An Introduction to Cryptology

Bart Preneel*

Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium
bart.preneel@esat.kuleuven.ac.be

Abstract. This paper provides an overview of the state of the art in the design of cryptographic algorithms. It reviews the different type of algorithms for encryption and authentication and explains the principles of stream ciphers, block ciphers, hash functions, public-key encryption algorithms, and digital signature schemes. Subsequently the design and evaluation procedures for cryptographic algorithms are discussed.

1 Introduction

In our society, digital information and the systems and networks carrying this information are abused under many forms: financial transactions are eavesdropped or modified, sensitive information of individuals and organizations is eavesdropped or stolen, electronic services are used without paying for them, and computer systems and networks are broken into or brought down. The tools to perform this vary from simple bugs, password sniffers, and password crackers, over malicious software such as viruses and malicious applets, to complete hacker workbenches. Traditionally computer networks existed within one organization, and one tried to defend them against an opponent that came from outside the system. Now that we move to open and global networks, and that we are entering an era of electronic commerce, a more complex threat model arises: we cannot even trust the parties we are dealing with, and the system has to be designed to fight fraud within the system. For example, in an electronic transaction system, sellers can deny having sent an order if it turns out badly, and traders can deny having received an order when it turns out profitable (in order to keep the money). The risk for misuse has increased considerably, as potential attackers can operate from all over the globe. Moreover, if someone gains access to an electronic information system, the scale and impact of the abuse can be much larger than in a paper-based system.

These risks create the need for adequate security measures to protect electronic information systems. It is clear that in an electronic world physical security or personnel security by itself cannot be sufficient. An essential component of every secure information system is formed by cryptographic techniques. Other important building blocks are secure operating systems and procedural aspects

* F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium).

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

such as audit tools and management guidelines. Building secure computer systems and networks requires a conservative approach which is not always compatible with the current rapid developments in the industry; moreover, that security has to be kept in mind from the first step of the design.

In this paper we discuss the principles underlying the design of cryptographic algorithms; we distinguish between confidentiality protection (the protection against passive eavesdroppers) and authentication (the protection against active eavesdroppers, who try to modify information). Then we review the different issues that arise when selecting, designing, and evaluating a cryptographic algorithm. Finally we present some concluding remarks.

2 Encryption for Secrecy Protection

The use of cryptography for protection the secrecy of information is as old as writing itself (for an excellent historical overview, see D. Kahn [14]). The basic idea is to apply a 'complicated' transformation to the information to be protected. When the sender (usually called Alice in cryptography) wants to send a message to the recipient (Bob), she will apply to the *plaintext* P the mathematical transformation $E()$. This transformation $E()$ is called the encryption algorithm; the result of this transformation is called the *ciphertext* or $C = E(P)$. Bob will decrypt C by applying the inverse transformation $D = E^{-1}$; this way he recovers P or $P = D(C)$. For a secure algorithm E, the ciphertext C does not make sense to outsiders: Eve, who is tapping the connection, can obtain C, but she cannot obtain (partial information on) the corresponding plaintext P.

This approach only works when Bob can keep the transformation D secret. While this is acceptable for a person-to-person exchange, it is not feasible for large scale use. Bob needs a software or hardware implementation of D: either he has to program it himself, or he has to trust someone to write the program for him. Moreover, he will need a different transformation (and program) for each correspondent, which is not very practical. Bob and Alice always have to face the risk that somehow Eve will obtain D (or E), for example by bribing the author of the software or their system manager, or by breaking into their computer system.

This problem can be solved by introducing into the encryption algorithm $E()$ a secret parameter, the key K. Typically such a key is a binary string of 40 to a few thousand bits. A corresponding key K^* is used for the decryption algorithm D. One has thus $C = E_K(P)$ and $P = D_{K^*}(C)$ (see also Figure 1, which assumes that $K^* = K$). The transformation has to depend strongly (and in a very complicated way) on the keys: if one uses a wrong key $K^{*\prime} \neq K^*$, one does not obtain the plaintext P but a 'random' plaintext P'. Now it is possible to publish the encryption algorithm $E()$ and the decryption algorithm $D()$; the security of the system relies only on the secrecy of two short keys. This implies that $E()$ and $D()$ can be evaluated publicly and distributed on a commercial basis. One can think of the analogy with a mechanical lock: everyone knows how such a lock works, but in order to open a particular lock, one needs to know the

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

key or the secret combination. The assumption that the algorithm is known to the opponent is known in cryptography as "Kerckhoffs's principle"; Kerckhoffs was a 19th century Dutch cryptographer who was the first to formulate this approach.

A simple example of an encryption algorithm is the so-called 'Caesar cipher,' after the Roman emperor who used it. The plaintext is encrypted letter by letter; the ciphertext is obtained by shifting the letters over a fixed number of positions in the alphabet. The secret key indicates the number of positions. It is claimed that Caesar always used the value of three, such that "AN EXAMPLE" would be encrypted to "DQ HADPSOH". Another example is the name of the computer "HAL" from S. Kubrick's "A Space Odyssey (2001)", which was obtained by replacing the letters of "IBM" by their predecessor in the alphabet. This corresponds to a shift over 25 positions. It is clear that such a system is completely insecure.

A problem which has not yet been addressed is how Alice and Bob exchange the secret key. The easy answer is that cryptography does not solve this problem; cryptography only makes problems easier. In this case the secrecy of a (large) plaintext has been reduced to that of two *short* keys, which can be exchanged on beforehand. The problem of exchanging keys is studied in more detail in an area of cryptography that is called 'key management'. We will not discuss it in further detail here.

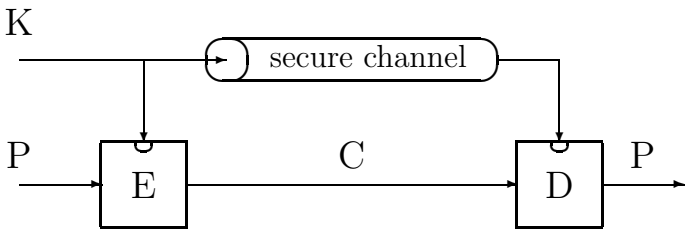

Fig. 1. Model for conventional or symmetric encryption

The branch of science which studies the encryption of information is called *cryptography*. A related branch tries to 'break' encryption algorithms, by recovering the plaintext without knowing the key or by deriving the key from the ciphertext and parts of the plaintext; it is called *cryptanalysis*. The term *cryptology* covers both aspects. For more extensive introductions to cryptography, the reader is referred to [2, 15, 19, 20, 25, 26].

Thus far we have assumed that the key for decryption K^* is equal to the encryption key K, or that it is easy to derive K^* from K. This type of algorithms are called *conventional* or *symmetric* ciphers. In *public-key* or *asymmetric* ciphers, K^* and K are always different; moreover, it should be difficult to compute K^* from K. This has the advantage that one can make K public, which has important implications to the key management problem. The remainder of this section discusses conventional algorithms and public-key algorithms.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

2.1 Conventional Encryption

This section introduces the two most common conventional encryption algorithms: additive stream ciphers and block ciphers.

Additive Stream Ciphers. Additive stream ciphers are ciphers for which the encryption consists of a modulo 2 addition (exclusive or, exor) of a key stream to the plaintext (see Figure 2). The plaintext and ciphertext are divided into words of m bits (m is typically 1, 8, or a multiple of 8), and the ith word of the plaintext, ciphertext, and key stream is denoted with p_i, c_i, and k_i, respectively. The encryption operation can then be written as $c_i = p_i \oplus k_i$. Here $\oplus$ denotes addition modulo 2. The decryption operation is identical to the encryption (the cipher is an involution): indeed, $p_i = c_i \oplus k_i = (p_i \oplus k_i) \oplus k_i = p_i \oplus (k_i \oplus k_i) = p_i \oplus 0 = p_i$. It is clear that the m-bit key stream word k_i cannot be a constant (in that case a cryptanalyst can compute the key stream word from a single ciphertext word and the corresponding plaintext word; also repetitions in the plaintext would be visible in the ciphertext). One can show that for a strong cipher the sequence of k_i has to consist of randomly looking strings (see also Sect. 2.2).

In practice one computes the words k_i with a finite state machine. Such a machine stretches a short secret key K into a much longer key stream sequence k_i; this is called a *pseudo-random* string generator. The sequence k_i is eventually periodic. One important (but not sufficient) design criterion for the finite state machine is that the period has to be sufficient long (2^{64} is a typical lower bound). The values k_i should also be uniformly distributed; another condition is that there should be no correlations between (part of) successive words (note that cryptanalytic attacks exist which exploit correlations of less than 1 in 1 million). Formally, the sequence k_i can be parameterized with a security parameter; then on requires that the sequence satisfies every polynomial time statistical test for randomness (here polynomial means polynomial in the security parameter). Another desirable property is that no polynomial time machine can predict the next bit of the sequence (based on the previous outputs) with a probability that is significantly better than 1/2. An important (and perhaps surprising) result in theoretical cryptology by A. Yao shows that these two conditions are in fact equivalent [28].

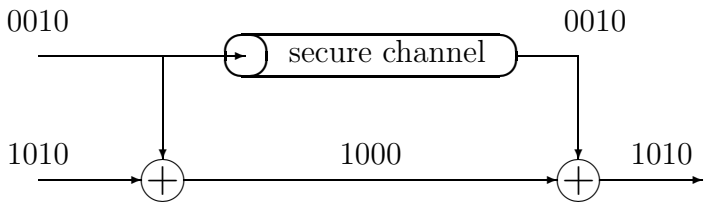

Fig. 2. An additive stream cipher

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

Block Ciphers. Block ciphers take a different approach to encryption: the plaintext is divided into larger words of n bits, called *blocks*. Every block is enciphered in the same way, using a keyed one-way permutation, i.e., a permutation on the set of n-bit strings that is controlled by a secret key. The simplest way to encrypt a plaintext using a block cipher is as follows: divide the plaintext into n-bit blocks, and encrypt these block by block. The decryption also operates on individual blocks:

$$c_i = E_K(p_i) \qquad \text{and} \qquad p_i = D_K(c_i)\,.$$

This way of using a block cipher is called the ECB (Electronic CodeBook) mode. Note that the encryption operation does *not* depend on the location in the ciphertext as is the case for additive stream ciphers.

Consider the following attack on a block cipher (the so-called tabulation attack): the cryptanalyst collects ciphertext blocks and their corresponding plaintext blocks (this is possible as part of the plaintext is often predictable); this is used to build a large table. With such a table, one can deduce information on other plaintexts encrypted under the same key. In order to preclude this attack, the value of n has to be quite large (e.g., 64 or 128) and the plaintext should not contain any repetitions (or other patterns), as these will be leaked to the ciphertext.

This shows that even if n is large, the ECB mode is not suited to encrypt plaintexts that are not random (such as text, images, etc.). This mode should only be used in very special cases, where the plaintext is already random, such as the encryption of cryptographic keys. There is however an easy way to randomize the plaintext, by using the block cipher in a different mode of operation.

The default mode of operation for a block cipher is the CBC (Cipher Block Chaining) mode. In this mode the different blocks are coupled by adding modulo 2 to a plaintext block the previous ciphertext block:

$$c_i = E_K(p_i \oplus c_{i-1}) \qquad \text{and} \qquad p_i = D_K(c_i) \oplus c_{i-1}\,.$$

Note that this 'randomizes' the plaintext, and hides patterns. To enable the encryption of the first plaintext block ($i = 1$), one defines c_0 as the initial value IV. By varying this value, one can ensure that the same plaintext is encrypted into a different ciphertext under the same key. The CBC mode allows for random access on decryption: if necessary, one can decrypt only a small part of the ciphertext.

It is also possible to use a block cipher as an additive stream cipher by feeding the output back to the input; this mode is known as the OFB (Output FeedBack) mode. A second stream mode is the CFB (Cipher FeedBack) mode; it has better synchronization properties. The modes of operation have been standardized in [6, 11].

This section has illustrated that a block cipher forms a very flexible building block. The most famous block cipher is the Data Encryption Standard (or DES) [5], which is widely used since 1977. The DES has a block size of 64 bits and a key length of 56 bits; it will be shown in Sect. 4.3 that this is no longer sufficient.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

Therefore the US government is planning to replace it by a new block cipher, the AES (Advanced Encryption Standard). Hereto an open call for algorithms has been launched in September '97; 15 candidates have been submitted by the deadline of June '98. Currently the evaluation procedure is under way. The AES will have a block length of 128 bits and a key length between 128 and 256 bits.

2.2 Security of Conventional Algorithms

An essential aspect in the choice of an encryption algorithm is the security level. In 1926 G.S. Vernam has published a simple encryption algorithm for telegraphic messages [27]. The cipher is an additive stream cipher, where the key stream consists of a completely random sequence, generated by a binary symmetric source (all bits are uniformly and identically distributed). In 1949 C. Shannon, the father of information theory, was able to prove mathematically that this scheme offers perfect security, i.e., from observing the ciphertext, the opponent cannot obtain any information on the plaintext, no matter how much computing power he has [24]. The main disadvantage of this scheme is that the secret key is exactly as long as the message (one should never reuse a key stream); C. Shannon also showed that this the best one can do if one wants perfect security. In spite of the long key, the Vernam algorithm is still used by diplomats and spies; it has been used for the 'red telephone' between Washington and Moscow. Spies used to carry key pads with random characters (it is easy to generalize the scheme to arbitrary alphabets). The security of the scheme relies on the fact that every page of the pad is used only once, which explains the name "one-time pad".

In most commercial applications one cannot afford to distribute keys which are as long as the plaintext. Therefore one uses encryption algorithms which do not offer perfect security; this implies that it is *in principle* possible to recover the plaintext and/or the secret key from the ciphertext, in the sense that one has sufficient information to do this. This does not mean that it is also possible *in practice*. For example, additive stream ciphers try to mimic the approach of the Vernam scheme by replacing the random key stream sequence by a pseudo-random sequence generated from a short key.

2.3 Public-Key Encryption

The main problem that is left unsolved by conventional cryptography is the key distribution problem. Especially in a large network it is not feasible to distribute keys between all user pairs (in a network with t users there are $t(t-1)/2$ such pairs). An alternative is to manage all keys in a central location, but this may then become a single point of failure. Public-key cryptography offers a much more elegant solution to this problem.

The concept of public-key cryptography has been invented by in 1976, independently by W. Diffie and M. Hellman [3] and by R. Merkle [18]. The key idea behind public-key cryptography is the concept of *trapdoor one-way functions*. A *one-way function* is a function that is easy to compute, but hard to invert. For example, in a conventional block cipher, the ciphertext has to be a one-way

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

function of the plaintext and the key: it is easy to compute the ciphertext from the plaintext and the key, but given the plaintext and the ciphertext it should be hard to recover the key (otherwise the block cipher would not be secure). Similarly one can show that the existence of additive stream ciphers (pseudo-random string generators) implies the existence of one-way functions. Trapdoor one-way functions are one-way function with an additional property: given some additional information (the trapdoor), it becomes possible to invert the one-way function.

With such functions Bob can send a secret message to Alice without the need for prior arrangement of a secret key. Alice chooses a trapdoor one-way function with public parameter P_A (Alice's public key) and with secret parameter S_A (Alice's secret key). Alice makes her public key widely available (she can put it on her home page, but it can also be included in special directories). Anyone who wants to send some confidential information to Alice, computes the ciphertext as the image of the plaintext under the trapdoor one-way function using the parameter P_A. Upon receipt of this ciphertext, Alice recovers the plaintext by using her trapdoor information S_A (see Figure 3). An attacker, who does not know S_A, sees only the image of the plaintext under a one-way function, and will not be able to recover the plaintext. This assumes that it is infeasible to compute S_A from P_A. Note that if one wants to send a message to Alice, one has to know Alice's public key P_A, and one has to be sure that this key really belongs to Alice (and not to Eve), since it is only the owner of the corresponding secret key who will be able to decrypt the ciphertext. Public keys do not need a secure channel for their distribution, but they do need an authentic channel. As the keys for encryption and decryption are different, and Alice and Bob have different information, public-key algorithms are also known as asymmetric algorithms.

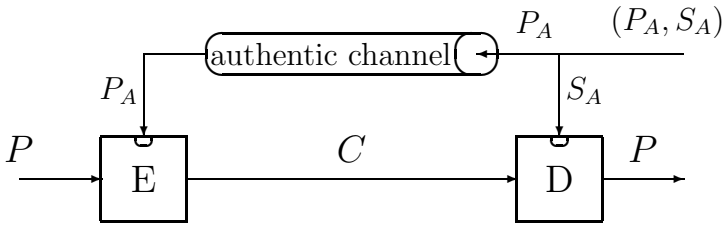

Fig. 3. Model for public-key or asymmetric encryption

The conditions which a public-key encryption algorithm has to satisfy are:
- the generation of a key pair (P_A, S_A) has to be easy;
- encryption and decryption have to be easy operations;
- it should be hard to compute the public key P_A from the corresponding secret key S_A;
- $S_A(P_A(P)) = P$.

Designing a secure public-key encryption algorithm is apparently a very difficult problem. From the large number of proposals, only a few have survived

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

(for example, almost all knapsack-based systems have been broken). The most popular algorithm is the RSA algorithm [22], which was named after its inventors (R.L. Rivest, A. Shamir, and L. Adleman). The security of RSA is based on the fact that it is relatively simple to find two large prime numbers (in 1998 large means 115 decimal digits or more) and to multiply these, while factoring their product (of 230 decimal digits) is not feasible with the current algorithms and computers.

key generation: Find 2 prime numbers p and q with at least 115 digits and compute their product, the modulus $n = p \cdot q$. Compute the Carmichael function $\lambda(n)$, the least common multiple of $p-1$ and $q-1$. Choose an encryption exponent e (at least 40 to 64 bits long), which is relatively prime to $\lambda(n)$ and compute the decryption exponent as $d = e^{-1} \bmod \lambda(n)$ (with Euclid's algorithm). The public key consists of the pair (e, n), and the secret key consists of the decryption exponent d or the pair (p, q);

encryption: represent the plaintext as an integer in the interval $[0, n-1]$ and compute the ciphertext as $C = P^e \bmod n$;

decryption: $P = C^d \bmod n$.

Without explaining the mathematical background of the algorithm, one can observe that decryption requires the extraction of modular eth roots; no algorithm is known for this problem which does not use the prime factors of n; finding the decryption exponent requires knowledge of $\lambda(n)$ and hence of the factors of n. On the other hand, this knowledge is not required for the encryption operation. For the practical use of RSA, one has to take into account many technical details: for example, the plaintext P has to be mapped (with a function that is easy to invert) to a random integer $\in [0, n-1]$ in order to avoid trivial attacks (e.g., the extraction of natural eth roots when $P^e < n$).

The more complex properties of public-key cryptography seem to require some 'high level' mathematical structure; most public-key algorithms are based on number theoretic problems (such as factoring and discrete logarithm in certain groups). While these number theoretic problems are believed to be difficult, it should be noted that since the invention of public-key cryptography significant progress has been made in factoring: the factorization record in 1975 was 39 decimal digits; in 1985 this was increased to 65 digits, and in 1994 a 130-digit modulus was factored. This evolution is due to a combination of more sophisticated factoring algorithms with progress in hardware and parallel processing. The cryptographer should take this into account by selecting sufficiently large keys for public-key algorithms.

The main advantage of public-key algorithms is the simplified key management. The main disadvantages are the larger keys (typically 64 to 256 bytes) and the slow performance: both in software and hardware public-key encryption algorithms are two to three orders of magnitude slower than conventional algorithms. For example, a 1024-bit exponentiation requires about 0.3 seconds on a 90 MHz Pentium, which corresponds to 3.4 kbit/s. On the same machine, DES runs at 16.9 Mbit/s. Because of the large difference in performance and the larger block length (which influences error propagation), one always employs

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

hybrid systems: the public-key encryption scheme is used to distribute a secret key, which is then used in a fast conventional algorithm.

3 Hashing and Signatures for Authentication

Information authentication includes two main aspects:
- *data origin authentication*, or who has originated the information;
- *data integrity*, or has the information been modified.

Other aspects which can be important are the timeliness of the information, the sequence of messages, and the destination of information. These aspects can be accounted for by using sequence numbers and time stamps in the messages and by including addressing information in the data. In data communications, the implicit authentication created by recognition of the handwriting, signature, or voice disappears. The reason is that information becomes much more vulnerable to falsification as the physical coupling between information and its bearer is lost.

Until recently it was widely believed that encryption of information (with a conventional algorithm) was sufficient for protecting its authenticity. The reasoning was that if a certain ciphertext resulted after decryption in a *meaningful* plaintext, it had to be created by someone who knew the key, and therefore it must be authentic. A few counterexamples are sufficient to refute this claim: if a block cipher is used in ECB mode, an attacker can always reorder the blocks. For any additive stream cipher (including the Vernam scheme), an opponent can always modify any plaintext bit (without knowing whether a 0 has been changed to a 1 or vice versa). The concept 'meaningful' information implicitly assumes that the information contains redundancy, which allows to distinguish genuine information from an arbitrary plaintext. However, one can envisage applications where the plaintext contains very little or no redundancy. The separation between secrecy and authentication has also been clarified by public-key cryptography: anyone who knows Alice's public key can send her a confidential message, and therefore Alice has no idea who has actually sent this message.

Two different levels of information authentication can be distinguished. If two parties trust each other and want to protect themselves against malicious outsiders, the term 'conventional message authentication' is used. In this setting, both parties are at equal footing (for example, they share the same secret key). If however a dispute arises between them, a third party (such as a judge) will not be able to resolve it (for example a judge cannot tell whether a message has been created by Alice or by Bob). If protection between two mutually distrustful parties is required (which is often the case in commercial relationships), an electronic equivalent of a manual signature is needed. In cryptographic terms this is called a digital signature.

3.1 Symmetric Authentication

The underlying idea is similar to that for encryption, where the secrecy of a large amount of information is replaced by the secrecy of a short key. In the

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

case of authentication, one replaces the authenticity of the information by the protection of a short string, which is a unique 'fingerprint' of the information. Such a 'fingerprint' is computed as a hash result. This can also be interpreted as adding a special form of redundancy to the information. This process consists of two components. First one compresses the information to a string of fixed length, with a (cryptographic) hash function. Then the resulting string (the hash result) is protected as follows:

- either the hash result is communicated over an authentic channel (e.g., it can be read over the phone). It is then sufficient to use a hash function without a secret parameter, which is called a Manipulation Detection Code or MDC;
- or the hash function uses a secret parameter (the key); it is then called a Message Authentication Code or MAC.

MDCs. If an additional (authentic) channel is available, MDCs can provide authenticity without requiring secret keys. Moreover an MDC is a flexible primitive, which can be used for a variety of other cryptographic applications. An MDC has to satisfy the following conditions:

- it should be hard to find an input with a given hash result (preimage resistance);
- it should be hard to find a second input with the same hash result as a given input (2nd preimage resistance);
- it should be hard to find two different inputs with the same hash result (collision resistance).

An MDC satisfying these three conditions is called a *collision resistant* hash function. For a strong hash function with an n-bit result, solving one of the first two problems requires about 2^n evaluations of the hash function. This implies that $n = 64 \ldots 80$ is sufficient. However, finding collisions is much easier: one will find with high probability a collision in a set of hash results corresponding to $2^{n/2}$ inputs. This implies that collision resistant hash functions need a hash result of 128 to 160 bits. This last property is also known as the birthday paradox based on the following observation: within a group of 24 persons the probability that there are two persons with the same birthday is about 50%. The reason is that a group of this size contains 276 different pairs of persons, which is a large fraction of the 365 days in a year. Note that the birthday paradox plays an essential role in the security of many cryptographic primitives (cf. Sect. 4.3). Examples of MDCs in use today are RIPEMD-160 and SHA-1; both have been standardized in [12]. Not all applications need collision resistant hash functions; sometimes (2nd) preimage resistance is sufficient.

MACs. MACs have been used for more than twenty years in electronic transactions in the banking environment. They require the exchange of a secret key between the communicating parties. The MAC corresponding to a message is a complex function of every bit of the message and every bit of the key; it should

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

be infeasible to derive the key from observing a number of text/MAC pairs, or to compute or predict a MAC without knowing the secret key.

A MAC is used as follows: Alice computes for her message P the value $\mathrm{MAC}_K(P)$ and appends this MAC to the message (here MAC denotes both the function and its result). Bob recomputes the value of $\mathrm{MAC}_K(P)$ based on the received message P, and verifies whether it matches the received MAC. If the answer is positive, he accepts the message as authentic, i.e., as a genuine message from Alice. Eve, the active eavesdropper, can modify the message P to P', but she is not able to compute the corresponding MAC value $\mathrm{MAC}(P')$, as she is not privy to the secret key K. For a secure MAC, the best Eve can do is guessing the MAC. In that case, Bob can detect the modification with high probability: for an n-bit MAC Eve's probability of success is only $1/2^n$. The value of n lies typically between 32 and 64. Note that if encryption and authentication are combined, the key for encryption and authentication need to be different. Moreover, the preferred option is to compute the MAC on the plaintext.

A popular way to compute a MAC is to encrypt the message with a block cipher using the CBC mode (yet another use of a block cipher), and to keep only part of the bits of the last block as the MAC. However, recent research has indicated that this approach is less secure than previously believed [21]; again, the birthday paradox plays a role in this work.

For a MAC, the equivalent of the Vernam scheme exists. This implies that one can design a MAC algorithm which is unconditionally secure, in the sense that the security of the MAC is independent of the computing power of the opponent. The requirement is again that the secret key is used only once. The basic idea of this approach is due to G.J. Simmons and dates back to the seventies (see for example [25]). It turns out that these algorithms can be computationally very efficient, since the properties required from this primitive are combinatorial rather than cryptographic. Recent constructions are therefore one order of magnitude faster than other cryptographic primitives (encryption algorithms, hash functions), and achieve speeds up to 1 Gbit/s on fast processors [9]. A simple example is described here, which is derived from Reed-Solomon codes for error-correction [13]. The key consists of two n-bit words denoted with K_1 and K_2. The plaintext is divided into t n-bit words, denoted with p_1 through p_t. The MAC, which consists of a single n-bit word, is computed based on a simple polynomial evaluation:

$$\mathrm{MAC}_{K_1,K_2}(x) = K_1 + \sum_{i=1}^{t} p_i \cdot (K_2)^i \,,$$

where addition and multiplication are to be computed in the finite field with 2^n elements. It can be proved that the probability of creating another valid message/MAC pair is upper bounded by $t/2^n$. A practical choice is $n = 64$, which results in a 128-bit key. For messages up to 1 Mbyte, the success probability of a forgery is then less than $1/2^{47}$. Note that it turns out to be possible to reuse K_2; however, for every message a new key K_1 is required. This key could be generated from a short initial key using an additive stream cipher, but then

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

the unconditional security is lost. However, one can argue that it is easier to understand the security of this scheme than that of a computationally secure MAC.

3.2 Digital Signatures

A digital signature is the electronic equivalent of a manual signature on a document. It provides a strong binding between the document and a person, and in case of a dispute, a third party can decide whether or not the signature is valid. Of course a digital signature will not bind a person and a document, but will bind a key and a document. Additional measures are then required to bind the person to his or her key. Note that for a MAC, both Alice and Bob can compute the MAC, hence a third party cannot distinguish between them. While block ciphers (and even one-way functions) can be used to construct digital signatures, the most elegant and efficient constructions for digital signature rely on public-key cryptography.

If Alice wants to sign some information P intended for Bob, she adds some redundancy to the information, resulting in $\tilde{P}$, and decrypts the resulting text with her secret key. This operation can only be carried out by Alice. Upon receipt of the signature, Bob encrypts it using Alice's public key, and verifies that the information $\tilde{P}$ has the prescribed redundancy. If so, he accepts the signature on P as valid. Such a digital signature (which is a signature with 'message recovery') imposes an additional condition on the public-key system: $P_A(S_A(P)) = P$. Note that anyone who knows Alice's public key can verify the signature. The RSA public-key encryption scheme is a bijection (a trapdoor one-way permutation), and thus it allows for the construction of digital signatures with message recovery. We leave it as an exercise to the reader to show why the redundancy is essential in this approach.

If Alice wants to sign very long messages (without encrypting them), this approach results in signatures that are as long as the message. Moreover, signing with a public-key system is a relatively slow operation. In order to solve these problems, Alice does not sign the information itself, but the hash result of the information computed with an MDC. The signature now consists of a single block, which is appended to the information (this is called a digital signature 'with appendix'). In order to verify such a signature, Bob recomputes the MDC of the message and encrypts the signature with Alice's public key. If both operations give the same result, Bob accepts the signature as valid. MDCs used in this way need to be collision resistant: otherwise Alice can sign a message P, and later be held accountable for a fraudulent message P' with the same MDC (and thus with the same signature).

Note that there exist other signature schemes with appendix (such as the DSA [7]), which are not derived immediately from a public-key encryption scheme. For these schemes one can define a 'signing operation' (using the secret key) and a 'verification operation' (using the public key), without referring to 'decryption' and 'encryption' operations.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

4 Analysis and Design of Conventional Cryptographic Algorithms

In this section we compare three approaches to the design of cryptographic algorithms. Next we describe the typical phases in the life of an algorithm. Then we contrast brute force and shortcut attacks, public and secret algorithms, and weak and strong algorithms.

4.1 Three Approaches in Cryptography

Present day cryptology tries to develop provably secure and efficient cryptographic algorithms. Often such algorithms are not available; therefore cryptographic algorithms are studied following three approaches: the information theoretic approach, the complexity theoretic approach, and the system based approach. These approaches differ in the assumptions about the capabilities of an opponent, in the definition of a cryptanalytic success, and in the notion of security.

The most desirable from the viewpoint of the cryptographer are unconditionally secure algorithms; this design approach is also known as the *information theoretic* approach. However, few such schemes exist: examples are the Vernam scheme (Sect. 2.2), and the MAC based on Reed-Solomon codes (Sect. 3.1). While they are computationally very efficient, the cost in terms of key material may be prohibitively large (certainly for the Vernam scheme). For most applications one has to live with schemes which offer only conditional security.

A second approach is to reduce the security of his scheme to that of other well known difficult problems, or to that of other cryptographic primitives. The *complexity theoretic* approach starts from an abstract model for computation, and assumes that the opponent has limited computing power within this model [8]. This approach has many positive sides:

- It forces the formulation of exact definitions, and to state clearly the security properties and assumptions.
- Once the proofs are written down, anyone can verify them and decide whether or not they are correct.

However, this approach also has some limitations:

- Many cryptographic applications need building blocks, such are one-way functions, one-way permutations, and pseudo-random functions, which cannot be reduced to other primitives. In terms of the existence of such primitives, complexity theory has only very weak results: in non-uniform complexity (Boolean circuits) the best proved thus far is that there exist functions which are twice as hard to invert as to compute, which is far too weak to be of any use in cryptography [10].
- Sometimes the resulting scheme is not very efficient, or the security reduction is quite loose: for example, the correct properties are proved, but the proof is only asymptotic and gives no indication of the exact security level for a concrete instance.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

This implies that for many instances, the cryptographer has to rely on the *system-based* or *practical approach.* This approach tries to produce practical solutions; the security estimates are based on the best algorithm known to break the system and on realistic estimates of the necessary computing power or dedicated hardware to carry out the algorithm. By trial and error procedures, several *cryptanalytic principles* have emerged, and it is the goal of the designer to avoid attacks based on these principles. The second aspect is to design *building blocks with provable properties*, and to assemble such basic building blocks to design cryptographic primitives.

4.2 Life Cycle of a Cryptographic Algorithm

A cryptographic algorithm usually starts with a new idea of a cryptographer. A first step should always consist of an evaluation of the resulting algorithm, in which the cryptographer tries to determine whether or not the scheme is secure. If the scheme is unconditionally secure, he has to write the proofs, and to convince himself that the model is correct and matches the application. For computational security, it is again very important to write down security proofs, and to check these for subtle flaws. Moreover, one has to assess whether the assumptions behind the proofs are realistic. For the system-based approach, it is important to prove partial results, and to write down arguments which should convince others of the security of the algorithm. Often such cryptographic algorithms have security parameters (the number of steps, the size of the key, ...); it is then very important to give lower bounds for these parameters, and to indicated the value of the parameters which corresponds to a certain security level.

The next step is the publication of the algorithm at a conference, in a journal, or in an Internet Request for Comment (RFC). This (hopefully) results in an *independent* evaluation of the algorithm. Often more or less subtle flaws are then discovered by other researchers. This can vary from small errors in proofs, to complete security breaks. Depending on the outcome, this can lead to a small fix of the scheme or to abandoning the idea altogether. Sometimes such weaknesses can be found 'in real-time' when the author is presenting his ideas at a conference, but often evaluating a cryptographic algorithm is a very time consuming task; for example, the design effort of the Data Encryption Standard (DES) has been more than 17 man-years, and the open academic evaluation since has taken a multiple of this effort. Cryptanalysis is quite destructive; in this respect it differs from usual scientific activities, even when proponents of competing theories criticize each other.

Few algorithms survive the evaluation stage; ideally, this stage should last for several years. The survivors can be integrated into products and find their way to the market. Sometimes they are standardized by organizations such as NIST (National Institute of Standards and Technology, US), IEEE, IETF, or ISO.

As will be explained below, even if no new security weaknesses are found, the security of a cryptographic algorithm degrades over time; if the algorithm is not modular, the moment will come when it has to be taken out of service.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

4.3 Brute Force Attacks Versus Shortcut Attacks

A detailed description of the evaluation procedures for cryptographic algorithms is beyond the scope of this paper. We restrict ourselves to explaining the difference between brute force attacks and shortcut attacks.

Brute Force Attacks. Brute force attacks are attacks which exist against any cryptographic algorithm that is conditionally secure, no matter how it works internally. These attacks only depend on the size of the external parameters of the algorithm, such as the block length of a block cipher, or the key length of any encryption algorithm or MAC. It is the task of the designer to choose the external parameters in such a way that brute force attack are infeasible.

A typical brute force attack against an encryption algorithm or a MAC is an exhaustive key search; it is equivalent to breaking into a safe by trying all the combinations of the lock. The lock should be designed such that this is not feasible in a reasonable amount of time. This attack requires only a few known plaintext/ciphertext (or plaintext/MAC) pairs, which one can always obtain in practice. It can be precluded by increasing the key length: adding one bit to the key doubles the time for exhaustive key search. One should also ensure that the key is selected uniformly at random in the key space.

On a standard PC, trying a single key for a typical algorithm requires a few microseconds. For example, a 40-bit key (which is at present the maximum value allowed by the US government for general purpose export) will be recovered after a few hundred hours. If a LAN with 100 machines can be used, one can find the key in a few hours. For a 56-bit key such as DES (which can be exported from the US under restrictive conditions), a key search requires a few months if several thousand machines are available (as has been demonstrated in the first half of 1997). However, if dedicated hardware is used, a different picture emerges. Recently a 250 000 US$ machine has been built that finds a 56-bit DES key in about 50 hours [4]; the design (that required 50% of the cost) has been made available for free.

One should also take into account "Moore's law" [23], which states that computers double their speed every 18 months (for the same cost). This implies that a 64-bit key, which offers a reasonable security level for the time being, is probably not sufficient for data which needs to be protected for 10 years. Such applications will need keys of at least 80 bits. As the cost of increasing the key size is quite low, it is advisable to design new algorithms with variable key size up to 128...256 bits.

There exist many other brute force attacks. For example, it turns out the security of a block cipher in the CBC mode is decreased by what is called the 'matching ciphertext' attack. As a consequence of the birthday paradox, after $2^{n/2}$ encryptions with a single key, information on the plaintext starts to leak (due to matches in the internal memory, which correspond to matching ciphertexts). This attack can be a problem for present day block ciphers with a 64-bit block length. It can only be prevented by designing new block ciphers with larger block lengths (128 or more), or by changing the key frequently.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

Shortcut Attacks. Many algorithms are less secure than suggested by the size of their external parameters. It is often possible to find more effective attacks than trying all keys. Assessing the strength of an algorithm requires cryptanalytic skills and experience, and often hard work. During the last 10 years powerful new tools have been developed: this includes differential cryptanalysis [1], which analyzes the propagation of differences through cryptographic algorithms, linear cryptanalysis [16], which is based on the propagation of bit correlations, and fast correlation attacks on stream ciphers [17].

The design of new algorithms according to the system-based approach is not a memoryless process: when new cryptanalytic techniques are developed, the cryptographers invent new designs which provide complete (or at least improved) resistance against these new attacks. In this way cryptology develops by trial and error procedures.

4.4 Public Versus Secret Algorithms

The open and independent evaluation process described in Sect. 4.2 offers a strong argument for publishing all details of a cryptographic algorithm. Publishing the algorithm opens it up for public scrutiny, and is the best way to guarantee that it is as strong as claimed. (Note that a public algorithm should not be confused with a public-key algorithm.) Published algorithms can be standardized, and will be available from more than one source.

Nevertheless, certain governments and organizations prefer to keep their algorithms secret. They argue (correctly) that obtaining the algorithm raises an additional barrier for the attacker. Moreover, governments want to protect their know-how on the design of cryptographic algorithms. (However, obtaining a description of the algorithm is often not harder than just bribing one person.) This approach is acceptable, provided that sufficient experience and resources are available for *independent* evaluation and re-evaluation of the algorithm.

4.5 Insecure Versus Secure Algorithms

In spite of the fact that secure cryptographic algorithms are available, which offer good performance, in many applications one encounters very insecure cryptographic algorithms. For example, popular software sometimes 'encrypts' data by adding a constant key word to all data words. Several reasons can be indicated for this:

- one excuse is performance: while it is true that adding a constant will always be faster than strong encryption, it should be noted that in software, current encryption algorithms achieve between 20 and 400 Mbit/s; this is sufficient for many applications;
- legal and/or export restrictions: for national security reasons, certain countries (such as the USA) do not allow the export of strong encryption algorithms; some countries (such as France) do not allow for strong encryption within their territory (unless the keys are handed over to the government);

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

- commercial pressure: companies often rush their security solutions to market, without allowing for sufficient time for the slow evaluation process;
- evolution of computing power: the strength of a cryptographic algorithm erodes over time because of Moore's law; often there exists a large inertia to replace or upgrade an algorithm. A typical example is the DES, that is still widely used in spite of a 56-bit key.
- evolution of cryptanalysis: if designers are not aware of the latest developments in cryptanalysis, it is quite likely that their algorithms will not resist these attacks. For example, the FEAL block cipher with 8 rounds, which was published in 1987, can now be broken with only 10 chosen plaintexts.

5 Concluding Remarks

Securing an application should be based on a careful analysis of the risks and vulnerabilities; this should lead to understanding the security requirements for the data and the communication channels. The next step consists of selecting the right mix of cryptographic algorithms to satisfy these requirements. A very important aspect is the underlying key management infrastructure, which ensures that private and public keys can be established and maintained throughout the system in a secure way. This is where cryptography meets the constraints of the real world.

This paper only scratches the surface of modern cryptology, as the discussion is restricted to a few basic techniques. Other problems solved in cryptography include secure identification, secure sharing of secrets, electronic cash, and copyright protection. Many interesting problems are studied under the umbrella of secure multi-party computation; examples are electronic elections, and the generation and verification of digital signatures in a distributed way.

References

1. E. Biham, A. Shamir, *"Differential Cryptanalysis of the Data Encryption Standard,"* Springer-Verlag, 1993.
2. D.W. Davies, W.L. Price, *"Security for Computer Networks. An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer,"* (2nd Ed.), Wiley, 1989.
3. W. Diffie, M.E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, Vol. IT–22, No. 6, 1976, pp. 644–654.
4. EFF, *"Cracking DES. Secrets of Encryption Research, Wiretap Politics & Chip Design,"* O'Reilly, May 1998.
5. FIPS 46, *"Data Encryption Standard,"* Federal Information Processing Standard, NBS, U.S. Dept. of Commerce, January 1977 (revised as FIPS 46-2:1993).
6. FIPS 81, *"DES Modes of Operation,"* Federal Information Processing Standard, NBS, US Dept. of Commerce, December 1980.
7. FIPS 186, *"Digital Signature Standard,"* Federal Information Processing Standard, NIST, US Dept. of Commerce, May 1994.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.

8. M.R. Garey, D.S. Johnson, *"Computers and Intractability: A Guide tot the Theory of NP-Completeness,"* W.H. Freeman and Company, San Francisco, 1979.
9. S. Halevi, H. Krawczyk, "MMH: software message authentication in the Gbit/second rates," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 172–189.
10. A.P.L. Hiltgen, "Construction of feebly-one-way families of permutations," *Proc. Auscrypt'92, LNCS 718*, J. Seberry, Y. Zheng, Eds., Springer-Verlag, 1993, pp. 422–434.
11. ISO/IEC 10116, *"Information technology – Security techniques – Modes of operation of an n-bit block cipher algorithm,"* 1996.
12. ISO/IEC 10118, *"Information technology – Security techniques – Hash-functions, Part 3: Dedicated hash-functions,"* 1998.
13. G.A. Kabatianskii, T. Johansson, B. Smeets, "On the cardinality of systematic A-codes via error correcting codes," *IEEE Trans. on Information Theory*, Vol. IT–42, No. 2, 1996, pp. 566–578.
14. D. Kahn, *"The Codebreakers. The Story of Secret Writing,"* MacMillan, New York, 1967.
15. N. Koblitz, *"A Course in Number Theory and Cryptography,"* Springer-Verlag, 1987.
16. M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Proc. Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.
17. W. Meier, O. Staffelbach, "Fast correlation attacks on stream ciphers," *J. of Cryptology*, Vol. 1, 1989, pp. 159–176.
18. R. Merkle, *"Secrecy, Authentication, and Public Key Systems,"* UMI Research Press, 1979.
19. A.J. Menezes, P.C. van Oorschot, S. Vanstone, *"Handbook of Applied Cryptography,"* CRC Press, 1996.
20. *"State of the Art and Evolution of Computer Security and Industrial Cryptography," LNCS 741*, B. Preneel, R. Govaerts, J. Vandewalle, Eds., Springer-Verlag, 1993.
21. B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.
22. R.L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Comm. ACM*, Vol. 21, No. 2, 1978, pp. 120–126.
23. R.R. Schaller, "Moore's law: past, present, and future," *IEEE Spectrum*, Vol. 34, No. 6, June 1997, pp. 53–59.
24. C.E. Shannon, "Communication theory of secrecy systems," *Bell System Techn. J.*, Vol. 28, No. 4, 1949, pp. 656–715.
25. *"Contemporary Cryptology: The Science of Information Integrity,"* G.J. Simmons, Ed., IEEE Press, 1991.
26. D. Stinson, *"Cryptography. Theory and Practice,"* CRC Press, 1995.
27. G.S. Vernam, "Cipher printing telegraph system for secret wire and radio telegraph communications," *J. Am. Inst. Electrical Engineers*, Vol. XLV, 1926, pp. 109–115.
28. A.C. Yao, "Theory and applications of trapdoor functions," *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, IEEE, 1982, pp. 80–91.

Appeared in *SOFSEM 1998: 25th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science 1521, B. Rovan (ed.), Springer-Verlag, pp. 204–221, 1998.