 2002. `NES/DOC/UIB/WP5/015`. [p. 20, 21, 46, 64, 65, 91]
312. P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 104–113, Springer-Verlag, 1996. [p. 290]
313. P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis." in *Proceedings of Crypto'99* (M. J. Wiener, ed.), no. 1666 in Lecture Notes in Computer Science, pp. 388–397, Springer-Verlag, 1999. [p. 290, 291]
314. H. Krawczyk, "Simple forward-secure signatures from any signature scheme." in *Proceedings of Conference on Computer and Communications Security – CCS'00*, ACM Press, 2000. [p. 214]
315. H. Krawczyk and T. Rabin, "Chameleon signatures." in *Proceedings of Network and Distributed System Security Symposium – NDSS'00*, pp. 143–154, The Internet Society, 2000. Also available at `http://www.research.ibm.com/security/chameleon.ps`. [p. 248]

316. T. Krovetz, *Software-optimized universal hashing and message authentication*. Doctoral dissertation, University of California Davis, 2000. [p. 148]
317. T. Krovetz, J. Black, S. Halevi, A. Hevia, H. Krawczyk, and P. Rogaway, "UMAC." Primitive submitted to NESSIE, Sept. 2000. [p. 5, 157, 160]
318. U. Kühn, "Cryptanalysis of reduced-round MISTY." in *Proceedings of Eurocrypt'01* (B. Pfitzmann, ed.), no. 2045 in Lecture Notes in Computer Science, pp. 325–339, Springer-Verlag, 2001. [p. 46, 91]
319. U. Kühn, "Improved cryptanalysis of MISTY1." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 61–75, Springer-Verlag, 2002. Also in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 46, 91]
320. K. Kurosawa, T. Iwata, and V. D. Quang, "Root finding interpolation attack." in *Proceedings of Selected Areas in Cryptography – SAC'00* (D. R. Stinson and S. E. Tavares, eds.), no. 2012 in Lecture Notes in Computer Science, pp. 303–314, Springer-Verlag, 2001. [p. 19]
321. X. Lai, *On the Design and Security of Block Ciphers*. Hartung-Gorre Verlag, Konstanz, 1992. [p. 16, 35]
322. X. Lai, *Communication and Cryptography, Two Sides of One Tapestry*. Kluwer Academic Publishers, 1994. [p. 16]
323. X. Lai, "Higher order derivatives and differential cryptanalysis." in *In Proceedings of "Symposium on Communication, Coding and Cryptography", in honor of James L. Massey on the occasion of his 60th birthday*, 1994. [p. 16, 46]
324. X. Lai and J. L. Massey, "A proposal for a new block encryption standard." in *Proceedings of Eurocrypt'90* (I. B. Damgård, ed.), no. 473 in Lecture Notes in Computer Science, pp. 389–404, Springer-Verlag, 1990. [p. 320]
325. X. Lai and J. L. Massey, "IDEA." Primitive submitted to NESSIE by R. Straub, MediaCrypt AG, Sept. 2000. Based on [324] and [326]. [p. 4, 32]
326. X. Lai, J. L. Massey, and S. Murphy, "Markov ciphers and differential cryptanalysis." in *Proceedings of Eurocrypt'91* (D. W. Davies, ed.), no. 547 in Lecture Notes in Computer Science, pp. 17–38, Springer-Verlag, 1991. [p. 15, 16, 320]
327. S. K. Langford and M. E. Hellman, "Differential-linear cryptanalysis." in *Proceedings of Crypto'94* (Y. Desmedt, ed.), no. 839 in Lecture Notes in Computer Science, pp. 17–25, Springer-Verlag, 1994. [p. 17, 92]
328. J. Lano and G. Peeters, "Cryptanalyse van NESSIE kandidaten." Master's thesis, ESAT-COSIC, KU-Leuven, 2002. [p. 295]
329. A. N. Lebedev and A. A. Volchkov, "Nush." Primitive submitted to NESSIE by LAN Crypto, Int., Sept. 2000. [p. 4, 81, 86]
330. S. Lee, S. Hong, S. Lee, J. Lim, and S. Yoon, "Truncated differential cryptanalysis of Camellia." in *Proceedings of ICISC'01* (K. Kim, ed.), no. 2288 in Lecture Notes in Computer Science, pp. 32–38, Springer-Verlag, 2001. [p. 95]
331. A. K. Lenstra and H. W. Lenstra, Jr, eds., *The development of the number field sieve*. No. 1554 in Lecture Notes in Mathematics, Springer-Verlag, 1993. [p. 175, 219]
332. A. K. Lenstra, A. Shamir, J. Tomlinson, and E. Tromer, "Analysis of Bernstein's factorization circuit." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 1–26, Springer-Verlag, 2002. [p. 222]
333. A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes." in *Proceedings of Public Key Cryptography – PKC'00* (H. Imai and Y. Zheng, eds.), no. 1751 in Lecture Notes in Computer Science, pp. 446–465, Springer-Verlag, 2000. [p. 320]
334. A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes." *Journal of Cryptology*, vol. 14, no. 4, pp. 255–293, Autumn 2001. Full version of [333]. [p. 221]
335. H. W. Lenstra, Jr, "Factoring integers with elliptic curves." *Annals of Mathematics*, vol. 126, no. 3, pp. 649–673, Nov. 1987. Second series. [p. 175, 219]

336. M. Luby, *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, Princeton University Press, 1996. [p. 112]
337. M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions." in *Proceedings of Crypto'85* (H. C. Williams, ed.), no. 218 in Lecture Notes in Computer Science, p. 447, Springer-Verlag, 1986. Full version in [338]. [p. 321]
338. M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions." *SIAM Journal on Computing*, vol. 17, no. 2, pp. 373–386, Apr. 1988. An abstract appeared in [337]. [p. 12, 321]
339. M. Luby and C. Rackoff, "A study of password security." in *Proceedings of Crypto'87* (C. Pomerance, ed.), no. 293 in Lecture Notes in Computer Science, pp. 392–397, Springer-Verlag, 1988. [p. 12]
340. S. Lucks, "Attacking triple encryption." in *Proceedings of Fast Software Encryption – FSE'98* (S. Vaudenay, ed.), no. 1372 in Lecture Notes in Computer Science, pp. 239–253, Springer-Verlag, 1998. [p. 54, 92]
341. S. Lucks, "Attacking seven rounds of Rijndael under 192-bit and 256-bit keys." in *Proceedings of the Third Advanced Encryption Standard Conference*, NIST, Apr. 2000. [p. 64, 65, 96]
342. S. Lucks, "The saturation attack - a bait for Twofish." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 1–15, Springer-Verlag, 2001. [p. 20]
343. S. Lucks, "A variant of the Cramer-Shoup cryptosystem for groups of unknown order." in *Proceedings of Asiacrypt'02* (Y. Zheng, ed.), no. 2501 in Lecture Notes in Computer Science, pp. 27–45, Springer-Verlag, 2002. [p. 189]
344. A. W. Machado, "Nimbus." Primitive submitted to NESSIE, Sept. 2000. [p. 4, 80]
345. T. Malkin, D. Micciancio, and S. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 400–417, Springer-Verlag, 2002. [p. 214]
346. J. Malone-Lee and N. P. Smart, "Modifications of ECDSA." in *Proceedings of Selected Areas in Cryptography – SAC'02* (K. Nyberg and H. Heys, eds.), no. 2595 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 242]
347. S. Mangard, "A simple power-analysis (SPA) attack on implementations of the AES key expansion." in *Proceedings of ICISC'02* (K. Kim, ed.), no. 2587 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 292]
348. J. Manger, "A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS # 1 v2.0." in *Proceedings of Crypto'01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 230–238, Springer-Verlag, 2001. [p. 210, 295]
349. I. Mantin and A. Shamir, "A practical attack on broadcast RC4." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 152–164, Springer-Verlag, 2001. [p. 122]
350. G. Martinet, "RSA-OAEP and RSA-PSS." Public report, NESSIE, 2001. `NES/DOC/ENS/WP3/007`. [p. 209]
351. G. Martinet, "The security assumptions." Public report, NESSIE, 2001. `NES/DOC/ENS/WP3/005`. [p. 172]
352. J. L. Massey, "Shift register synthesis and BCH decoding." *IEEE Transactions on Information Theory*, vol. IT-15, pp. 122–127, 1969. [p. 105]
353. J. L. Massey, G. Khachatrian, and M. K. Kuregian, "Nomination of SAFER++ as candidate algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE)." Primitive submitted to NESSIE by Cylink Corp., Sept. 2000. [p. 5, 47, 50, 68]

354. M. Matsui, "Linear cryptanalysis method for DES cipher." in *Proceedings of Eurocrypt'93* (T. Helleseth, ed.), no. 765 in Lecture Notes in Computer Science, pp. 386–397, Springer-Verlag, 1993. [p. 16]
355. M. Matsui, "Specification of MISTY1 - a 64-bit block cipher." Primitive submitted to NESSIE by E. Takeda, Mitsubishi, Sept. 2000. [p. 4, 45]
356. M. Matsui and A. Yamagishi, "A new method for known plaintext attack of FEAL cipher." in *Proceedings of Eurocrypt'92* (R. A. Rueppel, ed.), no. 658 in Lecture Notes in Computer Science, pp. 81–91, Springer-Verlag, 1992. [p. 16]
357. U. M. Maurer and Y. Yacobi, "Non-interactive public-key cryptography." in *Proceedings of Eurocrypt'91* (D. W. Davies, ed.), no. 547 in Lecture Notes in Computer Science, pp. 498–507, Springer-Verlag, 1991. [p. 322]
358. U. M. Maurer and Y. Yacobi, "A non-interactive public-key distribution system." *Designs, Codes, and Cryptography*, vol. 9, no. 3, pp. 305–316, 1996. Also available from `http://www.crypto.ethz.ch/~maurer/publications.html`, final version of [357]. [p. 248]
359. L. May, M. Henricksen, W. Millan, G. Carter, and E. Dawson, "Strengthening the key schedule of the AES." in *Proceedings of ACISP'02* (L. M. Batten and J. Seberry, eds.), no. 2384 in Lecture Notes in Computer Science, pp. 226–240, Springer-Verlag, 2002. [p. 64, 66]
360. L. McBride, "Q." Primitive submitted to NESSIE by Mack One Software, Sept. 2000. [p. 4, 86, 87]
361. D. McGrew and S. R. Fluhrer, "LEVIATHAN." Primitive submitted to NESSIE by Cisco Systems, Inc., Sept. 2000. [p. 5]
362. W. Meier, "On the security of the IDEA block cipher." in *Proceedings of Eurocrypt'93* (T. Helleseth, ed.), no. 765 in Lecture Notes in Computer Science, pp. 371–385, Springer-Verlag, 1993. [p. 34, 36, 90]
363. W. Meier and O. Staffelbach, "Fast correlation attacks on certain stream ciphers." *Journal of Cryptology*, vol. 1, pp. 159–176, 1989. [p. 105]
364. A. J. Menezes, T. Okamoto, and S. A. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field." *IEEE Transactions on Information Theory*, vol. IT-39, pp. 1639–1646, 1993. [p. 175, 220]
365. A. J. Menezes and N. P. Smart, "Security of signature schemes in a multi-user setting." Technical report CORR 2001-63, Department of C&O, University of Waterloo, 2001. Available from `http://www.cacr.math.uwaterloo.ca/~ajmeneze/research.html`. [p. 215]
366. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`. [p. 3, 54, 92]
367. R. C. Merkle and M. E. Hellman, "On the security of multiple encryption." *Communications of the ACM*, vol. 24, 1981. [p. 54, 92]
368. T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards." in *Proceedings of CHES'99* (Çetin Kaya Koç and C. Paar, eds.), no. 1717 in Lecture Notes in Computer Science, pp. 144–157, Springer-Verlag, 2000. [p. 291]
369. S. Micali and L. Reyzin, "Signing with partially adversarial hashing." Technical report LCS-TM-575, MIT, 1998. Available at `http://www.cs.bu.edu/~reyzin/adv-hashing.html`. [p. 216, 256, 263, 267]
370. M. Michels, D. Naccache, and H. Petersen, "GOST 34.10. A brief overview of Russia's DSA." *Computers and Security*, vol. 8, no. 15, pp. 725–732, 1996. Also available at `http://www.geocities.com/CapeCanaveral/Lab/8967/TR-95-14.zip`. [p. 234, 241]
371. A. Miyaji, "A message recovery signature scheme equivalent to DSA over elliptic curves." in *Proceedings of Asiacrypt'96* (K. Kim and T. Matsumoto, eds.), no. 1163 in Lecture Notes in Computer Science, pp. 1–14, Springer-Verlag, 1996. [p. 234]

372. V. Moen, "Integral cryptanalysis of block ciphers." Master's thesis, University of Bergen, May 2002. [p. 46]
373. T. Moh, "On the Courtois-Pieprzyk's attack on Rijndael." University of San Diego Web-Site, Sept. 2002. Available from `http://www.usdsi.com/aes.html`. [p. 22, 64]
374. S. Murphy, "An analysis of SAFER." *Journal of Cryptology*, vol. 11, no. 4, pp. 235–251, 1998. [p. 50, 69]
375. S. Murphy and M. J. B. Robshaw, "Essential algebraic structure within the AES." in *Proceedings of Crypto'02* (M. Yung, ed.), no. 2442 in Lecture Notes in Computer Science, pp. 1–16, Springer-Verlag, 2002. `NES/DOC/RHU/WP5/022/1`. Also in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 22, 38, 41, 59, 64, 67, 308]
376. S. Murphy and M. J. B. Robshaw, "Key-dependent S-boxes and differential cryptanalysis." *Designs, Codes, and Cryptography*, vol. 27, pp. 229–255, 2002. [p. 19, 29]
377. S. Murphy and M. J. B. Robshaw, "Comments on the security of the AES and the XSL technique." *Electronics Letters*, vol. 39, no. 1, pp. 36–38, 2003. `NES/DOC/RHU/WP5/026`. [p. 22, 41, 59, 64, 67]
378. D. Naccache and J. Stern, "Signing on a postcard." in *Proceedings of Financial Cryptography – FC'00* (Y. Frankel, ed.), no. 1962 in Lecture Notes in Computer Science, pp. 121–135, Springer-Verlag, 2000. Also available at `http://grouper.ieee.org/groups/1363/Research/contributions/Postcard.ps`. [p. 234, 242]
379. J. Nakahara, Jr, "Extension of Knudsen's attack on SAFER K-64: L.R.Knudsen, "A detailed analysis of SAFER K", J.of Cryptology, vol.13, no.4, 2000,pp.417-436." Private Communication. [p. 70, 71, 96]
380. J. Nakahara, Jr, P. S. L. M. Barreto, B. Preneel, J. Vandewalle, and H. Y. Kim, "SQUARE attacks on reduced-round PES and IDEA block ciphers." in *Proceedings of the Second NESSIE Workshop*, 2001. `NES/DOC/KUL/WP5/017`. [p. 35, 36, 88, 90]
381. J. Nakahara, Jr, B. Preneel, and J. Vandewalle, "Linear cryptanalysis of reduced-round versions of the SAFER block cipher family." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 244–261, Springer-Verlag, 2000. [p. 51, 69, 71]
382. J. Nakahara, Jr, B. Preneel, and J. Vandewalle, "Linear cryptanalysis of reduced-round SAFER++." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 51, 52, 69, 70, 71, 92, 96]
383. J. Nakahara, Jr, B. Preneel, and J. Vandewalle, "Impossible differential attacks on reduced-round SAFER ciphers." Technical report, Department of Electrical Engineering, ESAT–COSIC, Nov. 2002. [p. 70, 71, 96]
384. National Institute of Standards and Technology, "Advanced encryption algorithm (AES) development effort.". `http://csrc.nist.gov/encryption/aes/`. [p. 324]
385. "Modes of operation for symmetric key block ciphers." NIST's CSRC Web Page. Available from `http://csrc.nist.gov/encryption/modes/`. [p. 8]
386. National Institute of Standards and Technology, "FIPS-113: Computer Data Authentication." May 1985. Available at `http://csrc.nist.gov/publications/fips/`. [p. 153, 160]
387. National Institute of Standards and Technology, "FIPS-186: Digital Signature Standard (DSS)." Nov. 1994. Available at `http://csrc.nist.gov/publications/fips/`. [p. 234, 241, 254]
388. National Institute of Standards and Technology, "FIPS-186-1: Digital Signature Standard (DSS)." Dec. 1998. Available at `http://csrc.nist.gov/publications/fips/`. [p. 254]
389. National Institute of Standards and Technology, "FIPS-186-2: Digital Signature Standard (DSS)." Jan. 2000. Available at `http://csrc.nist.gov/publications/fips/`. [p. 254]

390. National Institute of Standards and Technology, "New secure hash algorithms." 2000. [p. 76]
391. National Institute of Standards and Technology, "FIPS-197: Advanced Encryption Standard." Nov. 2001. Available at `http://csrc.nist.gov/publications/fips/`, see also [384]. [p. 31, 64]
392. National Institute of Standards and Technology, "DRAFT Recommendation for Block Cipher Modes of Operation: the RMAC Authentication Mode.." NIST Special Publication 800-38B, 4 Nov. 2002. [p. 161, 162]
393. National Institute of Standards and Technology, "FIPS-180-2: Secure Hash Standard (SHS)." Aug. 2002. Available at `http://csrc.nist.gov/publications/fips/`. [p. 76, 132, 137, 141, 142, 144]
394. National Institute of Standards and Technology, "FIPS-198: The Keyed-Hash Message Authentication Code (HMAC)." Mar. 2002. Available at `http://csrc.nist.gov/publications/fips/`, based on [36]. [p. 153, 162]
395. V. I. Nechaev, "Complexity of a determinate algorithm for the discrete logarithm." *Mathematical Notes*, vol. 55, no. 2, pp. 165–172, 1994. [p. 176, 220, 224]
396. NESSIE consortium, "Call for cryptographic primitives." NESSIE, Feb. 2000. `NES/DOC/KUL/WP1/001`. [p. 1, 2, 3, 169, 221]
397. NESSIE consortium, "NESSIE Phase I: Selection of primitives." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/017`. [p. 1]
398. NESSIE consortium, "Performance evaluation of NESSIE First Phase." Deliverable report D14, NESSIE, 2001. `NES/DOC/UCL/WP4/D14`. [p. 1, 86]
399. NESSIE consortium, "Security evaluation of NESSIE First Phase." Deliverable report D13, NESSIE, 2001. `NES/DOC/RHU/WP3/D13`. [p. 1]
400. NESSIE consortium, "Toolbox version 1." Deliverable report D9, NESSIE, 2001. `NES/DOC/SAG/WP2/0D9`. [p. 24, 25, 106]
401. P. Q. Nguyen and I. Shparlinsky, "The insecurity of the digital signature algorithm with partially known nonces." *Journal of Cryptology*, vol. 15, pp. 151–176, 2002. Also available at `ftp://ftp.ens.fr/pub/dmi/users/pnguyen/PubDSA.ps.gz`. [p. 256]
402. P. Q. Nguyen and I. Shparlinsky, "The insecurity of the elliptic curve digital signature algorithm with partially known nonces." *Design, Codes and Cryptography*, 2002. Also available at `ftp://ftp.ens.fr/pub/dmi/users/pnguyen/PubECDSA.ps.gz`. [p. 256]
403. M. Nishioka, H. Satoh, and K. Sakurai, "Design and analysis of fast provably secure public-key cryptosystems based on modular squaring." in *Proceedings of ICISC'01* (K. Kim, ed.), no. 2288 in Lecture Notes in Computer Science, pp. 81–102, Springer-Verlag, 2001. [p. 197]
404. NTT Information Sharing Platform Laboratories, "EPOC-2 specifications." Technical report, NTT Corporation, 2001. [p. 194]
405. NTT Information Sharing Platform Laboratories, "PSEC-KEM specifications." Technical report, NTT Corporation, 2001. [p. 197]
406. K. Nyberg, "Linear approximations of block ciphers." in *Proceedings of Eurocrypt'94* (A. De Santis, ed.), no. 950 in Lecture Notes in Computer Science, pp. 439–444, Springer-Verlag, 1995. [p. 17]
407. K. Nyberg and L. R. Knudsen, "Provable security against a differential attack." *Journal of Cryptology*, vol. 8, no. 1, pp. 27–38, 1995. [p. 16]
408. K. Nyberg and R. A. Rueppel, "A new signature scheme based on the dsa giving message recovery." in *Proceedings of Conference on Computer and Communications Security – CCS'93*, pp. 58–61, ACM Press, Nov. 1993. [p. 234, 242]
409. K. Nyberg and R. A. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem." in *Proceedings of Eurocrypt'94* (A. De Santis, ed.), no. 950 in Lecture Notes in Computer Science, pp. 182–193, Springer-Verlag, 1994. [p. 234, 242]

410. A. M. Odlyzko, "The future of integer factorization." *Cryptobytes*, vol. 1, no. 2, Summer 1995. Available at `http://www.rsasecurity.com/rsalabs/cryptobytes/`. [p. 221]
411. A. M. Odlyzko, "Discrete logarithms: The past and the future." *Designs, Codes, and Cryptography*, vol. 19, pp. 129–145, 2000. Also available at `http://www.research.att.com/~amo/doc/discrete.logs.future.ps`. [p. 175, 220]
412. K. Ohkuma, F. Sano, H. Muratani, M. Motoyama, and S. Kawamura, "Hierocrypt." Primitive submitted to NESSIE by Toshiba Corp., Sept. 2000. See also [413]. [p. 4, 78, 79, 84, 85, 99, 100]
413. K. Ohkuma, H. Shimizu, F. Sano, and S. Kawamura, "The block cipher Hierocrypt." in *Proceedings of Selected Areas in Cryptography – SAC'00* (D. R. Stinson and S. E. Tavares, eds.), no. 2012 in Lecture Notes in Computer Science, pp. 72–88, Springer-Verlag, 2001. [p. 325]
414. K. Ohkuma, H. Shimizu, F. Sano, and S. Kawamura, "Security assessment of Hierocrypt and Rijndael against the differential and linear cryptanalysis (extended abstract)." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 64, 66, 79, 85]
415. T. Okamoto, "Comments on PSEC-KEM, ACE-KEM and ECIES-KEM." 2002. Comments of Security Report 1.0, presented in part during the rump session of the Third NESSIE Workshop. [p. 210]
416. T. Okamoto, "ESIGN-D specification." Technical report, NESSIE external documents, Aug. 2002. Available from `http://www.cryptonessie.org/tweaks.html`. [p. 257]
417. T. Okamoto, "Security of ESIGN-D signature scheme." Technical report, NESSIE external documents, Aug. 2002. Available from `http://www.cryptonessie.org/tweaks.html`. [p. 258]
418. T. Okamoto and D. Pointcheval, "The gap problems: A new class of problems for the security of cryptographic schemes." in *Proceedings of Public Key Cryptography – PKC'01* (K. Kim, ed.), no. 1992 in Lecture Notes in Computer Science, pp. 104–118, Springer-Verlag, 2001. [p. 174]
419. T. Okamoto and D. Pointcheval, "REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform." in *Proceedings of CT-RSA'01* (D. Naccache, ed.), no. 2020 in Lecture Notes in Computer Science, pp. 159–175, Springer-Verlag, 2001. [p. 201, 203]
420. T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring." in *Proceedings of Eurocrypt'98* (K. Nyberg, ed.), no. 1403 in Lecture Notes in Computer Science, pp. 308–318, Springer-Verlag, 1998. [p. 196, 202, 203]
421. K. Okeya, H. Kurumatani, and K. Sakurai, "OK-ECDSA.". Submission to CRYPTREC, available at `http://www.sdl.hitachi.co.jp/crypto/ok-ecdsa/`. [p. 255]
422. K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic curves with the Montgomery-form and their cryptographic applications." in *Proceedings of Public Key Cryptography – PKC'00* (H. Imai and Y. Zheng, eds.), no. 1751 in Lecture Notes in Computer Science, pp. 238–257, Springer-Verlag, 2000. Also available at `http://www.sdl.hitachi.co.jp/crypto/ok-ecdsa/`. [p. 255]
423. E. Oswald, "Enhancing simple power-analysis attacks on elliptic curve cryptosystems." in *Proceedings of CHES'02* (B. S. Kaliski, Çetin Kaya Koç, and C. Paar, eds.), no. 2535 in Lecture Notes in Computer Science, Springer-Verlag, 2002. [p. 292]
424. E. Oswald and B. Preneel, "A survey on passive side-channel attacks and their countermeasures for the NESSIE public-key cryptosystems." Internal report, NESSIE, 2002. `NES/DOC/KUL/WP5/027`. [p. 177, 257, 292, 293, 294]

425. E. Oswald and B. Preneel, "A theoretical evaluation of some NESSIE candidates regarding their susceptibility towards power analysis attacks." Internal report, NESSIE, 2002. NES/DOC/KUL/WP5/022. [p. 23, 293, 294]
426. M. G. Parker, "Generalised S-Box nonlinearity." Public report, NESSIE, June 2002. NES/DOC/UIB/WP5/020. [p. 39, 46]
427. J. Patarin, N. T. Courtois, and L. Goubin, "FLASH: a fast multivariate signature algorithm." in *Proceedings of CT-RSA'01* (D. Naccache, ed.), no. 2020 in Lecture Notes in Computer Science, pp. 298–307, Springer-Verlag, 2001. [p. 266]
428. J. Patarin *et al.*, "Flash and Sflash." Primitives submitted to NESSIE, Sept. 2000. [p. 5, 229]
429. J. Patarin, L. Goubin, and N. T. Courtois, "C^{*+-} and HM: variations around two schemes of T. Matsumoto and H. Imai." in *Proceedings of Asiacrypt'98* (K. Ohta and D. Pei, eds.), no. 1514 in Lecture Notes in Computer Science, pp. 35–49, Springer-Verlag, 1998. [p. 259, 260]
430. R. Peralta and E. Okamoto, "Faster factoring of integers of a special form." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, vol. 4, no. E79-A, 1996. [p. 175, 220]
431. E. Petrank and C. Rackoff, "CBC MAC for real-time data sources." *Journal of Cryptology*, vol. 13, no. 3, pp. 315–338, 2000. [p. 151, 161]
432. L. A. Pintsov and S. A. Vanstone, "Postal revenue collection in the digital age." in *Proceedings of Financial Cryptography – FC'00* (Y. Frankel, ed.), no. 1962 in Lecture Notes in Computer Science, pp. 105–120, Springer-Verlag, 2000. Also available at http://www.cacr.math.uwaterloo.ca/techreports/2000/corr2000-43.ps. [p. 234, 242]
433. G. Piret and J.-J. Quisquater, "Integral cryptanalysis on reduced-round Safer++." Public report, NESSIE, 2003. NES/DOC/UCL/WP5/002. [p. 70, 96]
434. D. Pointcheval, "The composite discrete logarithm and secure authentication." in *Proceedings of Public Key Cryptography – PKC'00* (H. Imai and Y. Zheng, eds.), no. 1751 in Lecture Notes in Computer Science, pp. 113–128, Springer-Verlag, 2000. [p. 283]
435. D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures." *Journal of Cryptology*, vol. 13, no. 3, pp. 361–396, 2000. Also available at http://www.di.ens.fr/~pointche/pub.php?reference=PoSt00. [p. 223, 242, 243]
436. J. M. Pollard, "Monte Carlo methods for index computation mod p." *Mathematics of Computation*, vol. 32, pp. 918–924, 1978. [p. 175, 220]
437. G. Poupard *et al.*, "GPS." Primitive submitted to NESSIE, Sept. 2000. [p. 5]
438. G. Poupard and J. Stern, "Security analysis of a practical "on the fly" authentication and signature generation." in *Proceedings of Eurocrypt'98* (K. Nyberg, ed.), no. 1403 in Lecture Notes in Computer Science, pp. 422–436, Springer-Verlag, 1998. [p. 283]
439. G. Poupard and J. Stern, "On the fly signatures based on factoring." in *Proceedings of Conference on Computer and Communications Security – CCS'99*, pp. 37–45, ACM Press, 1999. [p. 283]
440. B. Preneel, "Cryptographic primitives for information authentication — state of the art." in *State of the Art in Applied Cryptography* (B. Preneel and V. Rijmen, eds.), no. 1528 in Lecture Notes in Computer Science, pp. 50–105, Springer-Verlag, 1998. [p. 123, 124, 126, 147, 149]
441. B. Preneel, A. Bosselaers, and H. Dobbertin, "The cryptographic hash function ripemd-160." *Cryptobytes*, vol. 3, no. 2, pp. 9–14, 1997. [p. 157]
442. B. Preneel and P. C. van Oorschot, "MDx-MAC and building fast MACs from hash functions." in *Proceedings of Crypto'95* (D. Coppersmith, ed.), no. 963 in Lecture Notes in Computer Science, pp. 1–14, Springer-Verlag, 1995. [p. 150, 152]

443. B. Preneel and P. C. van Oorschot, "On the security of two MAC algorithms." in *Proceedings of Eurocrypt'96* (U. Maurer, ed.), no. 1070 in Lecture Notes in Computer Science, pp. 19–32, Springer-Verlag, 1996. [p. 150]
444. S. Pyka, "Status report of SOBER-t16 cryptanalysis.". Private communication. [p. 115]
445. J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards." in *Proceedings of E-smart 2001* (I. Attali and T. P. Jensen, eds.), no. 2140 in Lecture Notes in Computer Science, pp. 200–210, Springer-Verlag, 2001. [p. 290]
446. C. Rackoff and D. R. Simon, "Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack." in *Proceedings of Crypto'91* (J. Feigenbaum, ed.), no. 576 in Lecture Notes in Computer Science, pp. 433–444, Springer-Verlag, 1991. [p. 170]
447. H. Raddum, "The statistical evaluation of the NESSIE submission IDEA." Public report, NESSIE, Sept. 2001. `NES/DOC/UIB/WP3/012`. [p. 36]
448. H. Raddum, "Cryptanalysis of IDEA-X/2." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. `NES/DOC/UIB/WP5/023`. [p. 35]
449. H. Raddum and L. R. Knudsen, "A differential attack on reduced-round SC2000." in *Proceedings of the Second NESSIE Workshop*, 2001. `NES/DOC/UIB/WP3/008`. [p. 88, 100]
450. V. Rijmen, B. Preneel, and E. De Win, "On weaknesses of non-surjective round functions." *Designs, Codes, and Cryptography*, vol. 12, no. 3, pp. 253–266, 1997. [p. 20]
451. R. L. Rivest, "The RC4 enycryption algorithm." RSA Security Inc., Mar. 1992. [p. 110, 122]
452. R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM*, vol. 21, pp. 120–126, 1978. [p. 167, 170, 201, 209, 225]
453. P. Rogaway and D. Coppersmith, "A software-oriented encryption algorithm." in *Proceedings of Fast Software Encryption – FSE'94* (B. Preneel, ed.), no. 1008 in Lecture Notes in Computer Science, pp. 56–63, Springer-Verlag, 1994. [p. 110]
454. T. Rosa, "On key-collisions in (EC)DSA schemes." Available at `http://eprint.iacr.org/2002/129/`, 2002. [p. 215]
455. RSA Labs, "PKCS #1 - RSA Cryptography Standard." June 2002. Available at `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/`. [p. 183]
456. H.-G. Rück, "On the discrete logarithm in the divisor class group of curve." *Mathematics of Computation*, vol. 68, no. 226, pp. 805–806, 1999. [p. 175, 220]
457. R. A. Rueppel, *Analysis and Design of stream ciphers*. Springer-Verlag, 1986. [p. 103]
458. M.-J. O. Saarinen, "Slid pairs in SHA-1." Rump session talk at Eurocrypt'02 and personal communication, 2002. [p. 75, 139]
459. M.-J. O. Saarinen, "A time-memory trade-off attack against LILI-128." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 231–236, Springer-Verlag, 2002. [p. 121]
460. M.-J. O. Saarinen, "Cryptanalysis of block ciphers based on SHA-1 and MD5." in *Proceedings of Fast Software Encryption – FSE'03* (T. Johansson, ed.), Lecture Notes in Computer Science, Springer-Verlag, 2003. Also available at `http://www.tcs.hut.fi/~mjos/shaan.ps`. [p. 75, 76, 77, 141]
461. T. Satoh and K. Araki, "Fermat quotients and the polynomial time discrete algorithm for anomalous elliptic curves." *Commentarii Math. Univ. Sancti Pauli*, vol. 47, no. 1, pp. 81–92, 1998. Errata in [462]. [p. 175, 220]

462. T. Satoh and K. Araki, "Errata to the paper: Fermat quotients and the polynomial time discrete algorithm for anomalous elliptic curves." *Commentarii Math. Univ. Sancti Pauli*, vol. 48, pp. 211–213, 1999. Summary available at `http://www.rimath.saitama-u.ac.jp/lab.en/TkkzSatoh/A1998.html`. [p. 327]
463. W. Schindler, "A combined timing and power attack." in *Proceedings of Public Key Cryptography – PKC'02* (D. Naccache and P. Paillier, eds.), no. 2274 in Lecture Notes in Computer Science, pp. 263–279, Springer-Verlag, 2002. [p. 291]
464. B. Schneier, "Bernstein's factoring breakthrough?." Crypto-Gram, Mar. 2002. Available from `http://www.counterpane.com/crypto-gram-0203.html`. [p. 222]
465. C. P. Schnorr, "Efficient identification and signatures for smart cards." in *Proceedings of Crypto'89* (G. Brassard, ed.), no. 435 in Lecture Notes in Computer Science, pp. 239–252, Springer-Verlag, 1989. [p. 234, 242]
466. C. P. Schnorr, "Efficient signature generation by smart cards." *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991. [p. 234]
467. C. P. Schnorr and S. Vaudenay, "Black box cryptanalysis of hash networks based on multipermutations." in *Proceedings of Eurocrypt'94* (A. De Santis, ed.), no. 950 in Lecture Notes in Computer Science, pp. 47–57, Springer-Verlag, 1994. [p. 78]
468. K. Schramm, "DES sidechannel collision attacks on smartcard implementations." Master's thesis, Ruhr Universität Bochum, Aug. 2002. [p. 292]
469. T. Schweinberger and V. Shoup, "ACE: The advanced cryptographic engine." Primitive submitted to NESSIE, Sept. 2000. [p. 5]
470. I. A. Semaev, "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p." *Mathematics of Computation*, vol. 67, no. 221, pp. 353–356, 1998. [p. 175, 220]
471. P. Serf, "The degrees of completeness, of avalanche effect, and of strict avalanche criterion for Mars, RC6, Rijndael, Serpent, and Twofish with reduced number of rounds." Public report, NESSIE, 2000. `NES/DOC/SAG/WP3/003`. [p. 25]
472. A. Shamir and A. Kipnis, "Cryptanalysis of the HFE public key cryptosystem." in *Proceedings of Crypto'99* (M. J. Wiener, ed.), no. 1666 in Lecture Notes in Computer Science, pp. 19–30, Springer-Verlag, 1999. [p. 22, 220]
473. C. E. Shannon, "Communication theory of secrecy systems." *Bell System Technical Journal*, vol. 28, 1949. [p. 7, 9]
474. T. Shimoyama, M. Takenaka, and T. Koshiba, "Multiple linear cryptanalysis of a reduced round RC6." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 76–88, Springer-Verlag, 2002. [p. 60, 62, 95]
475. T. Shimoyama, H. Yanami, K. Yokoyama, M. Takenaka, K. Itoh, J. Yajima, N. Torii, and H. Tanaka, "SC2000." Primitive submitted to NESSIE by N. Torii, Fujitsu Laboratories LTD, Sept. 2000. Based on [476]. [p. 5, 87, 88, 100]
476. T. Shimoyama, H. Yanami, K. Yokoyama, M. Takenaka, K. Itoh, J. Yajima, N. Torii, and H. Tanaka, "The block cipher SC2000." in *Proceedings of Fast Software Encryption – FSE'01* (M. Matsui, ed.), no. 2355 in Lecture Notes in Computer Science, pp. 312–327, Springer-Verlag, 2001. [p. 88, 328]
477. R. Shipsey, "ECIES." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/007`. [p. 191]
478. R. Shipsey, "GPS." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/004`. [p. 283]
479. R. Shipsey, "How long...?." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/015`. [p. 177, 221]
480. R. Shipsey, "PSEC-1-2-3." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/008`. [p. 199, 206, 208]
481. R. Shipsey, "PSEC-KEM." Public report, NESSIE, 2001. `NES/DOC/RHU/WP5/016`. [p. 199]

482. R. Shipsey, "Selecting a version of PSEC." Public report, NESSIE, 2001. NES/DOC/RHU/WP3/012. [p. 196, 199, 203, 204, 206, 208]
483. T. Shirai, "Differential, linear, boomerang and rectangle cryptanalysis of reduced-round Camellia." in *Proceedings of the Third NESSIE Workshop*, 2002. [p. 58, 95]
484. T. Shirai, S. Kanamaru, and G. Abe, "Improved upper bounds of differential and linear characteristic probability for Camellia." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 128–142, Springer-Verlag, 2002. [p. 58]
485. P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring." in *Proceedings of FOCS'94*, pp. 124–134, IEEE, 1994. [p. 175, 220]
486. V. Shoup, "Lower bounds for discrete logarithms and related problems." in *Proceedings of Eurocrypt'97* (W. Fumy, ed.), no. 1233 in Lecture Notes in Computer Science, pp. 256–266, Springer-Verlag, 1997. Revised version available at http://shoup.net/papers/dlbounds1.pdf. [p. 174, 176, 220, 224, 245]
487. V. Shoup, "Using hash functions as a hedge against chosen ciphertext attack." in *Proceedings of Eurocrypt'00* (B. Preneel, ed.), no. 1807 in Lecture Notes in Computer Science, pp. 275–288, Springer-Verlag, 2000. Also available at http://www.shoup.net/papers/hedge.ps. [p. 189]
488. V. Shoup, "OAEP reconsidered." in *Proceedings of Crypto'01* (J. Kilian, ed.), no. 2139 in Lecture Notes in Computer Science, pp. 239–259, Springer-Verlag, 2001. Also available at http://www.shoup.net/papers/oaep.pdf. [p. 209]
489. V. Shoup, "A proposal for an ISO standard for public key encryption (version 2.0)." Available at http://eprint.iacr.org/2001/112/, 2001. [p. 5, 178, 183, 186, 190, 191, 192, 199, 200, 235, 237]
490. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only." *IEEE Transactions on Computers*, vol. 34, no. 1, pp. 81–85, 1985. [p. 105]
491. R. D. Silverman, "A cost based security analysis of symmetric and asymmetric key lengths." RSA Bulletin 13, RSA Laboratories, Apr. 2000. Available at http://www.rsasecurity.com/rsalabs/bulletins/. [p. 221]
492. N. P. Smart, "The discrete logarithm problem on elliptic curve of trace one." *Journal of Cryptology*, vol. 12, no. 3, pp. 193–196, 1999. [p. 175, 220]
493. N. P. Smart, "The exact security of ECIES in the generic group model." in *Proceedings of Cryptography and Coding – CC'01* (B. Honary, ed.), no. 2260 in Lecture Notes in Computer Science, pp. 73–84, Springer-Verlag, 2001. [p. 192]
494. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "Efficient FPGA implementations of block ciphers Khazad and MISTY1." in *Proceedings of the Third NESSIE Workshop*, 2002. [p.]
495. R. Steinwandt, W. Geiselmann, and T. Beth, "A theoretical DPA-based cryptanalysis of the NESSIE candidates FLASH and SFLASH." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 261]
496. J. Stern, "Almost uniform density of e-th powers on large intervals." 2002. [p. 258]
497. J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart, "Flaws in applying proof methodologies to signature schemes." in *Proceedings of Crypto'02* (M. Yung, ed.), no. 2442 in Lecture Notes in Computer Science, pp. 93–110, Springer-Verlag, 2002. Also available at http://www.di.ens.fr/~pointche/pub.php?reference=MaPoSmSt02. [p. 213, 214, 230, 258]
498. D. R. Stinson, *Cryptography: Theory and Practice*. CRC Press, 1995. [p. 3]
499. M. Sudan, "Decoding of Reed Solomon codes beyond the error-correction bound." *Journal of Complexity*, vol. 13, no. 1, pp. 180–193, 1997. Also available at http://theory.lcs.mit.edu/~madhu/papers/reeds-journ.ps. [p. 19]
500. M. Sugita, "Higher order differential attack on block cipher MISTY1, 2." Technical report ISEC98-4, IEICE, May 1998. [p. 91]

501. M. Sugita, K. Kobara, and H. Imai, "Security of reduced version of the block cipher Camellia against truncated and impossible differential cryptanalysis." in *Proceedings of Asiacrypt'01* (C. Boyd, ed.), no. 2248 in Lecture Notes in Computer Science, pp. 193–207, Springer-Verlag, 2001. [p. 58, 95]
502. H. Tanaka, K. Hisamatsu, and T. Kaneko, "Strength of MISTY1 without FL function for higher order differential attack." in *Proceedings of AAECC'99* (M. Fossorier, H. Imai, S. Lin, and A. Poli, eds.), no. 1719 in Lecture Notes in Computer Science, pp. 221–230, Springer-Verlag, 1999. [p. 46, 91]
503. H. Tanaka, C. Ishii, and T. Kaneko, "On the strength of KASUMI without FL functions against higher order differential attack." in *Proceedings of ICISC'00* (D. Won, ed.), no. 2015 in Lecture Notes in Computer Science, pp. 14–21, Springer-Verlag, 2000. [p. 91]
504. E. van den Bogaert and V. Rijmen, "Differential analysis of SHACAL." Internal report, NESSIE, May 2001. `NES/DOC/KUL/WP3/009`. [p. 75, 140]
505. P. C. van Oorschot and M. J. Wiener, "Parallel collision search with applications to hash functions and discrete logarithms." in *Proceedings of Conference on Computer and Communications Security – CCS'94*, pp. 210–218, ACM Press, 1994. [p. 175, 220]
506. P. C. van Oorschot and M. J. Wiener, "Improving implementable meet-in-the-middle attacks by orders of magnitude." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 229–236, Springer-Verlag, 1996. [p. 54]
507. B. Van Rompay and B. Den Boer, "TTMAC." Primitive submitted to NESSIE, Sept. 2000. [p. 5, 153]
508. S. A. Vanstone, "Responses to NIST's proposal." *Communications of the ACM*, vol. 35, pp. 50–52, July 1992. [p. 254]
509. S. Vaudenay, "An experiment on DES - statistical cryptanalysis." in *Proceedings of Conference on Computer and Communications Security – CCS'95*, pp. 139–147, ACM Press, 1995. [p. 21]
510. S. Vaudenay, "Hidden collisions on DSS." in *Proceedings of Crypto'96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 83–88, Springer-Verlag, 1996. [p. 216, 256]
511. S. Vaudenay, "On the security of CS-Cipher." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 260–274, Springer-Verlag, 1999. [p. 78]
512. S. Vaudenay, "Security flaws induced by CBC padding — applications to SSL, IPSEC, WTLS." in *Proceedings of Eurocrypt'02* (L. R. Knudsen, ed.), no. 2332 in Lecture Notes in Computer Science, pp. 534–546, Springer-Verlag, 2002. [p. 291]
513. S. Vaudenay, "The security of DSA and ECDSA." in *Proceedings of Public Key Cryptography – PKC'03* (Y. Desmedt, ed.), no. 2567 in Lecture Notes in Computer Science, pp. 309–323, Springer-Verlag, 2003. Also available from `http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Vau03a`. [p. 216, 256]
514. D. Wagner, "The boomerang attack." in *Proceedings of Fast Software Encryption – FSE'99* (L. R. Knudsen, ed.), no. 1636 in Lecture Notes in Computer Science, pp. 156–170, Springer-Verlag, 1999. [p. 16, 17, 19, 36]
515. C. D. Walter and S. Thompson, "Distinguishing exponent digits by observing modular subtractions." in *Proceedings of CT-RSA'01* (D. Naccache, ed.), no. 2020 in Lecture Notes in Computer Science, pp. 192–207, Springer-Verlag, 2001. [p. 291]
516. W. Wenling and F. Dengguo, "Linear cryptanalysis of NUSH block cipher." *Science in China (Series F)*, vol. 45, no. 1, pp. 59–67, 2002. [p. 81, 99, 100]
517. R. Wernsdorf, "IDEA, SAFER++ and their permutation groups." in *Proceedings of the Second NESSIE Workshop*, 2001. [p. 34]

518. R. Wernsdorf, "The round functions of Rijndael generate the alternating group." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 143–148, Springer-Verlag, 2002. [p. 64, 66]
519. J. White, "Initial report on the EPOC asymmetric encryption scheme." Public report, NESSIE, 2001. `NES/DOC/RHU/WP3/011`. [p. 195, 203, 204]
520. M. J. Wiener, "Cryptanalysis of short RSA secret exponents." in *Proceedings of Eurocrypt'89* (J.-J. Quisquater and J. Vandewalle, eds.), vol. 434 of *Lecture Notes in Computer Science*, p. 372, Springer-Verlag, 1990. [p. 286]
521. M. J. Wiener, "Performance comparison of public key cryptosystems." *Cryptobytes*, vol. 4, no. 1, Summer 1998. Available at `http://www.rsasecurity.com/rsalabs/cryptobytes/`. [p. 221]
522. M. J. Wiener and R. J. Zuccherato, "Fast attacks on elliptic curve cryptosystems." in *Proceedings of Selected Areas in Cryptography – SAC'98* (S. E. Tavares and H. Meijer, eds.), no. 1556 in Lecture Notes in Computer Science, pp. 190–200, Springer-Verlag, 1999. [p. 175, 220]
523. Y. Yeom, S. Park, and I. Kim, "On the security of CAMELLIA against the square attack." in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 89–99, Springer-Verlag, 2002. [p. 58, 95]
524. A. M. Youssef and G. Gong, "On the interpolation attacks on block ciphers." in *Proceedings of Fast Software Encryption – FSE'00* (B. Schneier, ed.), no. 1978 in Lecture Notes in Computer Science, pp. 109–120, Springer-Verlag, 2000. [p. 19]
525. A. M. Youssef and S. E. Tavares, "On some algebraic structures in the AES round function." Available at `http://eprint.iacr.org/2002/144/`, 2002. [p. 23, 66]