Post-Quantum Cryptography

Daniel J. Bernstein • Johannes Buchmann
Erik Dahmen
Editors

Post-Quantum Cryptography

Springer

Editors

Daniel J. Bernstein
Department of Computer Science
University of Illinois, Chicago
851 S. Morgan St.
Chicago IL 60607-7053
USA
djb@cr.yp.to

Johannes Buchmann
Erik Dahmen
Technische Universität Darmstadt
Department of Computer Science
Hochschulstr. 10
64289 Darmstadt
Germany
buchmann@cdc.informatik.tu-darmstadt.de
dahmen@cdc.informatik.tu-darmstadt.de

ISBN: 978-3-540-88701-0 e-ISBN: 978-3-540-88702-7

Library of Congress Control Number: 2008937466

Mathematics Subject Classification Numbers (2000): 94A60

Cover design: WMX Design GmbH, Heidelberg

Printed on acid-free paper

springer.com

Preface

The first International Workshop on Post-Quantum Cryptography took place at the Katholieke Universiteit Leuven in 2006. Scientists from all over the world gave talks on the state of the art of quantum computers and on cryptographic schemes that may be able to resist attacks by quantum computers. The speakers and the audience agreed that post-quantum cryptography is a fascinating research challenge and that, if large quantum computers are built, post-quantum cryptography will be critical for the future of the Internet. So, during one of the coffee breaks, we decided to edit a book on this subject. Springer-Verlag promptly agreed to publish such a volume. We approached leading scientists in the respective fields and received favorable answers from all of them. We are now very happy to present this book. We hope that it serves as an introduction to the field, as an overview of the state of the art, and as an encouragement for many more scientists to join us in investigating post-quantum cryptography.

We would like to thank the contributors to this volume for their smooth collaboration. We would also like to thank Springer-Verlag, and in particular Ruth Allewelt and Martin Peters, for their support. The first editor would like to additionally thank Tanja Lange for many illuminating discussions regarding post-quantum cryptography and for initiating the Post-Quantum Cryptography workshop series in the first place.

Chicago and Darmstadt,
December 2008

Daniel J. Bernstein
Johannes A. Buchmann
Erik Dahmen

Contents

List of Contributors

Daniel J. Bernstein
University of Illinois at Chicago
djb@cr.yp.to

Johannes Buchmann
Technische Universität Darmstadt
buchmann@cdc.informatik.
tu-darmstadt.de

Erik Dahmen
Technische Universität Darmstadt
dahmen@cdc.informatik.
tu-darmstadt.de

Jintai Ding
University of Cincinnati
ding@math.uc.edu

Sean Hallgren
The Pennsylvania State University

Daniele Micciancio
University of California, San Diego
daniele@cs.ucsd.edu

Raphael Overbeck
EPFL, I&C, LASEC
raphael.overbeck@epfl.ch

Oded Regev
Tel-Aviv University

Nicolas Sendrier
INRIA Rocquencourt
nicolas.sendrier@inria.fr

Michael Szydlo
Akamai Technologies
mike@szydlo.com

Ulrich Vollmer
Berlin, Germany
ac@u.vollmer.name

Bo-Yin Yang
Academia Sinica
by@moscito.org

Introduction to post-quantum cryptography

Daniel J. Bernstein

Department of Computer Science, University of Illinois at Chicago.

1 Is cryptography dead?

Imagine that it's fifteen years from now and someone announces the successful construction of a large quantum computer. The *New York Times* runs a front-page article reporting that all of the public-key algorithms used to protect the Internet have been broken. Users panic. What exactly will happen to cryptography?

Perhaps, after seeing quantum computers destroy RSA and DSA and ECDSA, Internet users will leap to the conclusion that cryptography is dead; that there is no hope of scrambling information to make it incomprehensible to, and unforgeable by, attackers; that securely storing and communicating information means using expensive physical shields to prevent attackers from seeing the information—for example, hiding USB sticks inside a locked briefcase chained to a trusted courier's wrist.

A closer look reveals, however, that there is no justification for the leap from "quantum computers destroy RSA and DSA and ECDSA" to "quantum computers destroy cryptography." There are many important classes of cryptographic systems beyond RSA and DSA and ECDSA:

- **Hash-based cryptography.** The classic example is Merkle's hash-tree public-key signature system (1979), building upon a one-message-signature idea of Lamport and Diffie.
- **Code-based cryptography.** The classic example is McEliece's hidden-Goppa-code public-key encryption system (1978).
- **Lattice-based cryptography.** The example that has perhaps attracted the most interest, not the first example historically, is the Hoffstein–Pipher–Silverman "NTRU" public-key-encryption system (1998).
- **Multivariate-quadratic-equations cryptography.** One of many interesting examples is Patarin's "HFE^{v-}" public-key-signature system (1996), generalizing a proposal by Matsumoto and Imai.

- **Secret-key cryptography.** The leading example is the Daemen–Rijmen "Rijndael" cipher (1998), subsequently renamed "AES," the Advanced Encryption Standard.

All of these systems are believed to resist classical computers *and* quantum computers. Nobody has figured out a way to apply "Shor's algorithm"—the quantum-computer discrete-logarithm algorithm that breaks RSA and DSA and ECDSA—to any of these systems. Another quantum algorithm, "Grover's algorithm," does have some applications to these systems; but Grover's algorithm is not as shockingly fast as Shor's algorithm, and cryptographers can easily compensate for it by choosing somewhat larger key sizes.

Is there a better attack on these systems? Perhaps. This is a familiar risk in cryptography. This is why the community invests huge amounts of time and energy in cryptanalysis. Sometimes cryptanalysts find a devastating attack, demonstrating that a system is useless for cryptography; for example, every usable choice of parameters for the Merkle–Hellman knapsack public-key encryption system is easily breakable. Sometimes cryptanalysts find attacks that are not so devastating but that force larger key sizes. Sometimes cryptanalysts study systems for years without finding any improved attacks, and the cryptographic community begins to build confidence that the best possible attack has been found—or at least that real-world attackers will not be able to come up with anything better.

Consider, for example, the following three factorization attacks against RSA:

- 1978: The original paper by Rivest, Shamir, and Adleman mentioned a new algorithm, Schroeppel's "linear sieve," that factors any RSA modulus n—and thus breaks RSA—using $2^{(1+o(1))(\lg n)^{1/2}(\lg\lg n)^{1/2}}$ simple operations. Here $\lg = \log_2$. Forcing the linear sieve to use at least 2^b operations means choosing n to have at least $(0.5 + o(1))b^2/\lg b$ bits.

  Warning: $0.5 + o(1)$ means something that *converges* to 0.5 as $b \to \infty$. It does not say anything about, e.g., $b = 128$. Figuring out the proper size of n for $b = 128$ requires looking more closely at the speed of the linear sieve.

- 1988: Pollard introduced a new factorization algorithm, the "number-field sieve." This algorithm, as subsequently generalized by Buhler, Lenstra, and Pomerance, factors any RSA modulus n using $2^{(1.9\ldots+o(1))(\lg n)^{1/3}(\lg\lg n)^{2/3}}$ simple operations. Forcing the number-field sieve to use at least 2^b operations means choosing n to have at least $(0.016\ldots + o(1))b^3/(\lg b)^2$ bits.

  Today, twenty years later, the fastest known factorization algorithms for classical computers still use $2^{(\text{constant}+o(1))(\lg n)^{1/3}(\lg\lg n)^{2/3}}$ operations. There have been some improvements in the constant and in the details of the $o(1)$, but one might guess that $1/3$ is optimal, and that choosing n to have roughly b^3 bits resists all possible attacks by classical computers.

- 1994: Shor introduced an algorithm that factors any RSA modulus n using $(\lg n)^{2+o(1)}$ simple operations on a quantum computer of size $(\lg n)^{1+o(1)}$. Forcing this algorithm to use at least 2^b operations means choosing n to have at least $2^{(0.5+o(1))b}$ bits—an intolerable cost for any interesting value of b. See the "Quantum computing" chapter of this book for much more information on quantum algorithms.

Consider, for comparison, attacks on another thirty-year-old public-key cryptosystem, namely McEliece's hidden-Goppa-code encryption system. The original McEliece paper presented an attack that breaks codes of "length n" and "dimension $n/2$" using $2^{(0.5+o(1))n/\lg n}$ operations. Forcing this attack to use 2^b operations means choosing n at least $(2+o(1))b \lg b$. Several subsequent papers have reduced the number of attack operations by an impressively large factor, roughly $n^{\lg n} = 2^{(\lg n)^2}$, but $(\lg n)^2$ is much smaller than $0.5n/\lg n$ if n is large; the improved attacks still use $2^{(0.5+o(1))n/\lg n}$ operations. One can reasonably guess that $2^{(0.5+o(1))n/\lg n}$ is best possible. Quantum computers don't seem to make much difference, except for reducing the constant 0.5.

If McEliece's cryptosystem is holding up so well against attacks, why are we not *already* using it instead of RSA? The answer, in a nutshell, is efficiency, specifically key size. McEliece's public key uses roughly $n^2/4 \approx b^2(\lg b)^2$ bits, whereas an RSA public key—assuming the number-field sieve is optimal and ignoring the threat of quantum computers—uses roughly $(0.016\ldots)b^3/(\lg b)^2$ bits. If b were extremely large then the $b^{2+o(1)}$ bits for McEliece would be smaller than the $b^{3+o(1)}$ bits for RSA; but real-world security levels such as $b = 128$ allow RSA key sizes of a few thousand bits, while McEliece key sizes are closer to a million bits.

Figure 1 summarizes the process of designing, analyzing, and optimizing cryptographic systems before the advent of quantum computers; Figure 2 summarizes the same process after the advent of quantum computers. Both pictures have the same structure:

- cryptographers design systems to scramble and unscramble data;
- cryptanalysts break some of those systems;
- algorithm designers and implementors find the fastest unbroken systems.

Cryptanalysts in Figure 1 use the number-field sieve for factorization, the Lenstra–Lenstra–Lovasz algorithm for lattice-basis reduction, the Faugère algorithms for Gröbner-basis computation, and many other interesting attack algorithms. Cryptanalysts in Figure 2 have all of the same tools in their arsenal *plus* quantum algorithms, notably Shor's algorithm and Grover's algorithm. All of the most efficient unbroken public-key systems in Figure 1, perhaps not coincidentally, take advantage of group structures that can also be exploited by Shor's algorithm, so those systems disappear from Figure 2, and the users end up with different cryptographic systems.

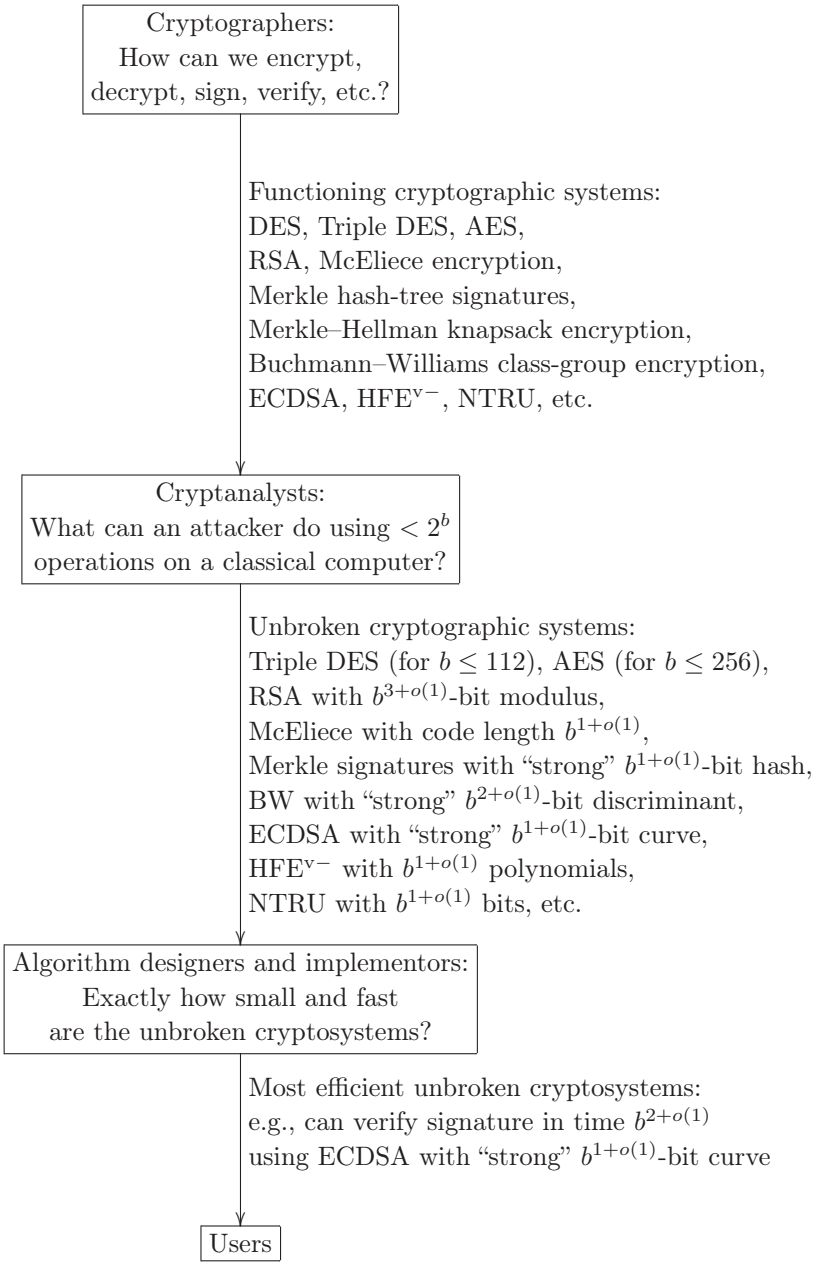

Fig. 1. Pre-quantum cryptography. Warning: Sizes and times are simplified to $b^{1+o(1)}$, $b^{2+o(1)}$, etc. Optimization of any specific b requires a more detailed analysis; e.g., low-exponent RSA verification is faster than ECDSA verification for small b.

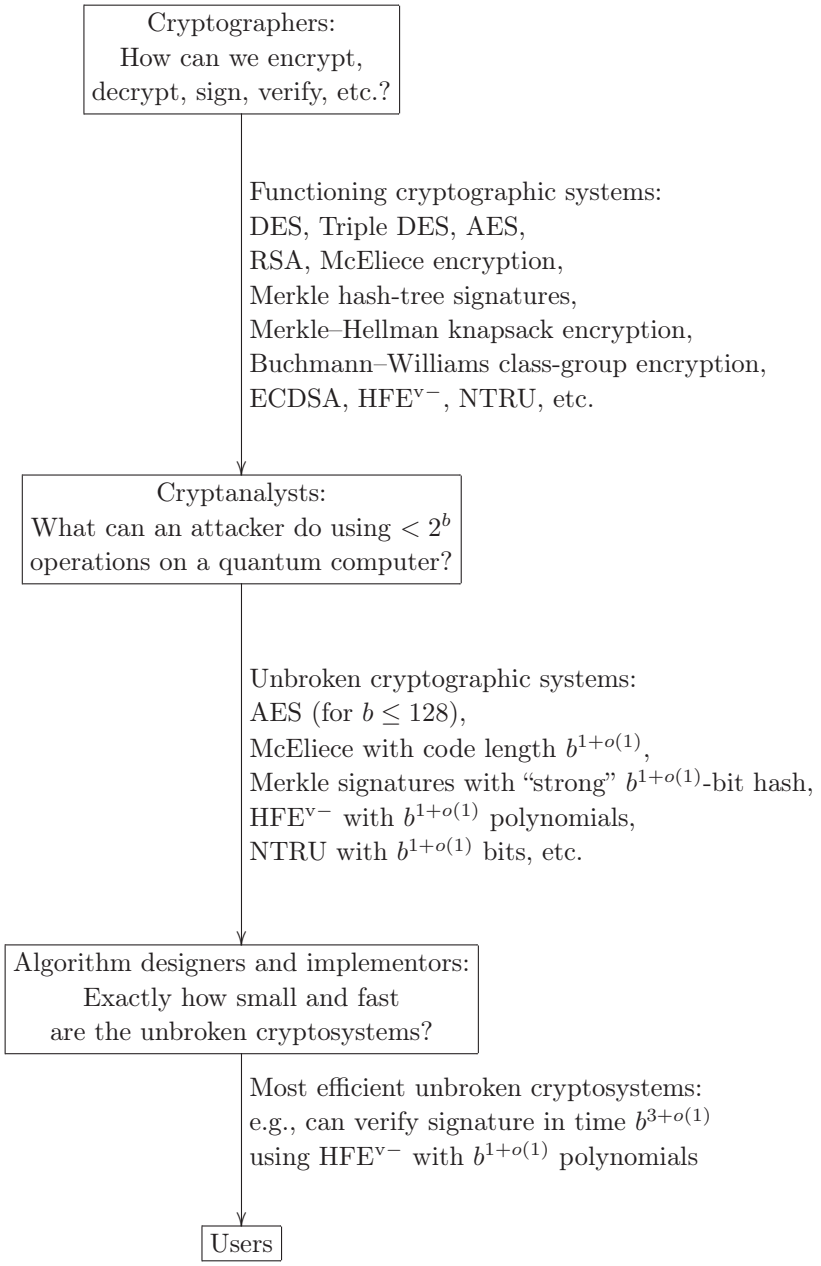

Fig. 2. Post-quantum cryptography. Warning: Sizes and times are simplified to $b^{1+o(1)}$, $b^{2+o(1)}$, etc. Optimization of any specific b requires a more detailed analysis.

2 A taste of post-quantum cryptography

Here are three specific examples of cryptographic systems that appear to be extremely difficult to break—even for a cryptanalyst armed with a large quantum computer.

Two of the examples are public-key signature systems; one of the examples is a public-key encryption system. All three examples are parametrized by b, the user's desired security level. Many more parameters and variants appear later in this book, often allowing faster encryption, decryption, signing, and verification with smaller keys, smaller signatures, etc.

I chose to focus on public-key examples—a focus shared by most of this book—because quantum computers seem to have very little effect on secret-key cryptography, hash functions, etc. Grover's algorithm forces somewhat larger key sizes for secret-key ciphers, but this effect is essentially uniform across ciphers; today's fastest pre-quantum 256-bit ciphers are also the fastest candidates for post-quantum ciphers at a reasonable security level. (There are a few specially structured secret-key ciphers that can be broken by Shor's algorithm, but those ciphers are certainly not today's fastest ciphers.) For an introduction to state-of-the-art secret-key ciphers I recommend the following book: Matthew Robshaw and Olivier Billet (editors), *New stream cipher designs: the eSTREAM finalists*, Lecture Notes in Computer Science **4986**, Springer, 2008, ISBN 978–3–540–68350–6.

2.1 A hash-based public-key signature system

This signature system requires a standard cryptographic hash function H that produces $2b$ bits of output. For $b = 128$ one could choose H as the SHA-256 hash function. Over the last few years many concerns have been raised regarding the security of popular hash functions, and over the next few years NIST will run a competition for a SHA-256 replacement, but all known attacks against SHA-256 are extremely expensive.

The signer's public key in this system has $8b^2$ bits: e.g., 16 kilobytes for $b = 128$. The key consists of $4b$ strings $y_1[0], y_1[1], y_2[0], y_2[1], \ldots, y_{2b}[0], y_{2b}[1]$, each string having $2b$ bits.

A signature of a message m has $2b(2b + 1)$ bits: e.g., 8 kilobytes for $b = 128$. The signature consists of $2b$-bit strings $r, x_1, \ldots, x_{2b}$ such that the bits $(h_1, \ldots, h_{2b})$ of $H(r, m)$ satisfy $y_1[h_1] = H(x_1)$, $y_2[h_2] = H(x_2)$, and so on through $y_{2b}[h_{2b}] = H(x_{2b})$.

How does the signer find x with $H(x) = y$? Answer: The signer starts by generating a secret x and then computes $y = H(x)$. Specifically, the signer's secret key has $8b^2$ bits, namely $4b$ independent uniform random strings $x_1[0], x_1[1], x_2[0], x_2[1], \ldots, x_{2b}[0], x_{2b}[1]$, each string having $2b$ bits. The signer computes the public key $y_1[0], y_1[1], y_2[0], y_2[1], \ldots, y_{2b}[0], y_{2b}[1]$ as $H(x_1[0]), H(x_1[1]), H(x_2[0]), H(x_2[1]), \ldots, H(x_{2b}[0]), H(x_{2b}[1])$.

To sign a message m, the signer generates a uniform random string r, computes the bits $(h_1, \ldots, h_{2b})$ of $H(r, m)$, and reveals $(r, x_1[h_1], \ldots, x_{2b}[h_{2b}])$ as a signature of m. The signer then discards the remaining x values and refuses to sign any more messages.

What I've described so far is the "Lamport–Diffie one-time signature system." What do we do if the signer wants to sign more than one message?

An easy answer is "chaining." The signer includes, in the signed message, a newly generated public key that will be used to sign the next message. The verifier checks the first signed message, including the new public key, and can then check the signature of the next message; the signature of the nth message includes all $n - 1$ previous signed messages. More advanced systems, such as Merkle's hash-tree signature system, scale logarithmically with the number of messages signed.

To me hash-based cryptography is a convincing argument for the existence of secure post-quantum public-key signature systems. Grover's algorithm is the fastest quantum algorithm to invert *generic* functions, and is widely believed to be the fastest quantum algorithm to invert the vast majority of *specific* efficiently computable functions (although obviously there are also many exceptions, i.e., functions that are easier to invert). Hash-based cryptography can convert any hard-to-invert function into a secure public-key signature system.

See the "Hash-based digital signature schemes" chapter of this book for a much more detailed discussion of hash-based cryptography. Note that most hash-based systems impose an extra requirement of collision resistance upon the hash function, allowing simpler signatures without randomization.

2.2 A code-based public-key encryption system

Assume that b is a power of 2. Write $n = 4b \lg b$; $d = \lceil \lg n \rceil$; and $t = \lfloor 0.5n/d \rfloor$. For example, if $b = 128$, then $n = 3584$; $d = 12$; and $t = 149$.

The receiver's public key in this system is a $dt \times n$ matrix K with coefficients in $\mathbf{F}_2$. Messages suitable for encryption are n-bit strings of "weight t," i.e., n-bit strings having exactly t bits set to 1. To encrypt a message m, the sender simply multiplies K by m, producing a dt-bit ciphertext Km.

The basic problem for the attacker is to "syndrome-decode K," i.e., to undo the multiplication by K, knowing that the input had weight t. It is easy, by linear algebra, to work backwards from Km to *some* n-bit vector v such that $Kv = Km$; however, there are a huge number of choices for v, and finding a weight-t choice seems to be extremely difficult. The best known attacks on this problem take time exponential in b for most matrices K.

How, then, can the receiver solve the same problem? The answer is that the receiver generates the public key K with a secret structure, specifically a "hidden Goppa code" structure, that allows the receiver to decode in a reasonable amount of time. It is conceivable that the attacker can detect the "hidden Goppa code" structure in the public key, but no such attack is known.

Specifically, the receiver starts with distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_n$ of the field $\mathbf{F}_{2^d}$ and a secret monic degree-t irreducible polynomial $g \in \mathbf{F}_{2^d}[x]$. The main work for the receiver is to syndrome-decode the $dt \times n$ matrix

$$H = \begin{pmatrix} 1/g(\alpha_1) & \cdots & 1/g(\alpha_n) \\ \alpha_1/g(\alpha_1) & \cdots & \alpha_n/g(\alpha_n) \\ \vdots & \ddots & \vdots \\ \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_n^{t-1}/g(\alpha_n) \end{pmatrix},$$

where each element of $\mathbf{F}_{2^d}$ is viewed as a column of d elements of $\mathbf{F}_2$ in a standard basis of $\mathbf{F}_{2^d}$. This matrix H is a "parity-check matrix for an irreducible binary Goppa code," and can be syndrome-decoded by "Patterson's algorithm" or by faster algorithms.

The receiver's public key K is a scrambled version of H. Specifically, the receiver's secret key also includes an invertible $dt \times dt$ matrix S and an $n \times n$ permutation matrix P. The public key K is the product SHP. Given a ciphertext $Km = SHPm$, the receiver multiplies by S^{-1} to obtain HPm, decodes H to obtain Pm, and multiplies by P^{-1} to obtain m.

What I've described here is a variant, due to Niederreiter (1986), of McEliece's original code-based public-key encryption system. Both systems are extremely efficient at key generation, encryption, and decryption, but—as I mentioned earlier—have been held back by their long public keys.

See the "Code-based cryptography" and "Lattice-based cryptography" chapters of this book for much more information about code-based cryptography and (similar but more complicated) lattice-based cryptography, including several systems that use shorter public keys.

2.3 A multivariate-quadratic public-key signature system

The public key in this system is a sequence $P_1, P_2, \ldots, P_{2b} \in \mathbf{F}_2[w_1, \ldots, w_{4b}]$: a sequence of $2b$ polynomials in the $4b$ variables $w_1, \ldots, w_{4b}$, with coefficients in $\mathbf{F}_2 = \{0, 1\}$. Each polynomial is required to have degree at most 2, with no squared terms, and is represented as a sequence of $1 + 4b + 4b(4b-1)/2$ bits, namely the coefficients of $1, w_1, \ldots, w_{4b}, w_1w_2, w_1w_3, \ldots, w_{4b-1}w_{4b}$. Overall the public key has $16b^3 + 4b^2 + 2b$ bits; e.g., 4 megabytes for $b = 128$.

A signature of a message m has just $6b$ bits: namely, $4b$ values $w_1, \ldots, w_{4b} \in \mathbf{F}_2$ and a $2b$-bit string r satisfying

$$H(r, m) = (P_1(w_1, \ldots, w_{4b}), \ldots, P_{2b}(w_1, \ldots, w_{4b})).$$

Here H is a standard hash function. Verifying a signature uses one evaluation of H and roughly b^3 bit operations to evaluate $P_1, \ldots, P_{2b}$.

The critical advantage of this signature system over hash-based signature systems is that each signature is short. Other multivariate-quadratic systems have even shorter signatures and, in many cases, much shorter public keys.

The basic problem faced by an attacker is to find a sequence of $4b$ bits $w_1, \ldots, w_{4b}$ producing $2b$ specified output bits

$$(P_1(w_1, \ldots, w_{4b}), \ldots, P_{2b}(w_1, \ldots, w_{4b})).$$

Guessing a sequence of $4b$ bits is fast but has, on average, chance only 2^{-2b} of success. More advanced equation-solving attacks, such as "XL," can succeed in considerably fewer than 2^{2b} operations, but no known attacks have a reasonable chance of succeeding in 2^b operations for most quadratic polynomials $P_1, \ldots, P_{2b}$ in $4b$ variables. The difficulty of this problem is not surprising, given how general the problem is: *every* inversion problem can be rephrased as a problem of solving multivariate quadratic equations.

How, then, can the signer solve the same problem? The answer, as in Section 2.2, is that the signer generates the public key $P_1, \ldots, P_{2b}$ with a secret structure, specifically an "HFE$^{\text{v}-}$" structure, that allows the signer to solve the equations in a reasonable amount of time. It is conceivable that the attacker can detect the HFE$^{\text{v}-}$ structure in the public key, or in the public key together with a series of legitimate signatures; but no such attack is known.

Fix a standard irreducible polynomial $\varphi \in \mathbf{F}_2[t]$ of degree $3b$. Define L as the field $\mathbf{F}_2[t]/\varphi$ of size 2^{3b}. The critical step in signing is finding roots of a secret low-degree *univariate* polynomial over L: specifically, a polynomial in $L[x]$ of degree at most $2b$. There are several standard algorithms that do this in time $b^{O(1)}$.

The secret polynomial is chosen to have all nonzero exponents of the form $2^i + 2^j$ or 2^i. If an element $x \in L$ is expressed in the form $x_0 + x_1 t + \cdots + x_{3b-1} t^{3b-1}$, with each $x_i \in \mathbf{F}_2$, then $x^2 = x_0 + x_1 t^2 + \cdots + x_{3b-1} t^{6b-2}$ and $x^4 = x_0 + x_1 t^4 + \cdots + x_{3b-1} t^{12b-4}$ and so on, so $x^{2^i+2^j}$ is a quadratic polynomial in the variables $x_0, \ldots, x_{3b-1}$. Some easy extra transformations hide the structure of this polynomial, producing the signer's public key.

Specifically, the signer's secret key has three components:

- An invertible $4b \times 4b$ matrix S with coefficients in $\mathbf{F}_2$.
- A polynomial $Q \in L[x, v_1, v_2, \ldots, v_b]$ where each term has one of the following six forms: $\ell x^{2^i+2^j}$ with $\ell \in L$, $2^i < 2^j$, $2^i + 2^j \le 2b$; $\ell x^{2^i} v_j$ with $\ell \in L$, $2^i \le 2b$; $\ell v_i v_j$; ℓx^{2^i}; ℓv_j; ℓ. If $b = 128$ then there are 9446 possible terms, each having a 384-bit coefficient ℓ, for a total of 443 kilobytes.
- A $2b \times 3b$ matrix T of rank $2b$ with coefficients in $\mathbf{F}_2$.

The signer computes the public key as follows. Compute a column vector $(x_0, x_1, \ldots, x_{3b-1}, v_1, v_2, \ldots, v_b)$ as S times the column vector $(w_1, \ldots, w_{4b})$. Inside the quotient ring $L[w_1, \ldots, w_{4b}]/(w_1^2 - w_1, \ldots, w_{4b}^2 - w_{4b})$, compute $x = \sum x_i t^i$ and $y = Q(x, v_1, v_2, \ldots, v_b)$. Write y as $y_0 + y_1 t + \cdots + y_{3b-1} t^{3b-1}$ with each y_i in $\mathbf{F}_2[w_1, \ldots, w_{4b}]$, and compute $(P_1, P_2, \ldots, P_{2b})$ as T times the column vector $(y_0, y_1, \ldots, y_{3b-1})$.

Signing works backwards through the same construction:

- Starting from the desired values of $P_1, P_2, \ldots, P_{2b}$, solve the secret linear equations $T(y_0, y_1, \ldots, y_{3b-1}) = (P_1, P_2, \ldots, P_{2b})$ to obtain values of $(y_0, y_1, \ldots, y_{3b-1})$. There are 2^b possibilities for $(y_0, y_1, \ldots, y_{3b-1})$; choose one of those possibilities randomly.
- Choose values $v_1, v_2, \ldots, v_b \in \mathbf{F}_2$ randomly, and substitute these values into the secret polynomial $Q(x, v_1, v_2, \ldots, v_b)$, obtaining a polynomial $Q(x) \in L[x]$.
- Compute $y = y_0 + y_1 t + \cdots + y_{3b-1} t^{3b-1} \in L$, and solve $Q(x) = y$, obtaining $x \in L$. If there are several roots x of $Q(x) = y$, choose one of them randomly. If there are no roots, restart the signing process.
- Write x as $x_0 + x_1 t + \cdots + x_{3b-1} t^{3b-1}$ with $x_0, \ldots, x_{3b-1} \in \mathbf{F}_2$. Solve the secret linear equations $S(w_1, \ldots, w_{4b}) = (x_0, \ldots, x_{3b-1}, v_1, \ldots, v_b)$, obtaining a signature $(w_1, \ldots, w_{4b})$.

This is an example of a class of HFE$^{\mathrm{v}-}$ constructions introduced by Patarin in 1996. "HFE" refers to the "Hidden Field Equation" $Q(x) = y$. The "$-$" refers to the omission of some bits: $Q(x) = y$ is equivalent to $3b$ equations on bits, but only $2b$ equations are published. The "v" refers to the "vinegar" variables $v_1, v_2, \ldots, v_b$. Pure HFE, with no omitted bits and no vinegar variables, is breakable in time roughly $2^{(\lg b)^2}$ by Gröbner-basis attacks, but HFE$^{\mathrm{v}-}$ has solidly resisted attack for more than ten years.

There are many other ways to build multivariate-quadratic public-key systems, and many interesting ideas for saving time and space, producing a huge number of candidates for post-quantum cryptography; see the "Multivariate public key cryptography" chapter of this book. It is hardly a surprise that some of the fastest candidates have been broken. A recent paper by Dubois, Fouque, Shamir, and Stern, after breaking an extremely simplified system with no vinegar variables and with only *one* nonzero term in Q, leaps to the conclusion that all multivariate-quadratic systems are dangerous:

> Multivariate cryptographic schemes are very efficient but have a lot of exploitable mathematical structure. Their security is not fully understood, and new attacks against them are found on a regular basis. It would thus be prudent not to use them in any security-critical applications.

Presumably the same authors would recommend already avoiding 4096-bit RSA in a pre-quantum world since 512-bit RSA has been broken, would recommend avoiding all elliptic curves since a few special elliptic curves have been broken (clearly elliptic curves have "a lot of exploitable mathematical structure"), and would recommend avoiding 256-bit AES since DES has been broken ("new attacks against ciphers are found on a regular basis").

My own recommendation is that the community continue to systematically study the security and efficiency of cryptographic systems, so that we can identify the highest-security systems that fit the speed and space requirements imposed by cryptographic users.

3 Challenges in post-quantum cryptography

Let me review the picture so far. Some cryptographic systems, such as RSA with a four-thousand-bit key, are believed to resist attacks by large classical computers but do not resist attacks by large quantum computers. Some alternatives, such as McEliece encryption with a four-million-bit key, are believed to resist attacks by large classical computers *and* attacks by large quantum computers.

So why do we need to worry *now* about the threat of quantum computers? Why not continue to focus on RSA and ECDSA? If someone announces the successful construction of a large quantum computer fifteen years from now, why not simply switch to McEliece etc. fifteen years from now?

This section gives three answers—three important reasons that parts of the cryptographic community are already starting to focus attention on post-quantum cryptography:

- We need time to improve the efficiency of post-quantum cryptography.
- We need time to build confidence in post-quantum cryptography.
- We need time to improve the usability of post-quantum cryptography.

In short, we are not yet prepared for the world to switch to post-quantum cryptography.

Maybe this preparation is unnecessary. Maybe we won't actually need post-quantum cryptography. Maybe nobody will ever announce the successful construction of a large quantum computer. However, if we don't do anything, and if it suddenly turns out years from now that users *do* need post-quantum cryptography, years of critical research time will have been lost.

3.1 Efficiency

Elliptic-curve signature systems with $O(b)$-bit signatures and $O(b)$-bit keys appear to provide b bits of security against classical computers. State-of-the-art signing algorithms and verification algorithms take time $b^{2+o(1)}$.

Can post-quantum public-key signature systems achieve similar levels of performance? My two examples of signature systems certainly don't qualify: one example has signatures of length $b^{2+o(1)}$, and the other example has keys of length $b^{3+o(1)}$. There are many other proposals for post-quantum signature systems, but I have never seen a proposal combining $O(b)$-bit signatures, $O(b)$-bit keys, polynomial-time signing, and polynomial-time verification.

Inefficient cryptography is an option for *some* users but is not an option for a busy Internet server handling tens of thousands of clients each second. If you make a secure web connection today to `https://www.google.com`, Google redirects your browser to `http://www.google.com`, deliberately turning off cryptographic protection. Google does have some cryptographically protected web pages but apparently cannot afford to protect its most heavily used web pages. If Google already has trouble with the slowness of today's cryptographic

software, surely it will not have *less* trouble with the slowness of post-quantum cryptographic software.

Constraints on space and time have always posed critical research challenges to cryptographers and will continue to pose critical research challenges to post-quantum cryptographers. On the bright side, research in cryptography has produced many impressive speedups, and one can reasonably hope that increased research efforts in post-quantum cryptography will continue to produce impressive speedups. There has already been progress in several directions; for details, read the rest of this book!

3.2 Confidence

Merkle's hash-tree public-key signature system and McEliece's hidden-Goppa-code public-key encryption system were both proposed thirty years ago and remain essentially unscathed despite extensive cryptanalytic efforts.

Many other candidates for hash-based cryptography and code-based cryptography are much newer; multivariate-quadratic cryptography and lattice-based cryptography provide an even wider variety of new candidates for post-quantum cryptography. Some specific proposals have been broken. Perhaps a new system will be broken as soon as a cryptanalyst takes the time to look at the system.

One could insist on using classic systems that have survived many years of review. But often the user cannot afford the classic systems and is forced to consider newer, smaller, faster systems that take advantage of more recent research into cryptographic efficiency.

To build confidence in these systems the community needs to make sure that cryptanalysts have taken time to search for attacks on the systems. Those cryptanalysts, in turn, need to gain familiarity with post-quantum cryptography and experience with post-quantum cryptanalysis.

3.3 Usability

The RSA public-key cryptosystem started as nothing more than a trapdoor one-way function, "cube modulo n." (Tangential historical note: The original paper by Rivest, Shamir, and Adleman actually used large random exponents. Rabin pointed out that small exponents such as 3 are hundreds of times faster.)

Unfortunately, one cannot simply use a trapdoor one-way function as if it were a secure encryption function. Modern RSA encryption does not simply cube a message modulo n; it has to first randomize and pad the message. Furthermore, to handle long messages, it encrypts a short random string instead of the message, and uses that random string as a key for a symmetric cipher to encrypt and authenticate the original message. This infrastructure around RSA took many years to develop, with many disasters along the way, such as the "PKCS#1 v1.5" padding standard broken by Bleichenbacher in 1998.

Furthermore, even if a secure encryption function has been defined and standardized, it needs software implementations—and perhaps also hardware implementations—suitable for integration into a wide variety of applications. Implementors need to be careful not only to achieve correctness and speed but also to avoid timing leaks and other side-channel leaks. A few years ago several implementations of RSA and AES were broken by cache-timing attacks; Intel has, as a partial solution, added AES instructions to its future CPUs.

This book describes randomization and padding techniques for some post-quantum systems, but much more work remains to be done. Post-quantum cryptography, like the rest of cryptography, needs complete hybrid systems and detailed standards and high-speed leak-resistant implementations.

4 Comparison to quantum cryptography

"Quantum cryptography," also called "quantum key distribution," expands a short shared key into an effectively infinite shared stream. The prerequisite for quantum cryptography is that the users, say Alice and Bob, both know (e.g.) 256 unpredictable secret key bits. The result of quantum cryptography is that Alice and Bob both know a stream of (e.g.) 10^{12} unpredictable secret bits that can be used to encrypt messages. The length of the output stream increases linearly with the amount of time that Alice and Bob spend on quantum cryptography.

This description of quantum cryptography might make "quantum cryptography" sound like a synonym for "stream cipher." The prerequisite for a stream cipher—for example, counter-mode AES—is that Alice and Bob both know (e.g.) 256 unpredictable secret key bits. The result of a stream cipher is that Alice and Bob both know a stream of (e.g.) 10^{12} unpredictable secret bits that can be used to encrypt messages. The length of the output stream increases linearly with the amount of time that Alice and Bob spend on the stream cipher.

However, the details of quantum cryptography are quite different from the details of a stream cipher:

- A stream cipher generates the output stream as a mathematical function of the input key. Quantum cryptography uses physical techniques for Alice to continuously generate random secret bits and to encode those bits for transmission to Bob.
- A stream cipher can be used to protect information sent through any number of untrusted hops on any existing network; eavesdropping fails because the encrypted information is incomprehensible. Quantum cryptography requires a direct fiber-optic connection between Alice's trusted quantum-cryptography hardware and Bob's trusted quantum-cryptography hardware; eavesdropping fails because it interrupts the communication.
- Even if a stream cipher is implemented perfectly, its security is merely conjectural—"nobody has figured out an attack so we conjecture that no

attack exists." If quantum cryptography is implemented perfectly then its security follows from generally accepted laws of quantum mechanics.

- A modern stream cipher can run on any commonly available CPU, and generates gigabytes of stream per second on a $200 CPU. Quantum cryptography generates kilobytes of stream per second on special hardware costing $50000.

One can reasonably argue that quantum cryptography, "locked-briefcase cryptography," "meet-privately-in-a-sealed-vault cryptography," and other physical shields for information are part of post-quantum cryptography: they will not be destroyed by quantum computers! But post-quantum cryptography is, in general, a quite different topic from quantum cryptography:

- Post-quantum cryptography, like the rest of cryptography, covers a wide range of secure-communication tasks, ranging from secret-key operations, public-key signatures, and public-key encryption to high-level operations such as secure electronic voting. Quantum cryptography handles only one task, namely expanding a short shared secret into a long shared secret.
- Post-quantum cryptography, like the rest of cryptography, includes some systems *proven* to be secure, but also includes many lower-cost systems that are *conjectured* to be secure. Quantum cryptography rejects conjectural systems—begging the question of how Alice and Bob can securely share a secret in the first place.
- Post-quantum cryptography includes many systems that can be used for a noticeable fraction of today's Internet communication—Alice and Bob need to perform some computation and send some data but do not need any new hardware. Quantum cryptography requires new network hardware that is, at least for the moment, impossibly expensive for the vast majority of Internet users.

My own interests are in cryptographic techniques that can be widely deployed across the Internet; I see tremendous potential in post-quantum cryptography and very little hope for quantum cryptography.

To be fair I should report the views of the proponents of quantum cryptography. Magiq, a company that sells quantum-cryptography hardware, has the following statement on its web site:

> Once the enormous energy boost that quantum computers are expected to provide hits the street, most encryption security standards—and any other standard based on computational difficulty—will fall, experts believe.

Evidently these unnamed "experts" believe—and Magiq would like you to believe—that quantum computers will break AES, and dozens of other well-known secret-key ciphers, and Merkle's hash-tree signature system, and McEliece's hidden-Goppa-code encryption system, and Patarin's HFE^{v-} signature system, and NTRU, and all of the other cryptographic systems discussed in this book. Time will tell whether this belief was justified!

Quantum computing

Sean Hallgren[1] and Ulrich Vollmer[2]

[1] The Pennsylvania State University.
[2] Berlin, Germany.

In this chapter we will explain how quantum algorithms work and how they can be used to attack crypto systems. We will outline the current state of the art of quantum algorithmic techniques that are, or might become relevant for cryptanalysis. And give an outlook onto possible future developments.

1 Classical cryptography and quantum computing

Quantum computation challenges the dividing line for tractable versus intractable problems for computation. The most significant examples for this are efficient quantum algorithms for breaking cryptosystems which are believed to be secure for classical computers. In 1994 Shor found quantum algorithms for factoring and discrete log, and these can be used to break the widely used RSA cryptosystem and Diffie-Hellman key-exchange using a quantum computer. The most obvious question this raises is what cryptosystems to use after quantum computers are built. Once a good replacement system is found there will still issues with the logistics of changing every cryptosystem in use, and it will take time to do so. Furthermore, the most sensitive of today's encrypted information should stay secure even after quantum computers are built. This data must therefore already be encrypted with quantum resistant cryptosystems.

Classical cryptography [12, 13] consists of problems and tools including encryption, key distribution, digital signatures, pseudo-random number generation, zero-knowledge proofs, and one-way functions. There are many applications such as signing contracts, electronic voting, and secure encryption. It turns out that these systems can only exist if there is some kind of computational difficulty which can be used to build these systems. For example, RSA is secure only if factoring is computationally hard for classical computers to solve. However, complexity theory does not provide the tools to prove that an efficient algorithm does not exist for a problem. Instead, decisions about which problems are difficult to solve are based entirely on empirical

evidence. Namely, if researchers have tried over a long period of time and the problem still seems difficult, then at least it appears difficult to find an algorithm. In order to understand which problems are difficult for quantum computers, we must conduct a long-term extensive study of the problems by many researchers.

Designing cryptographic schemes is a difficult task. The goal is to have schemes which meet security requirements no matter which way an adversary may use the system. Modern cryptography has focused on building a sound foundation to achieve this goal. In particular, the only assumption made about an adversary is its computational ability. Typically one assumes the adversary has a classical computer, and is restricted to randomized polynomial time. But if one now assumes that the adversary has a quantum computer, then which classical cryptosystems are secure, and which are not? Quantum computation uses rules which are new and unintuitive. Some subroutines, such as computing the quantum Fourier transform, can be performed exponentially faster than by classical computers. However, this is not for free. The methods to input and output the data from the Fourier transform are very restricted. Hence, finding quantum algorithms relies on walking a fine line between using extra power while being limited in some important ways. How do we design new classical cryptosystems that will remain secure even in the presence of quantum computers? Such systems would be of great importance since they could be implemented now, but will remain secure when quantum computers are built. Table 1 shows the current status of several cryptosystems.

Cryptosystem	**Broken by Quantum Algorithms?**
RSA public key encryption	Broken
Diffie-Hellman key-exchange	Broken
Elliptic curve cryptography	Broken
Buchmann-Williams key-exchange	Broken
Algebraically Homomorphic	Broken
McEliece public key encryption	Not broken yet
NTRU public key encryption	Not broken yet
Lattice-based public key encryption	Not broken yet

Table 1. Current status of security of classical cryptosystems in relation to quantum computers.

Given that the cryptosystems currently in use can be broken by quantum computers, what would it take for people to switch to new cryptosystems safe in a quantum world, and why hasn't it happened yet? First of all, the replacement systems must be efficient. There are alternative cryptosystems such as lattice-based systems or the McEliece system, but they are currently

too inefficient to use in practice. The second requirement is that there should be good evidence that a new system cannot be broken by a quantum computer, even after another decade or two of research has been done. Systems will only satisfy this after extensive research is done on them. To complicate matters, some of these systems are still being developed. In order to make them more competitive with the efficiency of RSA, special cases or new variants of the systems are being proposed. However, the special properties these systems have that make them more efficient may also make them more vulnerable to classical or quantum attacks.

In the remainder of this section we will give some more background on systems which have been broken. In Section 4 the basic framework behind the quantum algorithms that break them will be given.

1.1 Cryptosystems vulnerable to quantum computers

Public key cryptography, a central concept in cryptography, is used to protect web transactions, and its security relies on the hardness of certain number theoretic problems. As it turns out, number theoretic problems are also the main place where quantum computers have been shown to have exponential speedups. Examples of such problems include factoring and discrete log [38], Pell's equation [18], and computing the unit group and class group of a number field [17, 37]. The existence of these algorithms implies that a quantum computer could break RSA, Diffie-Hellman and elliptic curve cryptography, which are currently used, as well as potentially more secure systems such as the Buchmann-Williams key-exchange protocol [6]. Understanding which cryptosystems are secure against quantum computers is one of the fundamental questions in the field.

As an example, factoring is a long-studied problem and several exponential time algorithms for it are known including Lehman's method, Pollard's ρ method, and Shanks's class group method [7]. It became practically important with the invention of the RSA public-key cryptosystem in the late 1970s, and it started receiving much more attention. The security of RSA depends on the assumption that factoring does not have an efficient algorithm. Subexponential-time algorithms for it were later found [31, 34] using a continued fraction algorithm, a quadratic sieve, and elliptic curves. The number field sieve [26, 27], found in 1989, is the best known classical algorithm for factoring and runs in time $\exp(c(\log n)^{1/3}(\log\log n)^{2/3})$ for some constant c. In 1994, Shor found an efficient quantum algorithm for factoring.

Finding exponential speedups via quantum algorithms has been a surprisingly difficult task. The next problem solved after Shor's algorithms was eight years later, when a quantum algorithm for Pell's equation [18] was found. Given a positive non-square integer d, Pell's equation is $x^2 - dy^2 = 1$, and the goal is to compute a pair of integers (x, y) satisfying the equation. The first (classical) algorithm for Pell's equation dates back to 1000 a.d. – only Euclid's algorithm is older. Solving Pell's equation is at least as hard as factoring,

and the best known classical algorithm for it is exponentially slower than the best known factoring algorithm. In an effort to make this computational difficulty useful Buchmann and Williams devised a key-exchange protocol whose hardness is based on Pell's equation [6]. Their goal was to create a system that is secure even if factoring turns out to be polynomial-time solvable. The quantum algorithm breaks the Buchmann-Williams system using a quantum computer. Also broken are certain zero-knowledge protocols because they rely on the computational hardness of solving Pell's equation [5].

Most research in quantum algorithms has revolved around the hidden subgroup problem (HSP), which will be defined in Section 4. The HSP is a problem defined on a group, and many problems reduce to it. Factoring and discrete log reduce to the HSP when the underlying group is finite or countable. Pell's equation reduces to the HSP when the group is uncountable. For these cases there are efficient quantum algorithms to solve the HSP, and hence the underlying problem, because the group is abelian. Graph isomorphism reduces to the HSP for the symmetric group, and the unique shortest lattice vector problem is related to the HSP when the group is dihedral. These two groups are nonabelian, and much research over the last decade has focused on trying to generalize the success of the abelian HSP to the nonabelian HSP case. There are reasons to hope that the techniques which use Fourier analysis, may work. Some progress has been made on some cases [3, 10, 23]. However, much of what has been learned so far has been about the limitations of quantum computers for the HSP over nonabelian groups [20].

There have been exponential speedups for a few oracle problems which are not instances of the HSP. One example is the shifted Legendre symbol problem [40], where the quantum algorithm is able to pick out the amount that a function is cyclically rotated. This algorithm is able to break certain algebraically homomorphic encryption systems. There are also speedups for some problems from topology [1].

Finding exponential speedups remains a fundamental, important, and difficult problem. NP-Complete problems are not believed to have efficient quantum algorithms [4]. The problem of finding hard problems on which to base cryptosystems is similar: it is not believed possible to base cryptosystems on NP-Complete problems. In this sense, finding exponential speedups and breaking classical cryptosystems seem related. Furthermore, understanding which classical cryptosystems are secure against quantum attacks is a relevant and important question. The most sensitive data which is encrypted today should remain protected even if quantum computers are built in ten years, and believing that a cryptosystem is secure happens only after a very long and extensive study.

1.2 Other cryptographic primitives

Pseudo-random number generation is one of the basic tools of cryptography. A short string is stretched into a long string, and the next bit in the sequence

must be unpredictable by any polynomial-time machine. If this is the case then the sequence is as good as uniform, since the machine cannot detect a difference. Since this definition is based on the computational power of the machine, primitives must be reexamined for quantum computation.

Another central concept in cryptography is the zero-knowledge protocol. These protocols allow a prover to convince a verifier that it knows a secret without the verifier learning any information about the secret. In practice this is used to allow one party to prove its identity to another by proving it has a particular secret. For a protocol to be zero-knowledge, no information can be revealed no matter what strategy a so-called *cheating verifier* follows when interacting with the prover. Therefore, an important question is: what happens to these classical protocols when the cheating verifier is a quantum computer?

Watrous [41] showed that two well-known classical protocols are zero-knowledge against quantum computers. This was difficult due to the nature of quantum states and the technical definition of zero-knowledge. Watrous showed that the Goldreich-Micali-Wigderson [11] graph isomorphism protocol is secure, and also that the graph 3-coloring protocol in [11] is secure if one can find classical commitment schemes that are concealing against quantum computers.

These results were recently extended to SZK, extending Watrous's result to protocols with honest-verifier proofs [19]. The class SZK has received much attention in recent years [8, 15, 16, 32, 36, 39, 41]. From a complexity-theoretic perspective SZK is very interesting. It contains many important problems such as quadratic residuosity and non-residuosity, graph isomorphism and non-isomorphism, as well as problems related to discrete logarithm and the shortest and closest vector problems in lattices. These problems have the unique property that they are not believed to be NP-hard, and yet no efficient algorithm for them is known. These problems are also the natural candidates for constructing public-key cryptosystems, and incidentally, they are also the problems where one hopes to find an exponential speedup by a quantum algorithm.

2 The computational model

Classical computing devices are at any given point in time in a state that can be described by a single string of bits. This bit string represents the "data" the machine operates on and the "program", a sequence of directives for the processing of the data by the device. The distinction between the two while seemingly clear for the computer on our desktop is indeed somewhat artificial.

In a quantum machine the distinction is succinct. The program is again a sequence of "gates" from a well defined finite set which is independent from the input to the algorithm or derived from it by a classical algorithm. It is the data where quantum parallelism sets in: At each given time, the quantum

device is in a "superposition" of states each of which can be represented by a string of bits. The quantum part of the algorithm transforms all these states at once.

The most simple model describing the physical state of a quantum machine is finite dimensional Hilbert space. Abstracting from circumstantial aspects of the machine, what we are interested in is its heart, the "registers" storing the data. Quantum memory storing one quantum bit, or qubit as we will call it in all that follows, will have to allow for a superposition of the two states 0 and 1. Hence it is two-dimensional and can be modeled by the canonical two-dimensional Hilbert space

$$\mathcal{H} = \mathcal{H}_1 = \mathbb{C} \oplus \mathbb{C} \ .$$

We will use the set consisting of $(1,0)$ and $(0,1)$ as the standard (computational) basis for $\mathcal{H}$, and denote these vectors by $|0\rangle$, and $|1\rangle$, respectively.

Wider, n-bit registers need to be 2^n-dimensional and are, consequentially, modeled by

$$\mathcal{H}_n = \mathcal{H} \otimes \cdots \otimes \mathcal{H} \ .$$

We use the computational basis for $\mathcal{H}$ to construct one for $\mathcal{H}_n$. Define for bits $i_1, \ldots, i_n$ the vector

$$|i_1 \cdots i_n\rangle = |i_1\rangle \otimes \cdots \otimes |i_n\rangle.$$

These vectors with $i_1, \ldots, i_n$ running through the set $\mathcal{I}_n$ of all n-tuples of bits form a basis for $\mathcal{H}_n$.

Once the quantum device has performed its computations we need a way to transform its complex state back into a series of bits which will represent the classical output of the algorithm employed. This process is called "measurement" and is non-deterministic in nature.

Given the final state of the quantum machine is

$$v = \sum_{I \in \mathcal{I}_n} \alpha_I |I\rangle \ ,$$

measurement yields bit strings according to a probability distribution P_v which depends on v: For all $I \in \mathcal{I}_n$ the probability that I is obtained in the measurement is

$$P_v(I) = |\alpha_I|^2 / \sum_{J \in \mathcal{I}_n} |\alpha_J|^2 \ .$$

This implies that our quantum algorithms should yield final quantum states whose "amplitude" α_I at a desired output I is large in absolute value relative to the amplitudes at the other base vectors. Unless we succeed in reducing the amplitudes at non-desired base vectors to 0, we will need to be able to check the result of a quantum algorithm or live with some limited uncertainty about its correctness. Cryptanalytically, this is not a problem since we can regularly tell when an attack that uses the output of our computation was successful or not.

Back from data space to programs for quantum machines: Quantum systems evolve reversibly by unitary transitions. Thus the gates our quantum machines will put the data through need to be given as unitary operators on the state space $\mathcal{H}_n$. Depending on its physical realization, a quantum machine will be able to perform a small set of such unitary transformations. More complex transformations will need to be built out of this finite set.

The basic building blocks of our quantum algorithms will be operators on $\mathcal{H}_1$ and $\mathcal{H}_2$ which will be extended to $\mathcal{H}_n$ by tensoring with the trivial operator Id. Given an operator H on $\mathcal{H}_2$, we may extend it to $\mathcal{H}_n$ by defining

$$\tilde{H} : \mathcal{H}_n \to \mathcal{H}_n : v_1 \otimes v_2 \otimes v_3 \otimes \cdots \otimes v_n \longmapsto H(v_1 \otimes v_2) \otimes v_3 \otimes \cdots \otimes v_n .$$

Of course, H may operate on any two consecutive positions (qubits), not just positions 1 and 2.

Thus a program for a quantum machine is a sequence of gates from a fixed finite set $\mathcal{G}$. This sequence is computed by a (uniform) classical algorithm starting from the input. It is also called a *quantum circuit*.

The set $\mathcal{G}$ depends on the physical features of the quantum machine we model: each *gate* in the set $\mathcal{G}$ describes a manipulation of the quantum machine state we are able to perform. This correspondence is approximative, and requires fault-tolerant techniques to contain the slight errors introduced at each step.

For our purposes it is enough to know that $\mathcal{G}$ is chosen in such a way that any unitary operator can be approximated by a sequence of operators in $\mathcal{G}$. These approximations may be difficult to compute, however. Furthermore, we require that $\mathcal{G}$ contain with every operator also its inverse.

An example of such a gate set contains

$$U = \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{pmatrix} , \; W = \frac{1}{\sqrt{2}} \begin{pmatrix} 1&1 \\ 1&-1 \end{pmatrix} , \; S = \begin{pmatrix} 1&0 \\ 0&i \end{pmatrix} , \; T = \begin{pmatrix} 1&0 \\ 0&e^{\pi i/4} \end{pmatrix}$$

(or rather all their extensions to $\mathcal{H}^{\otimes n}$ obtained through tensoring suitably with Id), and their inverses.[1]

We measure the distance between two unitary operators—and thus also the distance between an operator and a quantum circuit which approximates it—by the operator norm: Two operators H_1 and H_2 have distance ϵ if $H_1 - H_2$ maps the unit ball into a ball of radius ϵ. For this we write $\|H_1 - H_2\| < \epsilon$. The quality of approximation is additive under concatenation. For any unitary operators H_1 and H_2 we have

$$\|\tilde{H}_i - H_i\| < \epsilon_i \text{ for } i = 1, 2 \quad \Rightarrow \quad \|\tilde{H}_1\tilde{H}_2 - H_1H_2\| < \epsilon_1 + \epsilon_2 .$$

[1] It seems strange to include S in $\mathcal{G}$ when $S = T^2$. The reason for this is the need to implement T fault-tolerantly which we only know how to do with the aid of S.

Approximation of operators which work only on one qubit is easy and efficient. Suppose some operator H affects only one qubit. that means that there exists a unitary operator H' and some k with $1 \leq k \leq n$ such that

$$H(|i_1 \cdots i_{k-1}\rangle \otimes |i_k\rangle \otimes |i_{k+1} \cdots i_n\rangle) = |i_1 \cdots i_{k-1}\rangle \otimes H'|i_k\rangle \otimes |i_{k+1} \cdots i_n\rangle$$

for all base vectors $|I\rangle = |i_1 \cdots i_n\rangle$ with $I \in \mathcal{I}_n$. Then we can efficiently compute a sequence of gates in $\mathcal{G}$ which approximates H. The length of this sequence grows quadratically with $\log(1/\epsilon)$ where ϵ is the desired closeness of approximation. Thus, it is justified to treat $\mathcal{G}$ as if it contains all one qubit gates.

In order to execute classical algorithms operating on n bit memory on a quantum machine, it is necessary to embed them reversibly in a state space of dimension $n + k$ with some small $k > 0$. It is possible to do this for the universal classical gate NAND by using the Toffoli gate which is a doubly controlled negation, and one auxiliary bit, cf. Figure 1.

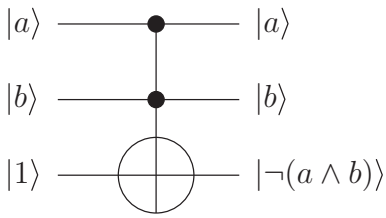

Fig. 1. Construction of the NAND gate from a doubly controlled negation—a so-called *Toffoli gate*—and one auxiliary bit

The Toffoli gate itself can be constructed as a word of length 16 in gates from the set $\mathcal{G}$ defined above. Moreover, we can emulate the drawing of random bits by using the state $W|0\rangle$ which yields when measured 0 or 1 each with the same probability.

In conclusion, we obtain for any classical algorithm which computes the boolean function f a quantum circuit U_f which maps $|I\rangle|0\rangle$ onto $|I\rangle|f(I)\rangle$ for all $I \in \mathcal{I}$. The length of U_f will be proportional to the length of the classical circuit computing f.

3 The quantum Fourier transform

The quantum Fourier transform (QFT) uses quantum parallelism for the fast computation of the discrete Fourier transform of functions on (boxes in) $\mathbb{Z}^n$. If we succeed in encoding some desired information into the period lattice of an efficiently computable function, then we may use QFT to extract this period lattice.

The typical application of the QFT is the solution of the hidden subgroup problem (HSP). In its simplest form, this problem asks given a periodic function on $\mathbb{Z}$ to find its period, i.e. to find the hidden subgroup $l\mathbb{Z}$ of $\mathbb{Z}$ of smallest index for which f is constant on the cosets $a + l\mathbb{Z}$.

This can be generalized to arbitrary groups as follows. Given a group $\mathcal{G}$, a set generating it, say $\mathcal{G} = \{g_1, \ldots, g_n\}$, and a function f on $\mathbb{Z}^n$ for which there is a normal subgroup H of G and an injective function g on G/H such that

$$f(x_1, \ldots, x_k) = g(\prod_{i=1}^{k} g_i^{x_i} \bmod H) \; .$$

The HSP then asks us to present a generating set of the largest such H and the relations between its elements.

If G is Abelian, it is possible to employ QFT to compute a generating set $\mathcal{L}$ for the period lattice

$$L = \left\{ (x_1, \ldots, x_n) \mid \prod_{i=1}^{n} g_i^{x_i} \in H \right\} \; .$$

Given $\mathcal{L}$, all that is left to do is to compute the Smith normal form of the matrix whose columns are the elements of $\mathcal{L}$. There is a classical algorithm for this computation which runs in time $O(n^3 l \log \|\mathcal{L}\|^2)$ where $l = \mathrm{card}\mathcal{L}$ and $\|\mathcal{L}\|$ denotes the maximum of all coordinates occurring in elements of $\mathcal{L}$.

In order to explain how QFT is used in the solution of the HSP, we will first define the QFT operator, and then show how to employ it in a larger algorithm.

We begin by defining QFT on an interval of length $N = 2^k$. For this purpose we identify the integer i with the base vector $|i\rangle$ in $\mathcal{H}_k$ according to the binary representation of i. The QFT operator is then defined by

$$\mathrm{QFT}_k : \mathcal{H}_k \to \mathcal{H}_k : |x\rangle \longmapsto 2^{-N/2} \sum_{y=0}^{N-1} e^{2\pi i x y / N} |y\rangle \; .$$

Proposition 1. *The operator QFT_k can be computed exactly in time $O(k^2)$. It can be approximated with* a priori *fixed given precision in time $O(k)$.*

A proof can be found in [33].

The QFT on $\mathbb{Z}^n$ is obtained a n-fold tensor product of one-dimensional QFT_k with itself.

For the solution of the HSP we prepare the following state using the circuit U_f derived from a circuit for the computation of the given function f.

$$|0\rangle|0\rangle \xrightarrow{W^{\otimes n}} \frac{1}{2^{N/2}} \sum_{x=0}^{N-1} |x\rangle|0\rangle \xrightarrow{U_f}$$

$$\frac{1}{2^{N/2}} \sum_{x=0}^{N-1} |x\rangle|f(x)\rangle = \frac{1}{2^{N/2}} \sum_{z \in f} \Big(\sum_{x|f(x)=z} |x\rangle|z\rangle \Big) . \quad (1)$$

The amplitudes of each of the summands on the right-hand side are given by the characteristic function of the period lattice of f (shifted by a constant vector).

The state we obtain after applying the QFT to (1) has amplitudes of large absolute value in those vectors $|y\rangle$ for which y seen as a point in space lies close to a point on the lattice which is dual to a scaled version of the period lattice of f. More precisely, y will lie close to a point on

$$L^* = \left\{ w \in \mathbb{Z}^k \mid Nw \cdot x \in \mathbb{Z} \text{ for all } x \in L \right\}$$

where L is the period lattice of f.

If we return to the one-dimensional case, this means that y is close to a integral multiple of N/l where l, we recall, is the generator of the sought lattice $l\mathbb{Z}$. Given several such multiples (in all likelihood two will suffice) we can extract the sought l.

There are some technical considerations to take into account in this process, one of which is the choice of a suitable N. (It should be large in comparison to a bound $\rho(L)$ on the length of all vectors in a short basis of L.) The qualitative picture, however, is as follows.

Proposition 2. *There is a probabilistic quantum algorithm with the following properties. Let $n \in \mathbb{N}$ and $L \subseteq \mathbb{Z}^n$. Suppose we are given a periodic function f for which U_f can be efficiently computed.*

Then the algorithm computes a basis of L with some constant success probability dependent only on n. It runs in time $O(T(f,N) + \log_2^3 N)$ where N is a power of 2 in $O(\rho(L)(\det L)^3)$ and $T(f,N)$ is the time required for the computation of f on arguments with coordinates in $0, \ldots, N-1$.

For a proof see [37].

Remark 1. The constants hidden in the O notation of the proposition seem to depend heavily (i.e. exponentially) on the dimension k. The same is true for the success probability. In all cryptanalytical applications, however, k is really small, say 2.

Remark 2. Moreover, you should note that the proposition gives an upper bound on the run-time. It is possible that the algorithm also succeeds if N is chosen substantially smaller than the bounds given in the proposition with corresponding effects on the run-time.

4 The hidden subgroup problem

The problems that can be solved efficiently on a quantum computer are best understood with reference to the framework of the hidden subgroup problem (HSP), which is a generalization of Shor's factoring and discrete log algorithms. The HSP is defined as: given a group and a function that is constant and distinct on cosets of some unknown subgroup, find a set of generators for the subgroup. The main tool used in algorithms is Fourier sampling, i.e. computing the Fourier transform and measuring, and its nice group theoretic properties lead to the solution of the HSP when the underlying group is finite and abelian. However, problems do not always fit directly into this group theoretic picture, and different methods are used to prove that the problem at hand still can be solved. For example, the extension to Pell's equation requires a solution to the HSP over groups that are not finitely generated. Another example is when a nonabelian case is reduced to the abelian case. Table 4 shows the current status of the abelian HSP.

Abelian Group G	**Associated Problem**	**Quantum Algorithm?**
$\mathbb{Z}_2^n$		Yes
The integers $\mathbb{Z}$	Factoring	Yes
Finite groups	Discrete Log	Yes
The reals $\mathbb{R}$	Pell's equation	Yes
The reals $\mathbb{R}^c$, c a constant	Unit group of number field	Yes
The reals $\mathbb{R}^n$, n arbitrary	Unit group, general case	Open

One of the main open questions in the area is to find an efficient quantum algorithm for the HSP when the underlying group is nonabelian. The main task in the nonabelian HSP is understanding the relationship between the nonabelian HSP and the representation theory of the underlying group. Unlike the abelian HSP, it is unknown how to solve this problem efficiently on a quantum computer. It was well known for many years that a solution of when G is the symmetric group would solve graph isomorphism, a long standing open problem in computer science, with many applications. For this reason, the nonabelian HSP has received much attention from researchers. However, even though Fourier sampling was well known to be sufficient to solve the abelian HSP, the same basic question of whether it was also sufficient to solve the nonabelian HSP has been more difficult to understand.

A positive and a negative answer to this question were given in [21]. There it was shown that the nonabelian HSP could be solved when the hidden subgroup is normal, if the Fourier transform over G is efficient, and if it is possible to compute the intersection of a set of representations. This is a direct generalization of the abelian HSP, since every subgroup of an abelian group is normal. It was also shown that restricted Fourier sampling is not enough to

Nonabelian Group G	**Associated Problem**	**Quantum Algorithm?**
Heisenberg group		Yes
$\mathbb{Z}_p^r \rtimes \mathbb{Z}_p$, r constant		Yes
$\mathbb{Z}_p^n \rtimes \mathbb{Z}_2$, p a fixed prime		Yes
Extraspecial groups		Yes
$\downarrow$?		
Dihedral group $D_n = \mathbb{Z}_n \rtimes \mathbb{Z}_2$	Unique shortest lattice vector	Subexponential-time
Symmetric group S_n	Graph isomorphism	Evidence of hardness

solve graph isomorphism, when attempting to use the well-known reduction of graph isomorphism to the nonabelian HSP.

It was shown in [28] that Fourier sampling a polynomial number of times cannot be used to solve graph isomorphism, and more generally, it does not suffice to use polynomially many quantum measurements. However, a simple information theoretic argument shows that if the algorithm instead uses quantum entanglement by performing one measurement across the polynomially many copies, then graph isomorphism can be solved. The problem is that it is unknown how to implement such large measurements efficiently. This left open the possibility that measurements across a small number of copies may suffice. But it was then shown that a joint measurement across all polynomially many copies is necessary, providing good evidence that this is indeed a hard problem [20]. The hardness of this problem was recently used in [30] to construct a classical one-way function which is believed to be secure against quantum computers. This is an example of a quantum inspired proposal for quantum resistant problems, and it provides a new promising candidate for one-way functions.

Another target for exponential speedups by quantum computation is the unique shortest lattice vector problem. Building cryptosystems based on them is the subject of Chapter 5 of this book. Given a set of n linearly independent vectors in $\mathbb{R}^n$, a lattice is defined as the set of integer linear combinations of these vectors. These vectors are called a basis of the lattice, and each lattice has an infinite number of different bases (when the dimension is greater than one).

The LLL algorithm can efficiently find vectors in a lattice whose lengths are within an exponential factor of the shortest vector [25], and this can be used to factor polynomials with rational coefficients. One open question is whether the problem of finding the shortest vector has an efficient solution when the lattice has the extra property that the shortest vector is much shorter than the rest of the non-parallel vectors. This problem is in NP$\cap$CoNP for the right parameter ranges, making it a good target for quantum algorithms. Cryptosystems proposed by Ajtai and Dwork [2], and also by Goldreich, Goldwasser, and Halevi [14], have been based on the hardness of this problem. Therefore the

problem is interesting from a complexity point of view, from a cryptographic point of view, and it is a long standing open question in theoretical computer science.

One of the main approaches to solving the shortest lattice vector problem is to use its connection to the HSP over the dihedral group as shown by Regev [35]. In this approach, so called *coset states* are created using the function. In the abelian case, Fourier sampling, i.e., computing the Fourier transform and measuring the result, is enough to solve the problem. The dihedral group is a nonabelian group which looks close to abelian by some measures and shares the property that one coset state has information about the subgroup, however it is unknown how to extract it efficiently. The best known quantum algorithm is a subexponential time sieve in [24]. Unfortunately, this algorithm provides no speedup over the best classical lattice algorithms.

4.1 The abelian HSP

Given an instance of the HSP on a finite group, the goal is to compute a set of generators for the hidden subgroup H in a number of steps that is polynomial in $\log |G|$. The *standard method* is the following sequence of steps, based on Simon's algorithm and Shor's algorithms:

Algorithm 4.1 The Standard Method for the HSP

Input: An HSP instance $f : G \to S$.
Output: Subgroup $H \subseteq G$.

1: Repeat the following polynomially many times:

   a. Evaluate f in superposition:

$$\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x, f(x)\rangle$$

   b. Measure the second register:

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |k + h, f(k)\rangle$$

   c. Compute the Fourier transform and measure.

2: Classically compute H from the measurement results in the first step.

Steps a–b create a random *coset state*, which is a uniform superposition over a random coset of H. If not for the coset representative k, it would be sufficient to measure, and get a random element of H. Instead, measurements must be used that will work despite the random coset representative produced in each iteration. Note the second register can be dropped from the notation since it is fixed, to give the state $|k + H\rangle$.

When the group is abelian the quantum Fourier transform takes a coset state to a state which is the Fourier transform of the subgroup state $|H\rangle$, with some coset dependent phases. These phases have norm one and do not change the resulting probability distribution. Therefore, the problem reduces to understanding the Fourier transform of a subgroup, and this is just a subgroup $\widehat{H}$ of the group of characters $\widehat{G}$ of G. Polynomially many samples gives a set of generators for $\widehat{H}$, and from these it is possible to efficiently classically compute a generating set for H.

Algorithms become more complicated when the underlying group is not finite or abelian. For factoring, the underlying group is the integers $\mathbb{Z}$ (or from another point of view, a finite group whose size is unknown). For Pell's equation the group is the reals $\mathbb{R}$. In these cases the standard method is used, but finite approximations must be used for the group G and for where the function is evaluated. For example, it is not possible to create a superposition over the original group elements. Using a finite group and a Fourier transform over a finite group, it must then be shown that the resulting distribution has enough information about the subgroup and that it can be computed efficiently. For arbitrary dimension n, the noise from using discrete approximations becomes very bad and this is one of the reasons the problem is still open.

4.2 The nonabelian HSP

For the nonabelian case, the underlying group determines whether the standard method provides enough information to be solved. Even when it does, the subgroup may still be difficult to compute from the samples.

It has been known for some time that polynomially many coset states have enough information to compute the subgroup [9], or to restrict to a simpler problem, just to determine if the subgroup is trivial or order two. That is, using Steps a–b on k registers, create the state

$$|g_1 H\rangle|g_2 H\rangle \otimes \cdots \otimes |g_k H\rangle,$$

where k is around log the group size. Then there is a joint quantum measurement across all k registers (instead of acting on each one independently) that determines whether the subgroup is trivial. Detecting trivial versus order two subgroups follows from a simple counting argument about the number of cosets and subgroups in the space for order two subgroups, versus the $|G|^k$ possible cosets of the trivial subgroup. The cosets of order two subgroups span an exponentially small fraction of the space as k grows, whereas the cosets of the trivial subgroup always span the whole space. This holds for any finite group.

As mentioned above, the main two cases with applications are the dihedral group and the symmetric group. For the dihedral group computing the Fourier transform of each register and measuring (i.e. using the standard approach) results in enough information to compute the subgroup, but the best known

algorithm for reconstructing H takes exponential time. For the symmetric group, it has been shown that no measurement on less than the full $n \log n$ set of registers will have sufficient information to compute the subgroup.

One area of research is determining what types of measurements on sets of coset states can be used to compute the subgroup. For the dihedral case, a sieve algorithm has been shown to take subexponential time to compute the subgroup. It works by starting with an exponential number of coset states and combining them two at a time to get a new one, and then repeating this process. The result is one coset state of a special form that allows the subgroup to be computed [24]. For the symmetric group much less is known. A sieve algorithm has been shown not to work [29].

Some progress has been made in some cases by reducing the nonabelian case to abelian case using classical and quantum techniques [22]. Semidirect products have also been a good source of groups to attack. In [10] it was shown how to solve the HSP over $\mathbb{Z}_p^n \rtimes \mathbb{Z}_2$ for constant prime p, and also over groups with smoothly solvable commutator subgroups. They use coset states but divert from the standard method. In [3] a different approach on coset states was used to understand the optimal measurement to extract information about the subgroup. There the HSP is solved for $\mathbb{Z}_p^r \rtimes \mathbb{Z}_p$ for a fixed r. One feature of this approach is that they show how to use entangled measurements across r coset states to compute the subgroup. Extraspecial groups have also been solved [23].

The nonabelian HSP remains an active research area. It represents both generalizations of most of the successes in quantum algorithms, and may also point to good quantum resistant problems if they are not solved.

5 Search algorithms

Given the value s of some boolean function f whose structure we cannot access, a search algorithm finds at least one pre-image. Classically this is only possible if we evaluate f a number of times which is proportional to the quotient between the cardinalities N and M of domain, and $f^{-1}(s)$, correspondingly. The ingenious quantum algorithm by Grover succeeds in lowering the classical complexity by a factor of $\sqrt{N/M}$.

The algorithm in its simplest form requires a priori knowledge of M. A slight modification allows for the determination of M in conjunction with the search.

The algorithm can also be employed to determine whether a given value lies in the image of f. This can be used to search for collisions of one or two functions, i.e. to search for differing values x and y for which $f(x) = f(y)$, or, respectively, $f(x) = g(y)$ if two functions f and g are given.

We now give the basic version of Grover's algorithm.

The crucial effect of Grover's operator G (cf. Algorithm 5.1) is to rotate the state away from the equilibrium $N^{-1/2} \sum |x\rangle$ where x runs through the

Algorithm 5.1 Grover's search algorithm

Input: Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ given by the associated operator $U_f : \mathbb{F}_2^n \times \mathbb{F}_2 \to \mathbb{F}_2^n \times \mathbb{F}_2^s : |x\rangle|y\rangle \longmapsto |x\rangle|y \oplus f(x)\rangle$, and $M = \mathrm{card} f^{-1}(1)$.
Output: Some $y \in \mathbb{F}_2^n$ with $f(y) = 1$.

1: If $M > 3/4 \cdot 2^n$, then choose y randomly and uniformly from $\mathbb{F}_2^n$ and **return** y.
2: Compute θ satisfying $\sin^2\theta = M/2^n$, and set $r \leftarrow \lfloor \pi/(4\theta) \rfloor$.
3: Transform

$$|0\rangle|1\rangle \xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} |x\rangle(|0\rangle - |1\rangle) \xrightarrow{G^r} \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} \alpha_x |x\rangle(|0\rangle - |1\rangle),$$

where $G = U_f \cdot (H^{\otimes n}(2\,|0\rangle\,\langle 0| - 1)H^{\otimes n}) \otimes \mathrm{Id}$.
4: Measure and output the first n bits of the result.

whole domain of f towards $\omega = M^{-1/2} \sum |y\rangle$ where the sum is only over those y which are mapped to 1 by f. The angle of the rotation is computed in step 2 of the algorithm. The number r of iterations in step 3 minimizes the angle between the final state before measurement, and ω.

Run-time and success probability of the algorithm are given by the following proposition.

Proposition 3. *Suppose we are given a classical circuit consisting of no more than K gates which computes the boolean function $f : \mathbb{F}_2^n \to F_2$. Let $M = \mathrm{card} f^{-1}(1)$, and $N = 2^n$. Then Grover's algorithm runs in time $O(K \cdot \sqrt{N/M})$ and succeeds in finding a pre-image of 1 with probability greater $1/4$.*

Proofs of this and the following propositions can be found e.g. in [33]

Remark 3. If Grover's operator G is applied only r/l times, for some $l > 1$, instead of r times as specified, then the success probability of the algorithm drops to $O(1/l^2)$.

This remark shows that it seems crucial to know the number M of elements in $f^{-1}(1)$ to find one element in it. One approach to circumvent this problem is to guess in a binary search manner a sufficiently good approximation for M. It is, however, also possible to apply Grover's technique to find M directly.

Quantum counting. Successive applications of the Grover operator first increase the amplitude of the elements in the pre-image of 1, then decrease it when the state vector is rotated beyond ω, then increase it again when approaching $-\omega$, and so forth. We can employ QFT to measure the period of this evolution. The equations in step 2 of the algorithms allow the extraction of the cardinality of the pre-image from the obtained period.

Proposition 4. *There is a quantum algorithm which computes for a boolean function f on $\mathbb{F}_2^n$ with values in $\mathbb{F}_2$ the cardinality M of $f^{-1}(1)$ in time $O((1/\epsilon)\sqrt{2^n/(M+1)})$ with error probability smaller than ϵ.*

Now it is clear that we can first apply the counting algorithm to a boolean function for which $\mathrm{card} f^{-1}(1)$ is not known, and then Grover's original algorithm to actually find a pre-image of 1. Indeed, it is possible to combine these two steps.

Quantum collision search. A special, cryptanalytically highly relevant type of search is that of collisions of a function, i.e. the search of two arguments yielding the same function value. Like in the classical situation, there is a time memory trade-off which allows us to speed up such a search in comparison to simple searches for the pre-image of a random function value.

For this purpose one selects a subset $\mathcal{M}$ of the domain of the given function f. Let M denote its cardinality. The set $\mathcal{M}$ is then put into memory (read-only access suffices), and the Grover algorithm is applied to the function

$$g : \mathbb{F}_2^n \to \mathbb{F}_2 \; : \; x \longmapsto \begin{cases} 1 & \text{if there is a } y \in \mathcal{M} \text{ with } x \neq y \text{ and } f(x) = f(y), \\ 0 & \text{else .} \end{cases}$$

Proposition 5. *For all $k, M \in \mathbb{N}$ there is a quantum algorithm with the following properties. Suppose f is a function on $\mathbb{F}_2^n$ which can be computed in time K for which we have $\mathrm{card} f^{-1}(x) = M$ for all x. Then the algorithm finds (with success probability larger than $1/4$) two distinct x_1 and x_2 with $f(x_1) = f(x_2)$ in time $O(K(k + \sqrt{N/(kM)}))$.*

Remark 4. For collision search we have the same run-time success probability trade-off we had for general quantum searches: If the run-time is shortened by a factor $c < 1$, then the success probability is lowered by a factor c^2.

6 Outlook

Quantum computation forces us to reexamine the cryptosystems we use. Some systems have been broken, and other systems need to be examined for security. Some new systems may be special cases of existing systems that are more efficient, or they may be quantum inspired from the particular quantum problems. In any case, it will be some time before we can feel confident that quantum computers cannot break any given system. Given that this chapter has been about breaking systems, we have perhaps taken a more cautious approach to what is secure. However, the rest of this book provides alternatives which may very well be immune to quantum attacks.

Lattice based systems provide a good alternative since they are based on a long-standing open problem for classical computation. Efforts to make it more secure may make it a reasonable alternative. Or, it may make the system vulnerable to classical or quantum attacks.

Another option is security assumptions coming from the hidden subgroup problem. This has probably been the most widely studied problem for more than a decade. It represents a generalization of most existing exponential

speedups by quantum computing, and a solution for the nonabelian case would result in an efficient quantum algorithm for graph isomorphism. Based on this hardness, it was recently suggested for use as a cryptographic primitive. However, it is not known how to embed a trap-door yet, so this is still a open area also. The code based systems may be related to the nonabelian HSP.

References

1. Dorit Aharonov, Vaughan Jones, and Zeph Landau. A polynomial quantum algorithm for approximating the jones polynomial. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 427–436, New York, NY, USA, 2006. ACM Press.
2. Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 284–293, El Paso, Texas, 4–6 May 1997.
3. Dave Bacon, Andrew M. Childs, and Wim van Dam. From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semidirect product groups. In *46th Annual IEEE Symposium on Foundations of Computer Science*, pages 469–478, 2005.
4. Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, October 1997.
5. Johannes Buchmann, Markus Maurer, and Bodo Möller. Cryptography based on number fields with large regulator. *Journal de Théorie des Nombres de Bordeaux*, 12:293–307, 2000.
6. Johannes A. Buchmann and Hugh C. Williams. A key exchange system based on real quadratic fields (extended abstract). In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 335–343. Springer-Verlag, 1990, 20–24 August 1989.
7. Henri Cohen. *A course in computational algebraic number theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1993.
8. Ivan Damgård, Oded Goldreich, and Avi Wigderson. Hashing functions can simplify zero-knowledge protocol design (too). Technical Report RS-94-39, BRICS, 1994.
9. Mark Ettinger, Peter Høyer, and Emanuel Knill. The quantum query complexity of the hidden subgroup problem is polynomial. *Information Processing Letters*, 91(2):43–48, 2004.
10. Katalin Friedl, Gabor Ivanyos, Frederic Magniez, Miklos Santha, and Pranab Sen. Hidden translation and orbit coset in quantum computing. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, San Diego, CA, 9–11June 2003.
11. O. Goldreich, S. Micali, and A. Widgerson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.
12. Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, New York, NY, USA, 2001.

13. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.
14. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *LNCS*, pages 112–131. SV, 1997.
15. Oded Goldreich, Amit Sahai, and Salil Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 399–408, 1998.
16. Oded Goldreich and Salil Vadhan. Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In *Proceedings of 14th Annual IEEE Conference on Computational Complexity*, 1999.
17. Sean Hallgren. Fast quantum algorithms for computing the unit group and class group of a number field. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 468–474, 2005.
18. Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *Journal of the ACM*, 54(1):1–19, 2007.
19. Sean Hallgren, Alexandra Kolla, Pranab Sen, and Shengyu Zhang. Making classical honest verifier zero knowledge protocols secure against quantum attacks. *Automata, Languages and Programming*, pages 592–603, 2008.
20. Sean Hallgren, Cristopher Moore, Martin Rötteler, Alexander Russell, and Pranab Sen. Limitations of quantum coset states for graph isomorphism. In *STOC '06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 604–617, New York, NY, USA, 2006. ACM Press.
21. Sean Hallgren, Alexander Russell, and Amnon Ta-Shma. Normal subgroup reconstruction and quantum computation using group representations. *SIAM Journal on Computing*, 32(4):916–934, 2003.
22. Gábor Ivanyos, Frédéric Magniez, and Miklos Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 263–270, Heraklion, Crete Island, Greece, 4-6July 2001.
23. Gábor Ivanyos, Luc Sanselme, and Miklos Santha. An efficient quantum algorithm for the hidden subgroup problem in extraspecial groups, 2007.
24. Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.
25. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
26. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 564–572, Baltimore, Maryland, 14–16 May 1990.
27. A.K. Lenstra and H.W. Lenstra, editors. *The Development of the Number Field Sieve*, volume 1544 of *Lecture Notes in Mathematics*. Springer–Verlag, 1993.
28. Cristopher Moore, Alexander Russell, and Leonard Schulman. The symmetric group defies strong Fourier sampling. In *Proceedings of the Symposium on the Foundations of Computer Science (FOCS'05)*, pages 479–488, 2005.
29. Cristopher Moore, Alexander Russell, and Piotr Sniady. On the impossibility of a quantum sieve algorithm for graph isomorphism. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 536–545, New York, NY, USA, 2007. ACM Press.
30. Cristopher Moore, Alexander Russell, and Umesh Vazirani. A classical one-way function to confound quantum adversaries. quant-ph/0701115, 2007.

31. M.A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29:183–205, 1975.
32. Minh-Huyen Nguyen, Shien Jin Ong, and Salil Vadhan. Statistical zero-knowledge arguments for NP from any one-way function. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 3–14, 2006.
33. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambirdge University Press, 2000.
34. C. Pomerance. Factoring. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 27–47. American Mathematical Society, 1990.
35. Oded Regev. Quantum computation and lattice problems. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 520–529, Los Alamitos, 2002.
36. Amit Sahai and Salil Vadhan. A complete promise problem for statistical zero knowledge. *Journal of the ACM*, 50(2):196–249, 2003.
37. Arthur Schmidt and Ulrich Vollmer. Polynomial time quantum algorithm for the computation of the unit group of a number field. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 475–480, 2005.
38. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
39. Salil Pravin Vadhan. *A Study of Statistical Zero-Knowledge Proofs*. PhD thesis, Massachusetts Institute of Technology, 1999.
40. Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum algorithms for some hidden shift problems. *SIAM Journal on Computing*, 36(3):763–778, 2006.
41. John Watrous. Zero-knowledge against quantum attacks. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 296–305, 2006.

Hash-based Digital Signature Schemes

Johannes Buchmann[1], Erik Dahmen[1], and Michael Szydlo[2]

[1] Department of Computer Science, Technische Universität Darmstadt.
[2] Akamai Technologies, Cambridge.

Digital signatures have become a key technology for making the Internet and other IT-infrastructures secure. Digital signatures provide authenticity, integrity, and non-repudiation of data. Digital signatures are widely used in identification and authentication protocols. Therefore, the existence of secure digital signature algorithms is crucial for maintaining IT-security.

The digital signature algorithms that are used in practice today are RSA [31], DSA [11], and ECDSA [15]. They are not quantum immune since their security relies on the difficulty of factoring large composite integers and computing discrete logarithms.

Hash-based digital signature schemes which are presented in this chapter offer a very interesting alternative. Like any other digital signature scheme, hash-based digital signature schemes use a cryptographic hash function. Their security relies on the collision resistance of that hash function. In fact, we will present hash-based digital signature schemes that are secure if and only if the underlying hash function is collision resistant. The existence of collision resistant hash functions can be viewed as a minimum requirement for the existence of a digital signature scheme that can sign many documents with one private key. That signature scheme maps documents (arbitrarily long bit strings) to digital signatures (bit strings of fixed length). This shows that digital signature algorithms are in fact hash functions. Those hash functions must be collision resistant: if it were possible to construct two documents with the same digital signature, the signature scheme could no longer be considered secure. This argument shows that there exist hash-based digital signature schemes as long as there exists any digital signature scheme that can sign multiple documents using one private key. As a consequence, hash-based signature schemes are the most important post-quantum signature candidates. Although there is no proof of their quantum computer resistance, their security requirements are minimal. Also, each new cryptographic hash function yields a new hash-based signature scheme. So the construction of secure signature schemes is independent of hard algorithmic problems in number theory or algebra. Constructions from symmetric cryptography suffice. This leads to another big advantage of

hash-based signature schemes. The underlying hash function can by chosen in view of the hardware and software resources available. For example, if the signature scheme is to be implemented on a chip that already implements AES, an AES based hash function can be used, thereby reducing the code size of the signature scheme and optimizing its running time.

Hash-based signature schemes were invented by Ralph Merkle [23]. Merkle started from one-time signature schemes, in particular that of Lamport and Diffie [18]. One-time signatures are even more fundamental. The construction of a secure one-time signature scheme only requires a one-way function. As shown by Rompel [28], one-way functions are necessary and sufficient for secure digital signatures. So one-time signature schemes are really the most fundamental type of digital signature schemes. However, they have a severe disadvantage. One key-pair consisting of a secret signature key and a public verification key can only be used to sign and verify a single document. This is inadequate for most applications. It was the idea of Merkle to use a hash tree that reduces the validity of many one-time verification keys (the leaves of the hash tree) to the validity of one public key (the root of the hash tree). The initial construction of Merkle was not sufficiently efficient, in particular in comparison to the RSA signature scheme. However in the meantime, many improvements have been found. Now hash-based signatures are the most promising alternative to RSA and elliptic curve signature schemes.

1 Hash based one-time signature schemes

This chapter explains signature schemes whose security is only based on the collision resistance of a cryptographic hash function. Those schemes are particularly good candidates for the post quantum era.

1.1 Lamport–Diffie one-time signature scheme

The Lamport–Diffie one-time signature scheme (LD-OTS) was proposed in [18]. Let n be a positive integer, the security parameter of LD-OTS. LD-OTS uses a one-way function

$$f : \{0,1\}^n \to \{0,1\}^n,$$

and a cryptographic hash function

$$g : \{0,1\}^* \to \{0,1\}^n.$$

LD-OTS key pair generation. The signature key X of LD-OTS consists of $2n$ bit strings of length n chosen uniformly at random,

$$X = \big(x_{n-1}[0], x_{n-1}[1], \ldots, x_1[0], x_1[1], x_0[0], x_0[1]\big) \in_R \{0,1\}^{(n,2n)}. \qquad (1)$$

The LD-OTS verification key Y is

$$Y = \big(y_{n-1}[0], y_{n-1}[1], \ldots, y_1[0], y_1[1], y_0[0], y_0[1]\big) \in \{0,1\}^{(n,2n)}, \tag{2}$$

where

$$y_i[j] = f\big(x_i[j]\big), \quad 0 \le i \le n-1, j = 0, 1. \tag{3}$$

So LD-OTS key generation requires $2n$ evaluations of f. The signature and verification keys are $2n$ bit strings of length n.

LD-OTS signature generation. A document $M \in \{0,1\}^*$ is signed using LD-OTS with a signature key X as in Equation (1). Let $g(M) = d = (d_{n-1}, \ldots, d_0)$ be the message digest of M. Then the LD-OTS signature is

$$\sigma = \big(x_{n-1}[d_{n-1}], \ldots, x_1[d_1], x_0[d_0]\big) \in \{0,1\}^{(n,n)}. \tag{4}$$

This signature is a sequence of n bit strings, each of length n. They are chosen as a function of the message digest d. The ith bit string in this signature is $x_i[0]$ if the ith bit in d is 0 and $x_i[1]$, otherwise. Signing requires no evaluations of f. The length of the signature is n^2.

LD-OTS Verification. To verify a signature $\sigma = (\sigma_{n-1}, \ldots, \sigma_0)$ of M as in (4), the verifier calculates the message digest $d = (d_{n-1}, \ldots, d_0)$. Then she checks whether

$$\big(f(\sigma_{n-1}), \ldots, f(\sigma_0)\big) = \big(y_{n-1}[d_{n-1}], \ldots, y_0[d_0]\big). \tag{5}$$

Signature verification requires n evaluations of f.

Example 1. Let $n = 3$, $f : \{0,1\}^3 \to \{0,1\}^3, x \mapsto x + 1 \bmod 8$, and let $d = (1,0,1)$ be the hash value of a message M. We choose the signature key

$$X = \big(x_2[0], x_2[1], x_1[0], x_1[1], x_0[0], x_0[1]\big) = \begin{pmatrix} 1\,0 & 0\,1 & 1\,0 \\ 1\,0 & 1\,1 & 0\,1 \\ 1\,0 & 1\,0 & 1\,0 \end{pmatrix} \in \{0,1\}^{(3,6)}$$

and compute the corresponding verification key

$$Y = \big(y_2[0], y_2[1], y_1[0], y_1[1], y_0[0], y_0[1]\big) = \begin{pmatrix} 0\,0 & 1\,1 & 1\,0 \\ 0\,0 & 0\,1 & 1\,1 \\ 0\,1 & 0\,1 & 0\,1 \end{pmatrix} \in \{0,1\}^{(3,6)}.$$

The signature of $d = (1,0,1)$ is

$$\sigma = (\sigma_2, \sigma_1, \sigma_0) = (x_2[1], x_1[0], x_0[1]) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \in \{0,1\}^{(3,3)}$$

Example 2. We give an example to illustrate why the signature keys of LD-OTS must be used only once. Let $n = 4$. Suppose the signer signs two messages with digests $d_1 = (1, 0, 1, 1)$ and $d_2 = (1, 1, 1, 0)$ using the same signature key. The signatures of these digests are $\sigma_1 = (x_3[1], x_2[0], x_1[1], x_0[1])$ and $\sigma_2 = (x_3[1], x_2[1], x_1[1], x_0[0])$, respectively. Then an attacker knows $x_3[1], x_2[0], x_2[1], x_1[1], x_0[0], x_0[1]$ from the signature key. She can use this information to generate valid signatures for messages with digests $d_3 = (1, 0, 1, 0)$ and $d_4 = (1, 1, 1, 1)$. This example can be generalized to arbitrary security parameters n. Also, the attacker is only able to generate valid signatures for certain digests. As long as the hash function used to compute the message digest is cryptographically secure, she cannot find appropriate messages.

1.2 Winternitz one-time signature scheme

While the key and signature generation of LD-OTS is very efficient, the size of the signature is quite large. The Winternitz OTS (W-OTS), which is explained in this section, produces significantly shorter signatures. The idea is to use one string in the one-time signature key to simultaneously sign several bits in the message digest. In literature this proposal appears first in Merkle's thesis [23]. Merkle writes that the method was suggested to him by Winternitz in 1979 as a generalization of the Merkle OTS also described in [23]. However, to the best of the authors knowledge, the Winternitz OTS was for the first time described in full detail in [10]. Like LD-OTS, W-OTS uses a one-way function

$$f : \{0,1\}^n \to \{0,1\}^n$$

and a cryptographic hash function

$$g : \{0,1\}^* \to \{0,1\}^n.$$

W-OTS key pair generation. A Winternitz parameter $w \geq 2$ is selected which is the number of bits to be signed simultaneously. Then

$$t_1 = \left\lceil \frac{n}{w} \right\rceil, \quad t_2 = \left\lceil \frac{\lfloor \log_2 t_1 \rfloor + 1 + w}{w} \right\rceil, \quad t = t_1 + t_2. \tag{6}$$

are determined. The signature key X is

$$X = \big(x_{t-1}, \ldots, x_1, x_0\big) \in_R \{0,1\}^{(n,t)}. \tag{7}$$

where the bit strings x_i are chosen uniformly at random.

The verification key Y is computed by applying f to each bit string in the signature key $2^w - 1$ times. So we have

$$Y = \big(y_{t-1}, \ldots, y_1, y_0\big) \in \{0,1\}^{(n,t)}, \tag{8}$$

where

$$y_i = f^{2^w-1}\big(x_i\big), 0 \leq i \leq t-1. \tag{9}$$

Key generation requires $t(2^w - 1)$ evaluations of f and the lengths of the signature and verification key are $t \cdot n$ bits, respectively.

W-OTS signature generation. A message M with message digest $g(M) = d = (d_{n-1}, \ldots, d_0)$ is signed. First, a minimum number of zeros is prepended to d such that the length of d is divisible by w. The extended string d is split into t_1 bit strings $b_{t-1}, \ldots, b_{t-t_1}$ of length w. Then

$$d = b_{t-1} \| \ldots \| b_{t-t_1}, \tag{10}$$

where $\|$ denotes concatenation. Next, the bit strings b_i are identified with integers in $\{0, 1, \ldots, 2^w - 1\}$ and the checksum

$$c = \sum_{i=t-t_1}^{t-1} (2^w - b_i) \tag{11}$$

is calculated. Since $c \leq t_1 2^w$, the length of the binary representation of c is less than

$$\lfloor \log_2 t_1 2^w \rfloor + 1 = \lfloor \log_2 t_1 \rfloor + w + 1. \tag{12}$$

A minimum number of zeros is prepended to this binary representation such that the length of the extended string is divisible by w. That extended string is split into t_2 blocks $b_{t_2-1}, \ldots, b_0$ of length w. Then

$$c = b_{t_2-1} || \ldots || b_0.$$

Finally the signature of M is computed as

$$\sigma = \big(f^{b_{t-1}}(x_{t-1}), \ldots, f^{b_1}(x_1), f^{b_0}(x_0)\big). \tag{13}$$

In the worst case, signature generation requires $t(2^w - 1)$ evaluations of f. The W-OTS signature size is $t \cdot n$.

W-OTS verification. For the verification of the signature $\sigma = (\sigma_{t-1}, \ldots, \sigma_0)$ the bit strings $b_{t-1}, \ldots, b_0$ are calculated as explained in the previous section. Then we check if

$$\big(f^{2^w-1-b_{t-1}}(\sigma_{n-1}), \ldots, f^{2^w-1-b_0}(\sigma_0)\big) = \big(y_{n-1}, \ldots, y_0\big). \tag{14}$$

If the signature is valid, then $\sigma_i = f^{b_i}(x_i)$ and therefore

$$f^{2^w-1-b_i}(\sigma_i) = f^{2^w-1}(x_i) = y_i \tag{15}$$

holds for $i = t-1, \ldots, 0$. In the worst case, signature verification requires $t(2^w - 1)$ evaluations of f.

Example 3. Let $n = 3, w = 2$, $f : \{0,1\}^3 \to \{0,1\}^3, x \mapsto x + 1 \bmod 8$ and $d = (1, 0, 0)$. We get $t_1 = 2, t_2 = 2$, and $t = 4$. We choose the signature key as

$$X = \big(x_3, x_2, x_1, x_0\big) = \begin{pmatrix} 1\,0\,0\,1 \\ 1\,0\,1\,1 \\ 1\,0\,1\,0 \end{pmatrix} \in \{0,1\}^{(3,4)}$$

and compute the verification key by applying f three times to the bit strings in X:

$$Y = \big(y_3, y_2, y_1, y_0\big) = \begin{pmatrix} 0\,0\,1\,0 \\ 1\,1\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix} \in \{0,1\}^{(3,4)}.$$

Prepending one zero to d and splitting the extended string into blocks of length 2 yields $d = 01||00$. The checksum c is $c = (4-1) + (4-0) = 7$. Prepending one zero to the binary representation of c and splitting the extended string into blocks of length 2 yields $c = 01||11$. The signature is

$$\sigma = (\sigma_3, \sigma_2, \sigma_1, \sigma_0) = \big(f(x_3), x_2, f(x_1), f^3(x_0)\big) = \begin{pmatrix} 0\,0\,1\,1 \\ 0\,0\,0\,1 \\ 0\,0\,0\,1 \end{pmatrix} \in \{0,1\}^{(3,4)}.$$

The signature is verified by computing

$$\big(f^2(\sigma_3), f^3(\sigma_2), f^2(\sigma_1), \sigma_0\big) = \begin{pmatrix} 0\,0\,1\,0 \\ 1\,1\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix} \in \{0,1\}^{(3,4)}$$

and comparing it with the verification key Y.

Example 4. We give an example to illustrate why the signature keys of the W-OTS must be used only once. Let $w = 2$. Suppose the signer signs two messages with digests $d_1 = (1,0,0)$ and $d_2 = (1,1,1)$ using the same signature key. The signatures of these digests are $\sigma_1 = \big(f(x_3), x_2, f(x_1), f^3(x_0)\big)$ and $\sigma_2 = \big(f(x_3), f^3(x_2), f(x_1), x_0\big)$, respectively. The attacker can use this information to compute the signatures for messages with digest $d_3 = (1,1,0)$ given as $\sigma_3 = \big(f(x_3), f^2(x_2), f(x_1), f(x_0)\big)$ Again this example can be generalized to arbitrary security parameters n. Also, the attacker can only produce valid signatures for certain digests. As long as the hash function used to compute the message digest is cryptographically secure, he cannot find appropriate messages.

2 Merkle's tree authentication scheme

The one-time signature schemes introduced in the last section are inadequate for most practical situations since each key pair can only be used for one signature. In 1979 Ralph Merkle proposed a solution to this problem [23]. His idea is to use a complete binary hash tree to reduce the validity of an arbitrary but fixed number of one time verification keys to the validity of one single public key, the root of the hash tree.

The Merkle signature scheme (MSS) works with any cryptographic hash function and any one-time signature scheme. For the explanation we let $g : \{0,1\}^* \to \{0,1\}^n$ be a cryptographic hash function. We also assume that a one-time signature scheme has been selected.

MSS key pair generation

The signer selects $H \in \mathbb{N}$, $H \geq 2$. Then the key pair to be generated will be able to sign/verify 2^H documents. Note that this is an important difference to signature schemes such as RSA and ECDSA, where potentially arbitrarily many documents can be signed/verified with one key pair. However, in practice this number is also limited by the devices on which the signature is generated or by some policy. The signer generates 2^H one-time key pairs (X_j, Y_j), $0 \leq j < 2^H$. Here X_j is the signature key and Y_j is the verification key. They are both bit strings. The leaves of the Merkle tree are the digests $g(Y_j)$, $0 \leq j < 2^H$. The inner nodes of the Merkle tree are computed according to the following construction rule: a parent node is the hash value of the concatenation of its left and right children. The MSS public key is the root of the Merkle tree. The MSS private key is the sequence of the 2^H one-time signature keys. To be more precise, denote the nodes in the Merkle tree by $\nu_h[j]$, $0 \leq j < 2^{H-h}$, where $h \in \{0, \ldots, H\}$ is the height of the node. Then

$$\nu_h[j] = g(\nu_{h-1}[2j] \| \nu_{h-1}[2j+1]), \quad 1 \leq h \leq H, 0 \leq j < 2^{H-h}. \tag{16}$$

Figure 1 shows an example for $H = 3$.

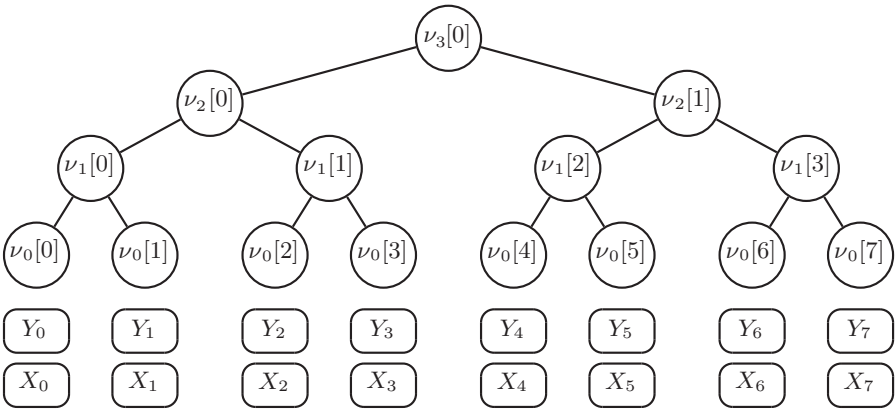

Fig. 1. A Merkle tree of height $H = 3$

MSS key pair generation requires the computation of 2^H one-time key pairs and $2^{H+1} - 1$ evaluations of the hash function.

Efficient root computation

In order to compute the root of the Merkle tree it is not necessary to store the full hash tree. Instead, the treehash algorithm 2.1 is applied. The basic idea of this algorithm is to successively compute leaves and, whenever possible,

compute their parents. To store nodes, the treehash algorithm uses a stack STACK equipped with the usual push and pop operations. Input of the tree hash algorithm is the height H of the Merkle tree. Output is the root of the Merkle tree, i.e. the MSS public key. Algorithm 2.1 uses the subroutine LEAFCALC(j) to compute the jth leaf. The LEAFCALC(j) routine computes the jth one-time key pair and computes the jth leaf from the jth one-time verification key as described above.

Algorithm 2.1 Treehash

Input: Height $H \geq 2$
Output: Root of the Merkle tree

1. **for** $j = 0, \ldots, 2^H - 1$ **do**
   a) Compute the jth leaf: $\text{NODE}_1 \leftarrow \text{LEAFCALC}(j)$
   b) **While** NODE_1 has the same height as the top node on STACK **do**
      i. Pop the top node from the stack: $\text{NODE}_2 \leftarrow \text{STACK.pop}()$
      ii. Compute their parent node: $\text{NODE}_1 \leftarrow g(\text{NODE}_2 \| \text{NODE}_1)$
   c) Push the parent node on the stack: $\text{STACK.push}(\text{NODE}_1)$
2. Let R be the single node stored on the stack: $R \leftarrow \text{STACK.pop}()$
3. **Return** R

Figure 2 shows the order in which the nodes of a Merkle tree are computed by the treehash algorithm. In this example, the maximum number of nodes that are stored on the stack is 3. This happens after node 11 is generated and pushed on the stack. In general, the treehash algorithm needs to store at most H so-called *tail nodes* on the stack. To compute the root of a Merkle tree of height H, the treehash algorithm requires 2^H calls of the LEAFCALC subroutine, and $2^H - 1$ evaluations of the hash function.

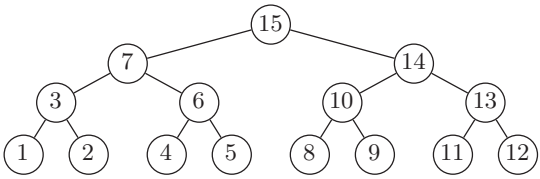

Fig. 2. The treehash algorithm

MSS signature generation

MSS uses the one-time signature keys successively for the signature generation. To sign a message M, the signer first computes the n-bit digest $d = g(M)$. Then he generates the one-time signature σ_{OTS} of the digest using the sth

one-time signature key X_s, $s \in \{0, \ldots, 2^H - 1\}$. The Merkle signature will contain this one-time signature and the corresponding one-time verification key Y_s. To prove the authenticity of Y_s to the verifier, the signer also includes the index s as well as an authentication path for the verification key Y_s which is a sequence $A_s = (a_0, \ldots, a_{H-1})$ of nodes in the Merkle tree. This index and the authentication path allow the verifier to construct a path from the leaf $g(Y_s)$ to the root of the Merkle tree. Node h in the authentication path is the sibling of the height h node on the path from leaf $g(Y_s)$ to the Merkle tree root:

$$a_h = \begin{cases} \nu_h[s/2^h - 1] \text{ , if } \lfloor s/2^h \rfloor \equiv 1 \bmod 2 \\ \nu_h[s/2^h + 1] \text{ , if } \lfloor s/2^h \rfloor \equiv 0 \bmod 2 \end{cases} \tag{17}$$

for $h = 0, \ldots H - 1$. Figure 3 shows an example for $s = 3$. So the sth Merkle signature is

$$\sigma_s = \big(s, \sigma_{\mathrm{OTS}}, Y_s, (a_0, \ldots, a_{H-1})\big) \tag{18}$$

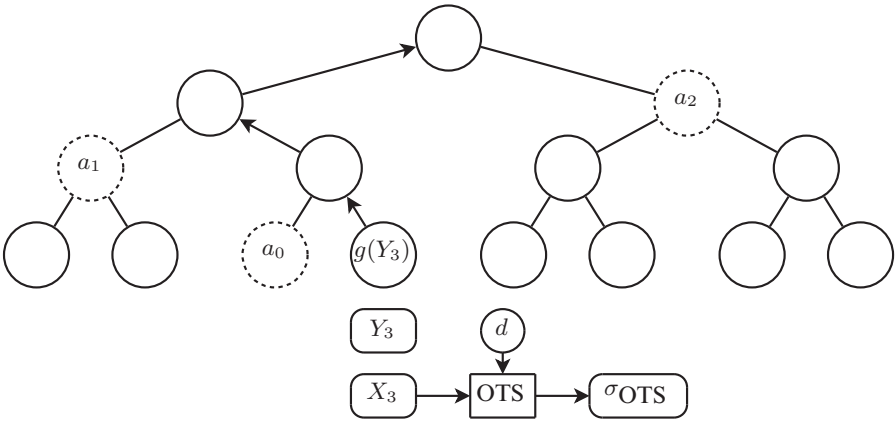

Fig. 3. Merkle signature generation for $s = 3$. Dashed nodes denote the authentication path for leaf $g(Y_3)$. Arrows indicate the path from leaf $g(Y_3)$ to the root.

MSS signature verification

Verification of the Merkle signature from the previous section consists of two steps. In the first step, the verifier uses the one-time verification key Y_s to verify the one-time signature σ_{OTS} of the digest d by means of the verification algorithm of the respective one-time signature scheme. In the second step the verifier validates the authenticity of the one-time verification key Y_s by constructing the path $(p_0, \ldots, p_H)$ from the sth leaf $g(Y_s)$ to the root of the Merkle tree. He uses the index s and the authentication path $(a_0, \ldots, a_{H-1})$ and applies the following construction.

$$p_h = \begin{cases} g(a_{h-1}||p_{h-1}) \text{ , if } \lfloor s/2^{h-1} \rfloor \equiv 1 \bmod 2 \\ g(p_{h-1}||a_{h-1}) \text{ , if } \lfloor s/2^{h-1} \rfloor \equiv 0 \bmod 2 \end{cases} \tag{19}$$

for $h = 1, \ldots H$ and $p_0 = g(Y_s)$. The index s is used for deciding in which order the authentication path nodes and the nodes on the path from leaf $g(Y_s)$ to the Merkle tree root are to be concatenated. The authentication of the one-time verification key Y_s is successful if and only if p_H equals the public key.

3 One-time key-pair generation using an PRNG

According to the description of MSS from Section 2, the MSS private key consists of 2^H one-time signature keys. Storing such a huge amount of data is not feasible for most practical applications. As suggested in [3], space can be saved by using a deterministic pseudo random number generator (PRNG) and storing only the seed of that PRNG. Then each one-time signature key must be generated twice, once for the MSS public key generation and once during the signing phase.

In the following, let PRNG be a cryptographically secure pseudo random number generator that on input an n-bit seed SEED_{in} outputs a random number RAND and an updated seed SEED_{out}, both of bit length n.

$$\begin{aligned} \text{PRNG} : \{0,1\}^n &\to \{0,1\}^n \times \{0,1\}^n \\ \text{SEED}_{\text{in}} &\mapsto (\text{RAND}, \text{SEED}_{\text{out}}) \end{aligned} \tag{20}$$

MSS key pair generation using an PRNG

We explain how MSS key-pair generation using a PRNG works. The first step is to choose an n-bit seed SEED_0 uniformly at random. For the generation of the one-time signature keys we use a sequence of seeds SEEDOTS_j, $0 \leq j < 2^H$. They are computed iteratively using

$$(\text{SEEDOTS}_j, \text{SEED}_{j+1}) = \text{PRNG}(\text{SEED}_j), 0 \leq j < 2^H. \tag{21}$$

Here SEEDOTS_j is used to calculate the jth one-time signature key.

For example, in the case of W-OTS (see Section 1.2) the jth signature key is $X_j = (x_{t-1}, \ldots, x_0)$. The t bit strings of length n in this signature key are generated using SEEDOTS_j.

$$(x_i, \text{SEEDOTS}_j) = \text{PRNG}(\text{SEEDOTS}_j), i = t-1, \ldots, 0 \tag{22}$$

The seed SEEDOTS_j is updated during each call to the PRNG. This shows that in order to calculate the signature key X_j only knowledge of SEED_j is necessary. When SEEDOTS_j is computed, the new seed SEED_{j+1} for the

generation of the signature key X_{j+1} is also determined. Figure 4 visualizes the one-time signature key generation using an PRNG.

If this method is used, the MSS private key is initially SEED_0. Its length is n. It is replaced by the seeds SEED_{j+1} determined during the generation of signature key X_j.

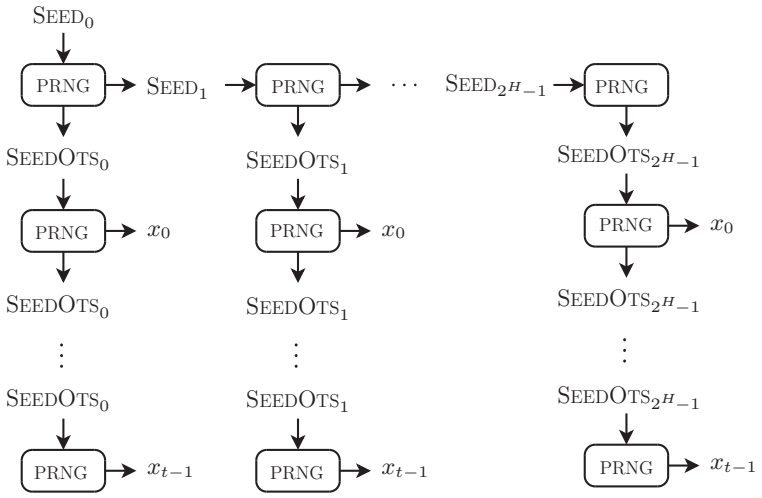

Fig. 4. One-time signature key generation using an PRNG

MSS signature generation using an PRNG

In contrast to the original MSS signature generation, the one-time signature key must be computed before the signature is generated. When the signature key is computed the seed is updated for the next signature.

Forward security

In addition to reducing the private key size, using a PRNG for the one-time signature key generation has another benefit. It makes MSS forward secure as long as PRNG is forward secure which means that calculating previous seeds from the actual seed is infeasible. Forward security of the signature scheme means that all signatures issued before a revocation remain valid. MSS is forward secure, since the actual MSS private key can only be used to generate one-time signature keys for upcoming signatures but not to forge previous.

4 Authentication path computation

In this chapter we will present a variety of techniques for traversal of Merkle trees of height H. The use of the techniques is *transparent* to a verifier, who will not need to know how a set of outputs were generated, but only that they are correct. Therefore, the technique can be employed in any construction for which the generation and output of authentication paths for consecutive leaves is required.

The first traversal algorithm is structurally very simple and allows for various tradeoffs between storage and computation. For one choice of parameters, the total space required is bounded by $1.5H^2/\log H$ hash values, and the worst-case computational effort is $2H/\log H$ tree node computations per output.

The next Merkle tree-traversal algorithm has a better space and time complexity than the previously known algorithms. Specifically, the algorithm requires computation of at most $2H$ tree nodes per round and requires storage of less than $3H$ node values. We also prove that this complexity is optimal in the sense that there can be no Merkle Tree traversal algorithm which requires both less than $O(H)$ time and less than $O(H)$ space.

In the analysis of the first two algorithms, the computation of a leaf and an inner node are each counted as a single elementary operation[1].

The third Merkle tree-traversal algorithm has the same space and time complexity as the second. However it has a significant constant factor improvement and was designed for practical implementation. It distinguishes between leaf computations and the computation of inner nodes. To traverse a tree of height H it roughly requires the computation of $H/2$ leaves and $3H/2$ inner nodes.

4.1 The Classic Traversal

The challenge of Merkle tree traversal is to ensure that all node values are ready when needed, but are computed in a manner which conserves space and time. To motivate the new algorithms, we first discuss what the average per-round computation is expected to be, and review the classic Merkle tree traversal.

Average Costs. Each node in the tree is eventually part of an authentication path, so one useful measure is the total cost of computing each node value exactly once. There are 2^{H-h} right (respectively, left) nodes at height h, and if computed independently, each costs $2^{h+1} - 1$ operations. Rounding up, this is $2^{H+1} = 2N$ operations, or two per round. Adding together the costs for each height h ($0 \leq h < H$), we expect, on average, $2H = 2\log(N)$ operations per round to be required.

[1] This differs from the measurement of *total* computational cost, which includes, e.g., the scheduling algorithm itself.

Three Components. As with a digital signature scheme, the tree-traversal algorithms consists of three components: *key generation*, *output*, and *verification*. During *key generation*, the first authentication path and some upcoming authentication node values are computed.

The *output* phase consists of N rounds, one for each leaf $s \in \{0, \ldots, N-1\}$. During round s, the authentication path for the sth leaf, AUTH_i, $i = 0, \ldots, H-1$ is output. Additionally, the algorithm's state is modified in order to prepare for future outputs.

The *verification* phase is identical to the traditional verification phase for Merkle trees described in Section 2.

Notation. In addition to denoting the current authentication nodes AUTH_h, we need some notation to describe the stacks used to compute upcoming needed nodes. Define STACK_h to be an object which contains a stack of node values as in the description of the treehash algorithm in Section 2, Algorithm 2.1. STACK_h.*initialize* and STACK_h.*update* will be methods to setup and incrementally execute treehash.

Algorithm presentation

Key Generation and Setup. The main task of key generation is to compute and publish the root value. This is a direct application of the treehash algorithm described in Section 2. In the process of this computation, every node value is computed, and, it is important to record the initial values AUTH_i, as well as the upcoming values for each of the AUTH_i.

If we denote the jth node at height h by $\nu_h[j]$, we have $\text{AUTH}_h = \nu_h[1]$ (these are right nodes). The "upcoming" authentication node at height h is $\nu_h[0]$ (these are left nodes). These node values are used to initialize STACK_h to be in the state of the treehash algorithm having completed.

Algorithm 4.1 Key-Gen and Setup

1. **Initial Authentication Nodes** For each $h \in \{0, 1, \ldots H-1\}$:
   Calculate $\text{AUTH}_h = \nu_h[1]$.
2. **Initial Next Nodes** For each $h \in \{0, 1, \ldots H-1\}$:
   Setup STACK_h with the single node value $\text{AUTH}_h = \nu_h[0]$.
3. **Public Key** Calculate and publish tree root, $\nu_H[0]$.

Output and Update. Merkle's tree traversal algorithm runs one instance of the treehash algorithm for each height h to compute the next authentication node value for that level. Every 2^h rounds, the authentication path will shift to the right at level h, thus requiring a new node (its sibling) as the height h authentication node.

At each round the state of the treehash algorithm is updated with two units of computation. After 2^h rounds this node value computation will be completed, and a new instance of treehash begins for the next authentication node at that level.

To specify how to refresh the AUTH nodes, we observe how to easily determine which heights need updating: height h needs updating if and only if 2^h divides $s+1$ evenly, where $s \in \{0, \ldots, N-1\}$ denotes the current round. Furthermore, we note that at round $s+1+2^h$, the authentication *path* will pass though the $(s+1+2^h)/2^h$th node at height h. Thus, its *sibling's* value, (the new required upcoming AUTH_h) is determined from the 2^h leaf values starting from leaf number $(s+1+2^h) \oplus 2^h$, where $\oplus$ denotes bitwise XOR.

In this language, we summarize Merkle's classic traversal algorithm in Algorithm 4.2.

Algorithm 4.2 Classic Merkle Tree Traversal

1. Set $s = 0$.
2. **Output:**
- For each $h \in [0, H-1]$ output AUTH_h.
3. **Refresh Auth Nodes:**
   For all h such that 2^h divides $s+1$:
- Set AUTH_h be the sole node value in STACK_h.
- Set startnode $= (s+1+2^h) \oplus 2^h$.
- STACK_h.*initialize*(startnode, h).
4. **Build Stacks:**
   For all $h \in [0, H-1]$:
- STACK_h.*update*(2). (Each stack receives two updates)
5. **Loop:**
- Set $s = s+1$.
- If $s < 2^H$ go to Step 2.

4.2 Fractal Merkle Tree Traversal

The term "fractal" was chosen due to the focus on many smaller binary trees within the larger structure of the Merkle tree.

The crux of this algorithm is the selection of which node values to compute and retain at each step of the output algorithm. We describe this selection by using a collection of subtrees of fixed height h. We begin with some notation and then provide the intuition for the algorithm.

Notation. Starting with a Merkle tree TREE of height H, we introduce further notation to deal with subtrees. First we choose a *subtree height* $h < H$. We let the altitude of a node ν in TREE be the length of the path from ν to a leaf of TREE (therefore, the altitude of a leaf of TREE is zero). Consider a node ν

with altitude at least h. We define the h-subtree at ν to be the unique subtree in TREE which has ν as its root and which has height h. For simplicity in the suite, we assume h is a divisor of H, and let the ratio, $L = H/h$, be the number of *levels* of subtrees. We say that an *h-subtree at ν* is "at level i" when it has altitude ih for some $i \in \{1, 2, \ldots H\}$. For each i, there are 2^{H-ih} such h-subtrees at level i.

We say that a series of h-subtrees TREE_i $(i = 1 \ldots L)$ is a *stacked series of h-subtrees*, if for all $i < L$ the root of TREE_i is a leaf of TREE_{i+1}. We illustrate the subtree notation and provide a visualization of a *stacked series of h-subtrees* in Figure 5.

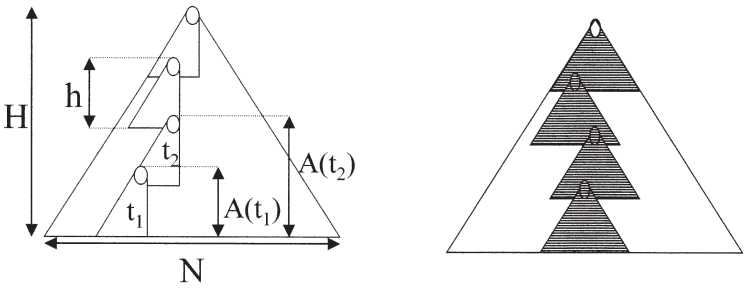

Fig. 5. *(Left)* The height of the Merkle tree is H, and thus, the number of leaves is $N = 2^H$. The height of each subtree is h. The altitude $A(t_1)$ and $A(t_2)$ of the subtrees t_1 and t_2 is marked. *(Right)* Instead of storing all tree nodes, we store a smaller set - those within the stacked subtrees. The leaf whose pre-image will be output next is contained in the lowest-most subtree; the entire authentication path is contained in the stacked set of subtrees.

Existing and Desired Subtrees

Static view. As previously mentioned, we store some portion of the node values, and update what values are stored over time. Specifically, during any point of the output phase, there will exist a series of stacked *existing* subtrees, as in Figure 2. We say that we place a *pebble* on a node ν of the tree TREE when we store this node. There are always L such subtrees EXIST_i for each $i \in \{1, \ldots L\}$, with pebbles on each of their nodes (except their roots). By design, for any leaf in EXIST_1, the corresponding authentication path is completely contained in the stacked set of existing subtrees.

Dynamic view. Apart from the above set of *existing* subtrees, which contain the next required authentication path, we will have a set of *desired* subtrees. If the root of the tree EXIST_i has index a, according to the ordering of the height-ih nodes, then DESIRE_i is defined to be the h-subtree with index $a+1$

(provided that $a < 2^{H-i \cdot h} - 1$). In case $a = 2^{H-i \cdot h} - 1$, then EXIST_i is the last subtree at this level, and there is no corresponding desired subtree. In particular, there is never a desired subtree at level L. The left part of Figure 6 depicts the adjacent existing and desired subtrees.

As the name suggests, we need to compute the pebbles in the desired subtrees. This is accomplished by adapting an application of the treehash algorithm (Section 2, Algorithm 2.1) to the root of DESIRE_i. For these purposes, the treehash algorithm is altered to save the pebbles needed for DESIRE_i, rather than discarding them, and secondly to terminate one round early, never actually computing the root. Using this variant of treehash, we see that each desired subtree being computed has a *tail* of saved intermediate pebbles. We depict this dynamic computation in the right part of Figure 6, which shows partially completed subtrees and their associated tails.

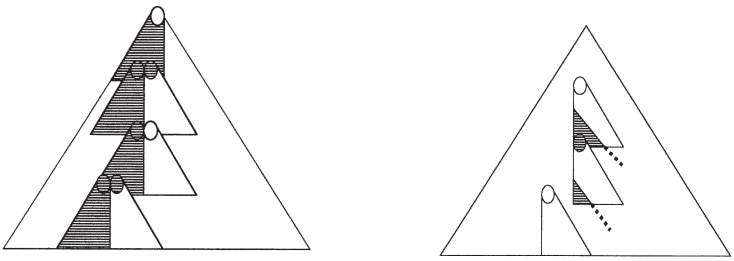

Fig. 6. *(Left)* The grey subtrees correspond to the *existing* subtrees (as in figure 5) while the white subtrees correspond to the *desired* subtrees. As the existing subtrees are used up, the desired subtrees are gradually constructed. *(Right)* The figure shows the set of desired subtrees from the previous figure, but with grey portions corresponding to nodes that have been computed and dotted lines corresponding to pebbles in the tail.

Algorithm Intuition

We now can present intuition for the main algorithm, and explain why the existing subtrees EXIST_i will always be available.

Overview. The goal of the traversal is to sequentially output authentication paths. By design, the existing subtrees should always contain the next authentication path to be output, while the desired subtrees contain more and more completed pebbles with each round, until the existing subtree expires.

When EXIST_i is used in an output for the last time, we say that it *dies*. At that time, the adjacent subtree, DESIRE_i will need to have been completed, i.e., have values assigned to all its nodes but its root (since the latter node is already part of the parent tree.) The tree EXIST_i is then *reincarnated* as DESIRE_i. First all the old pebbles of EXIST_i are discarded; then the pebbles of DESIRE_i (and their associated values) taken by EXIST_i. (Once this occurs, the computation of the new and adjacent subtree DESIRE_i will be initiated.) This way, if one can ensure that the pebbles on trees DESIRE_i are always computed on time, one can see that there will always be completed existing subtrees EXIST_i.

Modifying the treehash algorithm. As mentioned above, our tool used to compute the desired tree is a modified version of the classic treehash algorithm applied to the root of DESIRE_i. This version differs in that (1) it stops the algorithm one round earlier (thereby skipping the root calculation), and (2) every pebble of height greater than ih is saved into the tree DESIRE_i. For purposes of counting, we won't consider such saved pebbles as part of the *proper tail*.

Amortizing the computations. For a particular level i, we recall that the computational cost for tree DESIRE_i is $2 \cdot 2^{ih} - 2$, as we omit the calculation of the root. At the same time, we know that EXIST_i will serve for 2^{ih} output rounds. We amortize the computation of DESIRE_i over this period, by simply computing two iterations of treehash each round. In fact, DESIRE_i will be ready before it is needed, exactly 1 round in advance!

Thus, for each level, allocating 2 computational units ensures that the desired trees are completed on time. The total computation per round is thus $2(L-1)$.

Solution and Algorithm Presentation

Three phases. We now describe more precisely the main algorithm. There are three phases, the *key generation* phase; the *output* phase; and the *verification* phase. During the key generation phase (which may be performed offline by a relatively powerful computer), the root of the tree is computed and output, taking the role of a public key. Additionally, the iterative output phase needs some setup, namely the computation of pebbles on the initial *existing* subtrees. These are stored on the computer performing the output phase.

The *output* phase consists of a number of rounds. During round s, the authentication path of the sth leaf is output. In addition, some number of pebbles are discarded and some number of pebbles are computed, in order to prepare for future outputs.

The *verification* phase is identical to the traditional verification phase for Merkle trees and has been described above. We remark again that the outputs the algorithm generates will be indistinguishable from the outputs generated by a traditional algorithm. Therefore, we do not detail the verification phase, but merely the key generation phase and output phase.

Key Generation. First, the pebbles of the left-most set of stacked *existing* subtrees are computed and stored. Each associated pebble has a *value*, a *position*, and a *height*. In addition, a list of *desired* subtrees is created, one for each level $i < L$, each initialized with an empty stack for use in the modified treehash algorithm.

Recalling the indexing of the leaves, indexed by $s \in \{0, 1, \dots N-1\}$, we initialize a counter $\textsc{Desire}_i.\textit{position}$ to be 2^{ih}, indicating which Merkle tree leaf is to be computed next.

Algorithm 4.3 Key-Gen and Setup

1. **Initial Subtrees** For each $i \in \{1, 2, \dots L\}$:
   - Calculate all (non-root) pebbles in existing subtree at level i.
   - Create new empty desired subtree at each level i (except for $i = L$), with leaf *position* initialized to 2^{ih}.
2. **Public Key** Calculate and publish tree root.

Output and Update Phase. Each round of the execution phase consists of the following portions: *generating an output*, *death and reincarnation of existing subtrees*, and *growing desired subtrees*.

At round s, the output consists of the authentication path associated to the sth leaf. The pebbles for this authentication path will be contained in the existing subtrees.

When the last authentication path requiring pebbles from a given existing subtree has been output, then the subtree is no longer useful, and we say that it "*dies*." By then, the corresponding desired subtree has been completed, and the recently died existing subtree "*reincarnates*" as this completed desired subtree. Notice that a new subtree at level i is needed once every 2^{ih} rounds, and so once per 2^{ih} rounds the pebbles in the existing tree are discarded. More technically, at round s, $s = 0 \pmod{2^{ih}}$ the pebbles in the old tree $\textsc{Exist}_i$ are discarded; the completed tree $\textsc{Desire}_i$ becomes the new tree $\textsc{Exist}_i$; and a new, empty desired subtree is created.

In the last step we grow each desired subtree that is not yet completed a little bit. More specifically, we apply two computational units to the new or already started invocations of the treehash algorithm. We concisely present this algorithm as follows:

Time and Space Analysis

Time. As presented above, the algorithm allocates 2 computational units to each desired subtree. Here, a computational unit is defined to be either a call to Leafcalc, or the computation of a hash value. Since there are at most $L - 1$ desired subtrees, the total computational cost per round is

$$T_{\max} = 2(L-1) < 2H/h. \tag{23}$$

Algorithm 4.4 Stratified Merkle Tree Traversal

1. Set $s = 0$.
2. **Output** Authentication Path for leaf number s.
3. **Next Subtree** For each $i \in \{1, 2, \ldots L\}$ for which Exist_i is no longer needed, i.e, $s = 0 \pmod{2^{hi}}$:
   - Remove Pebbles in Exist_i.
   - Rename tree Desire_i as tree Exist_i.
   - Create new, empty tree Desire_i (if $s + 2^{hi} < 2^H$).
4. **Grow Subtrees** For each $i \in \{1, 2, \ldots h\}$: Grow tree Desire_i by applying 2 units to the modified treehash algorithm (unless Desire_i is completed).
5. Increment s and loop back to step 2 (while $s < 2^H$).

Space. The total amount of space required by the algorithm, or equivalently, the number of available pebbles required, may be bounded by simply counting the contributions from (1) the existing subtrees, (2) the desired subtrees, and (3) the tails.

First, there are L existing subtrees and up to $L - 1$ desired subtrees, and each of these contains up to $2^{h+1} - 2$ pebbles, since we do not store the roots. Additionally, the tail associated to a desired subtree at level $i > 1$ contains at most $h \cdot i + 1$ pebbles. If we count only the pebbles in the tail which do not belong to the desired subtree, then this "proper" tail contains at most $h(i - 1) + 1$ pebbles. Adding these contributions, we obtain the sum $(2L - 1)(2^{h+1} - 2) + h \sum_{i=1}^{L-2} i + 1$, and thus the bound:

$$\text{Space}_{\max} \leq (2L - 1)(2^{h+1} - 2) + L - 2 + h(L - 2)(L - 1)/2. \tag{24}$$

A marginally worse bound is simpler to write:

$$\text{Space}_{\max} < 2\,L\,2^{h+1} + H\,L\,/2. \tag{25}$$

Trade-offs. The solution just analyzed presents us with a trade-off between time and space. In general, the larger the subtrees are, the faster the algorithm will run, but the larger the space requirement will be. The parameter affecting the space and time in this trade-off is h; in terms of h the computational cost is below $2H/h$, the space required is bounded above by $2\,L\,2^{h+1} + H\,L/2$. Alternatively, and in terms of h, the space is bounded above by $2\,H\,2^{h+1}/h + H^2/2\,h$.

Low Space Solution. If one is interested in parameters requiring little space, there is an optimal h, due to the fact that for very small h, the number of tail pebbles increases significantly (when $H^2/2h$ becomes large). An approximation of this value is $h = \log H$. One could find the exact value by differentiating the expression for the space: $2\,H\,2^{h+1}/h + H^2/2\,h$. For this choice of $h = \log H = \log \log N$, we obtain

$$T_{\max} = \frac{2H}{\log H}. \tag{26}$$

$$\text{Space}_{\max} \leq \frac{5}{2} \cdot \frac{H^2}{\log H}. \tag{27}$$

These results are interesting because they asymptotically improve Merkle's result from Section 4.1 with respect to both space and time. Merkle's approach required $T_{\max} = 2H$ and $\text{Space}_{\max} \approx H^2/2$.

Additional Savings

We now return to the main algorithm, and explain how a small technical modification will improve the constants in the space bound, ultimately yielding the claimed result.

Although this modification does not affect the complexity *class* of either the space or time costs, it is of practical interest as it nearly halves the space bound in certain cases. It is presented after the main exposition in order to retain the original simplicity, as this analysis is slightly more technical. The modification is based on two observations: (1) There may be pebbles in existing subtrees which are no longer useful, and (2) The desired subtrees are always in a state of partial completion. In fact, we have found that pebbles in an existing subtree may be discarded *nearly* as fast as pebbles are entered into the corresponding desired subtree. The modifications are as follows:

1. Discard pebbles in the trees EXIST_i as soon as they will never again be required.
2. Omit the *first* application of 2 units to the modified treehash algorithm.

We note that with the second modification, the desired subtrees still complete, just in time. With these small changes, for all levels $i < L$, the number of pebbles contained in *both* EXIST_i, and DESIRE_i can be bounded by the following expression.

$$\text{Space}_{\text{EXIST}_i} + \text{Space}_{\text{DESIRE}_i} \leq 2^{ih+1} - 2 + (h-2). \tag{28}$$

This is nearly half of the previous bound of $2 \cdot (2^{ih+1} - 2)$. We remark here that the quantity $h - 2$ measures the maximum number of pebbles contained in DESIRE_i *exceeding* the number of pebbles contained in EXIST_i which have been discarded. Using the estimate (28), we revise the space bound computed in the previous section to be

$$\text{Space}_{\max} \leq (L)(2^{h+1} - 2) + (L-1)(h-2) + L - 2 + h(L-2)(L-1)/2. \tag{29}$$

We again round this up to obtain a simpler bound.

$$\text{Space}_{\max} < L\, 2^{h+1} H\, L\, /2. \tag{30}$$

Specializing to the choice $h = \log H$, we improve the above result to

$$\text{Space}_{\max} \leq \frac{3}{2} \cdot \frac{H^2}{\log H}. \tag{31}$$

by reducing the constant from $5/2$ to $3/2$.

Proof of Space Bound. Here we prove the assertion of Equation (28) which states for any level i the number of pebbles in the EXIST_i plus the number of pebbles in the DESIRE_i is less than $2 \cdot 2^{hi} - 2 + (h-2)$. This basic observation reflects the fact that the desired subtree can grow only slightly faster than the existing subtree shrinks. Without loss of generality, in order to simplify the exposition, we do not specify the subtree indices, and restrict our attention to the first existing-desired subtree pair at a given level i.

The first modification ensures that pebbles are returned more continuously than previously, so we quantify this. Subtree EXIST_i, has 2^h leaves, and as each leaf is no longer required, neither may be some interior nodes above it. These leaves are finished at rounds $2^{(i-1)h}a - 1$ for $a \in \{1, \ldots 2^h\}$. We may determine the number of pebbles returned at these times by observing that a leaf is returned every single round, a pebble at height $i\,h+1$ every two rounds, one at height $i\,h + 2$ every four rounds, etc. We are interested in the number returned at all times *up to* the time $2^{(i-1)h}a-1$; this is the sum of the greatest integer functions:

$$A + [A/2] + [A/4] + [A/8] + \ldots + [A/2^h]$$

Writing a in binary notation $a = a_0 + 2^1 a_1 + 2^2 a_2 + \ldots + 2^h a_h$, this sum is also

$$a_0(2^1 - 1) + a_1 \cdot (2^2 - 1) + a_2 \cdot (2^3 - 1) + \ldots + a_h(2^{h+1} - 1).$$

The cost to calculate the corresponding pebbles in DESIRE_i may also be calculated with a similar expression. Using the fact that a height h_0 node needs $2^{h_0+1} - 1$ units to compute, we see that the desired subtree requires

$$a_0(2^{(i-1)h+1} - 1) + a_1(2 \cdot 2^{(i-1)h+2} - 1) + \ldots + a_h(2 \cdot 2^{ih+1} - 1)$$

computational units to place those same pebbles. This cost is equal to $2 \cdot 2^{(i-1)h} \cdot a - z$, where z denotes the number of nonzero digits in the binary expansion of a.

At time $2^{(i-1)\,h}a - 1$, a total of $2 \cdot 2^{(i-1)\,h}a - 2$ units of computation has been applied to DESIRE_i, (factoring in our 1 round delay). Noting that $2^{(i-1)\,h} - 1$ more rounds may pass before EXIST_i loses any more pebbles, we see that the maximal number of pebbles during this interval must be realized at the very end of this interval. At this point in time, the desired subtree has computed exactly the pebbles that have been removed from the existing tree, plus whatever additional pebbles it can compute with its remaining $2 \cdot 2^{ih} - 2 + z - 2$ computational units. The next pebble, (a leaf) costs $2 \cdot 2^{ih} - 1$ which leaves $z - 3$ computational units. Even if all of these units result in new pebbles, the total extra is still less than or equal to $1 + z - 3$. Since $z \leq h$, this number of extra pebbles is bounded by $h - 2$, as claimed, and Equation (28) is proved.

4.3 Merkle Tree Traversal in Log Space and Time

Let us make some observations about the classic traversal algorithm from Section 4.1. We see that with the classic algorithm above, up to H instances of the treehash algorithm may be concurrently active, one for each height less than H. One can conceptualize them as H processes running in parallel, each requiring also a certain amount of space for the "tail nodes" of the treehash algorithm, and receiving a budget of two hash value computations per round, clearly enough to complete the $2^{h+1} - 1$ hash computations required over the 2^h available rounds.

Because the stack employed by treehash may contain up to $h + 1$ node values, we are only guaranteed a space bound of $1+2+\cdots+H$. The possibility of so many tail nodes is the source of the $\Omega(H^2/2)$ space complexity in the classic algorithm.

Considering that for the larger h, the treehash calculations have many rounds to complete, it appears that it might be wasteful to save so many intermediate nodes at once. Our idea is to schedule the concurrent treehash calculations differently, so that at any given round $s \in \{0, \ldots, 2^H - 1\}$, the associated stacks are mostly empty. We chose a schedule which generally favors computation of upcoming authentication nodes AUTH_h for lower h, (because they are required sooner), but delays beginning of a new instance of the treehash algorithm slightly, waiting until all stacks STACK_i are partially completed, containing no tail nodes of height less than h.

This delay, was motivated by the observation that in general, if the computation of two nodes at the same height in different treehash stacks are computed serially, rather than in parallel, less space will be used. Informally, we call the delay in starting new stack computations "zipping up the tails". We will need to prove the fact, which is no longer obvious, that the upcoming needed nodes will always be ready in time.

The New Traversal Algorithm

In this section we describe the new scheduling algorithm. Comparing to the classic traversal algorithm, the only difference will be in how the budget of $2H$ hash function evaluations will be allocated among the potentially H concurrent treehash processes.

Define $\text{STACK}_h.low$ to be the height of the lowest node in STACK_h, except in two cases: if the stack is empty $\text{STACK}_h.low$ is defined to be h, and if the treehash algorithm has completed $\text{STACK}_h.low$ is defined to be ∞.

Using the idea of zipping up the tails, there is more than one way to invent a scheduling algorithm which will take advantage of this savings. The one we present here is not optimal, but it is simple to describe. Additional practical improvements are discussed in Section 4.5.

This version can be concisely described as follows. The upcoming needed authentication nodes are computed as in the classic traversal, but the various

Algorithm 4.5 Logarithmic Merkle Tree Traversal

1. Set $s = 0$.
2. **Output:**
- For each $h \in [0, H-1]$ output AUTH_h.
3. **Refresh Auth Nodes:**
   For all h such that 2^h divides $s+1$:
- Set AUTH_h be the sole node value in STACK_h.
- Set startnode $= (s + 1 + 2^h) \oplus 2^h$.
- $\text{STACK}_h.\textit{initialize}(\text{startnode}, h)$.
4. **Build Stacks:**
   Repeat the following $2H - 1$ times:
- Let $l_{\min}$ be the minimum of $\text{STACK}_h.\textit{low}$.
- Let *focus* be the least h so $\text{STACK}_h.\textit{low} = l_{\min}$.
- $\text{STACK}_{focus}.\textit{update}$.
5. **Loop:**
- Set $s = s + 1$.
- If $s < 2^H$ go to Step 2.

stacks do not all receive equal attention. Each treehash instance can be characterized as being either not started, partially completed, or completed.

Our schedule prefers to complete STACK_h for the lowest h values first, *unless another stack has a lower tail node*. We express this preference by defining $l_{\min}$ be the minimum of the h values $\text{STACK}_h.\textit{low}$, then choosing to focus our attention on the smallest level h attaining this minimum. (setting $\text{STACK}_h.\textit{low} = \infty$ for completed stacks effectively skips them over).

In other words, all stacks must be completed to a stage where there are no tail nodes at height h or less before we start a new STACK_h treehash computation. The final algorithm is summarized in Algorithm 4.5.

Correctness and Analysis

In this section we show that our computational budget of $2H - 1$ is indeed sufficient to complete every STACK_h computation before it is required as an authentication node. We also show that the space required for hash values is less than $3H$.

Nodes are Computed on Time. As presented above, the algorithm allocates exactly a budget of $2H - 1$ computational units per round to spend updating the h stacks. Here, a computational unit is defined to be either a call to LEAFCALC, or the computation of a hash value. We do not model any extra expense due to complex leaf calculations.

To prove this, we focus on a given height h, and consider the period starting from the time STACK_h is created and ending at the time when the upcoming authentication node (denoted NEED_h here) is required to be completed. This is not immediately clear, due to the complicated scheduling algorithm. Our

approach to prove that NEED_h is completed on time is to showing that the total budget over this period exceeds the cost of *all* nodes computed within this period which can be computed before NEED_h.

The node NEED_h itself costs only $2^{h+1}-1$ units, a tractable amount given that there are 2^h rounds between the time STACK_h is created, and the time by which NEED_h must be completed. However, a non trivial calculation is required, since in addition to the resources required by NEED_h, many other nodes compete for the total budget of $2H2^h$ computational units available in this period. These nodes include all the future needed nodes NEED_i, $(i < h)$, for lower levels. Finally there may be a partial contribution to a node $\text{NEED}_i, i > h$, so that its stack contains no low nodes by the time NEED_h is computed.

It is easy to count the number of such needed nodes in the interval, and we know the cost of each one. As for the contributions to higher stacks, we at least know that the cost to raise any low node to height h must be less than $2^{h+1}-1$ (the total cost of a height h node). We summarize these quantities and costs in the following figure.

Table 1. Nodes built during 2^h rounds for NEED_h.

Node Type	Quantity	Cost each
NEED_h	1	$2^{h+1}-1$
NEED_{h-1}	2	2^h-1
$\vdots$	$\vdots$	$\vdots$
NEED_k	2^{h-k}	$2^{k+1}-1$
$\vdots$	$\vdots$	$\vdots$
NEED_0	2^h	1
TAIL	1	$\leq 2^{h+1}-2$

We proceed to tally up the total cost incurred during the interval. Notice that the row beginning NEED_0 requires a total of 2^{h+1} computational units. For every other row in the node chart, the number of nodes of a given type multiplied by the cost per node is less than 2^{h+1}. There are $h+1$ such rows, so the total cost of all nodes represented in the chart is

$$\text{TotalCost}_h < (h+2)2^h. \tag{32}$$

For heights $h \leq H-2$, it is clear that this total cost is less than $2H2^H$. It is also true for the remaining case of $h = H-1$, because there are no tail nodes in this case.

We conclude that, as claimed, the budget of $2H-1$ units per round is indeed always sufficient to prepare NEED_h on time, for any $0 \leq h < H$.

Space is Bounded by 3H. Our motivation leading to this relatively complex scheduling is to use as little space as possible. To prove this, we simply add up the quantities of each kind of node. We know there are always H nodes AUTH_h. Let $C < H$ be the number of completed nodes NEED_h.

$$\#\text{AUTH}_i + \#\text{NEED}_i = H + C. \tag{33}$$

We must finally consider the number of tail nodes in the STACK_h. As for these, we observe that since a STACK_h never becomes active until all nodes in "higher" stacks are of height at least h, there can never be two distinct stacks, each containing a node of the same height. Furthermore, recalling algorithm treehash, we know there is at most one height for which a stack has two node values. In all, there is at most one tail node at each height ($0 \leq h \leq H - 3$), plus up to one additional tail node per non-completed stack. Thus

$$\#\text{TAIL} \leq H - 2 + (H - C). \tag{34}$$

Adding all types of nodes we obtain:

$$\#\text{AUTH}_i + \#\text{NEED}_i + \#\text{TAIL} \leq 3H - 2. \tag{35}$$

This proves the assertion. There are at most $3H - 2$ stored nodes.

4.4 Asymptotic Optimality Result

An interesting optimality result states that a traversal algorithm can never beat both time $O(\log(N))$ and space $O(\log(N))$. It is clear that at least $H-2$ nodes are required for the treehash algorithm, so our task is essentially to show that if space is limited by any constant multiple of $\log(N)$, then the computational complexity must be $\Omega(\log(N))$. Let us be clear that this theorem does not quantify the constants. Clearly, with greater space, computation time can be reduced.

Theorem 1. *Suppose that there is a Merkle tree traversal algorithm for which the space is bounded by* $\alpha \log(N)$. *Then there exists some constant* β *so that the time required is at least* $\beta \log(N)$.

The theorem simply states that it is not possible to reduce space complexity below logarithmic without increasing the time complexity beyond logarithmic!

The proof of this technical statement is found in the upcoming subsection, but we will briefly describe the approach here. We consider only right nodes for the proof. We divide all right nodes into two groups: those which must be computed (at a cost of $2^{h+1} - 1$), and those which have been saved from some earlier calculation. The proof assumes a sub-logarithmic time complexity and derives a contradiction.

The more nodes in the second category, the faster the traversal can go. However, such a large quantity of nodes would be required to be saved in order

to reduce the time complexity to sub-logarithmic, that the average number of saved node values would have to exceed a linear amount! The rather technical proof presented next uses a certain sequence of subtrees to formulate the contradiction.

We now begin the technical proof of Theorem 1. This will be a proof by contradiction. We assume that the time complexity is sub logarithmic, and show that this is incompatible with the assumption that the space complexity is $O(\log(N))$. Our strategy to produce a contradiction is to find a bound on some linear combination of the average time and the average amount of space consumed.

Notation. The theorem is an asymptotic statement, so we will be considering trees of height $H = \log(N)$, for large H. We need to consider L *levels* of subtrees of height k, where $kL = H$. Within the main tree, the roots of these subtrees will be at heights $k, 2 \cdot k, 3 \cdot k \ldots H$. We say that the subtree is at *level* i if its root is at height $(i+1)k$. This subtree notation is similar to that used in Section 4.2.

Note that we will only need to consider right nodes to complete our argument. Recall that during a complete tree traversal every single right node is eventually output as part of the authentication data. This prompts us to categorize the right nodes in three classes.

1. Those already present after the key generation: *free nodes.*
2. Those explicitly calculated (e.g. with treehash): *computed nodes.*
3. Those retained from another node's calculation (e.g from another node's treehash): *saved nodes.*

Notice how type 2 nodes require computational effort, whereas type 1 and type 3 nodes require some period of storage. We need further notation to conveniently reason about these nodes. Let a_i denote the number of level i subtrees which contain *at least 1* non-root computed (right) node. Similarly, let b_i denote the number of level i subtrees which contain *zero* computed nodes. Just by counting the total number of level i subtrees we have the relation.

$$a_i + b_i = N/2^{(i+1)k}. \tag{36}$$

Computational costs. Let us tally the cost of some of the computed nodes. There are a_i subtrees containing a node of type 2, which must be of height at least ik. Each such node will cost at least $2^{ik+1} - 1$ operations to compute. Rounding down, we find a simple lower bound for the cost of the nodes at level i.

$$Cost > \sum_{i=0}^{L-1} (a_i 2^{ik}). \tag{37}$$

Storage costs. Let us tally the lifespans of some of the retained nodes. Measuring units of *Space* $\times$ *Rounds* is natural when considering average space consumed. In general, a saved node, S, results from a calculation of some

computed node C, say, located at height h. We know that S has been produced before C is even needed, and S will never become an authentication node before C is discarded. We conclude that such a node S must therefore be stored in memory for at least 2^h rounds.

Even (most of) the free nodes at height h remain in memory for at least 2^{h+1} rounds. In fact, there can be at most one exception: the first right node at level h.

Now consider one of the b_i subtrees at level i containing only free or stored nodes. Except for the leftmost subtree at each level, which may contain a free node waiting in memory less than $2^{(i+1)k}$ rounds, every other node in this subtree takes up space for at least $2^{(i+1)k}$ rounds. There are $2^k - 1$ nodes in a subtree and thus we find a simple lower bound on the $Space \times Rounds$.

$$Space \times Rounds \geq \sum_{i=0}^{L-1} (b_i - 1)(2^k - 1)2^{(i+1)k}. \tag{38}$$

Note that the $(b_i - 1)$ term reflects the possible omission of the leftmost level i subtree.

Mixed Bounds. We can now use simple algebra with Equations (36), (37), and (38) to yield combined bounds. First the cost is related to the b_i, which is then related to a space bound.

$$2^k Cost > \sum_{i=0}^{L-1} a_i 2^{(i+1)k} = \sum_{i=0}^{L-1} N - 2^{(i+1)k} b_i. \tag{39}$$

As series of similar algebraic manipulations finally yield (somewhat weaker) very useful bounds.

$$2^k Cost + \sum_{i=0}^{L-1} 2^{(i+1)k} b_i > NL. \tag{40}$$

$$2^k Cost + \sum_{i=0}^{L-1} \frac{2^{(i+1)k}}{2^{k-1}} + \frac{Space \times Rounds}{2^{k-1}} > NL \tag{41}$$

$$2^k Cost + 2N + \frac{Space \times Rounds}{2^{k-1}} > NL \tag{42}$$

$$2^k AverageCost + \frac{AverageSpace}{2^{k-1}} > (L - 2) \geq \frac{L}{2} \tag{43}$$

$$k2^{k+1} AverageCost + \frac{k}{2^{k-2}} AverageSpace > \frac{L}{2} \cdot 2k = H. \tag{44}$$

This last bound on the sum of average cost and space requirements will allow us to find a contradiction.

Proof by Contradiction. Let us assume the opposite of the statement of Theorem 1. Then there is some α such that the space is bounded above by $\alpha \log(N)$. Secondly, the time complexity is supposed to be sub-logarithmic, so for every small β the time required is less than $\beta \log(N)$ for sufficiently large N.

With these assumptions we are now able to choose a useful value of k. We pick k to be large enough so that $\alpha > 1/k2^{k+3}$. We also choose β to be less than $1/k2^{k+2}$. With these choices we obtain two relations.

$$k2^{k+1} AverageCost < \frac{H}{2} \tag{45}$$

$$k/2^{k-2} AverageSpace < \frac{H}{2} \tag{46}$$

By adding these two last equations, we contradict Equation (44).
QED.

4.5 Improvement of the Log Traversal Algorithm

In this section we describe improvements of the algorithm described in Section 4.3 which are very useful for practical implementations. The main differences are the following. Since left authentication nodes can be computed much cheaper than right nodes, the computation of left and right authentication nodes is done differently. In many cases the number of expensive leaf computations is reduced. Instead of using a separate stack for each instance of the treehash algorithm one shared stack is used. Input for the algorithm is an index $s \in \{0, 1, \ldots, 2^H - 2\}$. The algorithm determines the authentication path $\text{AUTH} = (\text{AUTH}_0, \ldots, \text{AUTH}_{H-1})$ for leaf $s + 1$.

As before, we denote the nodes in the Merkle tree by $\nu_h[j]$, where $h = H, \ldots, 0$ denotes the height of the node in the tree of height H. Leaves have height 0 and the root has height H. MSS uses a cryptographic hash function $g : \{0,1\}^* \to \{0,1\}^n$.

The algorithm determines $\tau = \max\{h : 2^h | (s+1)\}$ which is the height of the first ancestor of the sth leaf which is a left child. If leaf s is a left child itself, then $\tau = 0$. Figure 7 shows an example.

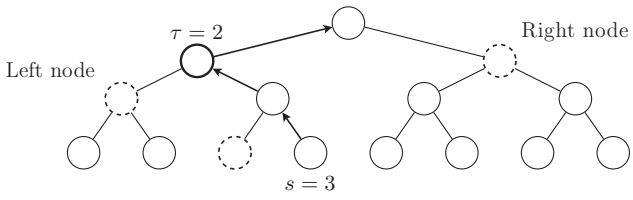

Fig. 7. The height of the first ancestor of leaf s that is a left child is $\tau = 2$. The dashed nodes denote the authentication path for leaf s. The arrows indicate the path from leaf s to the root.

The value τ is used to determine on which heights the authentication path for leaf $s+1$ requires new nodes. The authentication path for leaf $s+1$ requires new right authentication nodes on heights $h = 0, \ldots, \tau - 1$ and one new left authentication node on height τ.

Computing left and right authentication nodes

Computing left nodes. As explained above, we require the left node AUTH_τ for the next authentication path. If $\tau = 0$, then we set AUTH_0 to $\text{LEAFCALC}(s)$. Let $\tau > 0$. Then leaf s is a right child. Also, $\text{AUTH}_{\tau-1}$ is the left child of AUTH_τ. We assume that the right child of AUTH_τ is stored in $\text{KEEP}_{\tau-1}$. Then the new node AUTH_τ is computed as

$$\text{AUTH}_\tau = g\big(\text{AUTH}_{\tau-1} \| \text{KEEP}_{\tau-1}\big). \tag{47}$$

This requires only one hash evaluation. We also explain how KEEP is updated. If $\lfloor s/2^{\tau+1} \rfloor = 0 \pmod 2$, i.e. if the ancestor on height $\tau + 1$ is a left child, then AUTH_τ is a right node and we store it in KEEP_τ.

Computing right nodes. Unlike authentication nodes that are left children, right authentication nodes are computed from scratch, i.e. starting from the leaves. This is because none of their child nodes were used in previous authentication paths. As before we use the treehash algorithm (Section 2, Algorithm 2.1) for this task.

We use two different methods for computing right nodes. To distinguish those cases we select a positive integer $K \geq 2$ such that $H-K$ is even. Suppose that we wish to compute a right node on height h. If $H - K \leq h \leq H - 2$, then the right node on height h is calculated by RETAIN_h.pop() which pops the top element from a stack RETAIN_h. That stack has been filled with the right nodes $\nu_h[3], \ldots, \nu_h[2^{H-h} - 1]$ during MSS key generation. This is very useful since the nodes close to the root are expensive to compute.

For the computation of a right node on height h with $h < H - K$ we use an instance TREEHASH_h of the treehash algorithm. It is allowed to store one node. Initially, during MSS key generation, the second right node $\nu_h[3]$ is stored in TREEHASH_h. The treehash instances all share one stack. When it comes to determining a right authentication node on height h this is simply done by TREEHASH_h.pop() for $h = 0, \ldots, \min\{H - K - 1, \tau - 1\}$. Then all treehash instances for heights $h = 0, \ldots, \min\{H-K-1, \tau-1\}$ are initialized for the computation of the next right node. The index of the leaf they have to begin with is $s + 1 + 3 \cdot 2^h$ and the initialization is done using the method TREEHASH_h.initialize($s+1+3\cdot 2^h$). Then the algorithm updates the treehash instances using the TREEHASH_h.update() method. One update corresponds to one round of Algorithm 2.1, i.e. to computing one leaf and computing this leaf's parent nodes using tail nodes stored on the stack.

We allow a budget of $(H - K)/2$ updates in each round. We use the strategy from Section 4.3 to decide which of the $H - K$ treehash instances

receives an update. For this, we need the method TREEHASH_h.height() which returns the height of the lowest tail node stored by this treehash instance, either on the stack or in the treehash instance itself. If TREEHASH_h does not store any tail nodes TREEHASH_h.height() returns h and if TREEHASH_h is finished or not initialized TREEHASH_h.height() returns ∞ to skip these instances. The treehash instance that receives an update is the instance where TREEHASH_h.height() returns the smallest value. If there is more than one such instance, we choose the one with the lowest index.

The algorithm

Initialization. The initialization of our algorithm is done during the MSS key pair generation. We store the authentication path for the first leaf ($s = 0$): $\text{AUTH}_h = \nu_h[1], h = 0, \ldots, H-1$. Depending on the parameter K, we store the next right authentication node for each height $h = 0, \ldots, H-K-1$ in the treehash instances: TREEHASH_h.push($\nu_h[3]$). Finally we store the right authentication nodes close to the root using the stacks RETAIN_h: RETAIN_h.push($\nu_h[2j+3]$) for $h = H-K, \ldots, H-2$ and $j = 2^{H-h-1}-2, \ldots, 0$.

Update and output phase. Algorithm 4.6 contains the precise description. Input is the index of the current leaf $s \in \{0, \ldots, 2^H - 2\}$, the parameters H, K and the algorithm state AUTH, KEEP, RETAIN, TREEHASH prepared in previous rounds or the initialization. Output is the authentication path for the next leaf $s + 1$.

Correctness and analysis

In this section we show the correctness of Algorithm 4.6 and estimate its time and space requirements. First we show that the budget of $(H-K)/2$ updates per round is sufficient for the treehash instances to compute the nodes on time. Then we show that it is possible for all treehash instances to share a single stack. Next, we consider the time and space requirements of Algorithm 4.6. In detail we show that

i) The number of tail nodes stored on the stack is bounded by $H - K - 2$.
ii) The number of hashes per round is bounded by $3(H - K - 1)/2$.
iii) The number of nodes stored in KEEP is bounded by $\lfloor H/2 \rfloor + 1$.

To estimate the space complexity, we have to add the H nodes stored in AUTH, the $H - K$ nodes stored in TREEHASH and the $2^K - K - 1$ nodes stored in RETAIN. To estimate the time complexity, we have to add the $(H-K)/2$ leaf computations required to determine right nodes and one leaf and one hash to compute left nodes (Lines 3, 4a in Algorithm 4.6). Summing up the total time and space requirements results in the following theorem.

Algorithm 4.6 Authentication path computation

Input: $s \in \{0, \ldots, 2^H - 2\}$, H, K and the algorithm state.
Output: Authentication path for leaf $s + 1$

1. Let $\tau = 0$ if leaf s is a left node or let τ be the height of the first parent of leaf s which is a left node:
   $\tau \leftarrow \max\{h : 2^h | (s + 1)\}$
2. If the parent of leaf s on height $\tau + 1$ is a left node, store the current authentication node on height τ in KEEP_τ:
   if $\lfloor s/2^{\tau+1} \rfloor$ is even **and** $\tau < H - 1$ **then** $\text{KEEP}_\tau \leftarrow \text{AUTH}_\tau$
3. If leaf s is a left node, it is required for the authentication path of leaf $s + 1$:
   if $\tau = 0$ **then** $\text{AUTH}_0 \leftarrow \text{LEAFCALC}(s)$
4. Otherwise, if leaf s is a right node, the authentication path for leaf $s+1$ changes on heights $0, \ldots, \tau$:
   if $\tau > 0$ **then**
   a) The authentication path for leaf $s + 1$ requires a new left node on height τ. It is computed using the current authentication node on height $\tau - 1$ and the node on height $\tau - 1$ previously stored in $\text{KEEP}_{\tau-1}$. The node stored in $\text{KEEP}_{\tau-1}$ can then be removed:
      $\text{AUTH}_\tau \leftarrow g(\text{AUTH}_{\tau-1} || \text{KEEP}_{\tau-1})$, remove $\text{KEEP}_{\tau-1}$
   b) The authentication path for leaf $s + 1$ requires new right nodes on heights $h = 0, \ldots, \tau - 1$. For $h < H - K$ these nodes are stored in TREEHASH_h and for $h \geq H - K$ in RETAIN_h:
      for $h = 0$ **to** $\tau - 1$ **do**
         if $h < H - K$ **then** $\text{AUTH}_h \leftarrow \text{TREEHASH}_h.\text{pop}()$
         if $h \geq H - K$ **then** $\text{AUTH}_h \leftarrow \text{RETAIN}_h.\text{pop}()$
   c) For heights $0, \ldots, \min\{\tau - 1, H - K - 1\}$ the treehash instances must be initialized anew. The treehash instance on height h is initialized with the start index $s + 1 + 3 \cdot 2^h < 2^H$:
      for $h = 0$ **to** $\min\{\tau - 1, H - K - 1\}$ **do** $\text{TREEHASH}_h.\text{initialize}(s + 1 + 3 \cdot 2^h)$
5. Next we spend the budget of $(H - K)/2$ updates on the treehash instances to prepare upcoming authentication nodes:
   repeat $(H - K)/2$ **times**
   a) We consider only stacks which are initialized and not finished. Let k be the index of the treehash instance whose lowest tail node has the lowest height. In case there is more than one such instance we choose the instance with the lowest index:
      $k \leftarrow \min \left\{ h : \text{TREEHASH}_h.\text{height}() = \min_{j=0,\ldots,H-K-1} \{\text{TREEHASH}_j.\text{height}()\} \right\}$
   b) The treehash instance with index k receives one update:
      $\text{TREEHASH}_k.\text{update}()$
6. The last step is to output the authentication path for leaf $s + 1$:
   return $\text{AUTH}_0, \ldots, \text{AUTH}_{H-1}$.

Theorem 2. *Let $H \geq 2$ and $K \geq 2$ such that $H - K$ is even. Algorithm 4.6 stores at most $3H + \lfloor H/2 \rfloor - 3K - 2 + 2^K$ nodes, where each node requires n bits of memory. Further, the algorithm requires at most $(H - K)/2 + 1$ leaf computations and $3(H - K - 1)/2 + 1$ hash function evaluations per round to successively compute authentication paths.*

Nodes are computed on time. If TREEHASH_h is initialized in round s, the authentication node on height h computed by this instance is required in round $s + 2^{h+1}$. In these 2^{h+1} rounds there are $(H - K)2^h$ updates available. TREEHASH_h requires 2^h updates. During the 2^{h+1} rounds, $2^{h+1}/2^{i+1}$ treehash instances are initialized on heights $i = 0, \ldots, h-1$, each requiring 2^i updates. In addition, active treehash instances on heights $i = h+1, \ldots, H-K-1$ might receive updates until their lowest tail node has height h, thus requiring at most 2^h updates.

Summing up the number of updates required by all treehash instances yields

$$\sum_{i=0}^{h-1} \frac{2^{h+1}}{2^{i+1}} \cdot 2^i + 2^h + \sum_{i=h+1}^{H-K-1} 2^h = (H - K)2^h \tag{48}$$

as an upper bound for the number of updates required to finish TREEHASH_h on time. For $h = H - K - 1$ this bound is tight.

Sharing a single stack works. To show that it is possible for all treehash instances to share a single stack, we have to show that if TREEHASH_h receives an update and has tail nodes stored on the stack, all these tail nodes are on top of the stack.

When TREEHASH_h receives its first update, the height of the lowest tail node of TREEHASH_i, $i \in \{h+1, \ldots, H-K-1\}$ is at least h. This means that TREEHASH_h is completed before TREEHASH_i receives another update and thus tail nodes of higher treehash instances do not interfere with tail nodes of TREEHASH_h.

While TREEHASH_h is active and stores tail nodes on the stack, it is possible that treehash instances on lower heights $i \in \{0, \ldots, h-1\}$ receive updates and store nodes on the stack. If TREEHASH_i receives an update, the height of the lowest tail node of TREEHASH_h has height $\geq i$. This implies that TREEHASH_i is completed before TREEHASH_h receives another update and therefore doesn't store any tail nodes on the stack.

Space required by the stack. We will show that the stack stores at most one tail node on each height $h = 0, \ldots, H-K-3$ at a time. TREEHASH_h, $h \in \{0, \ldots, H-K-1\}$ stores up to h tail nodes on different heights to compute the authentication node on height h. The tail node on height $h-1$ is stored by the treehash instance and the remaining tail nodes on heights $0, \ldots, h-2$ are stored on the stack. When TREEHASH_h receives its first update, the following two conditions hold: (1) all treehash instances on heights $< h$ are either empty or completed and store no tail nodes on the stack. (2) All treehash instances

on heights $> h$ are either empty or completed or have tail nodes of height at least h. If a treehash instance on height $i \in \{h+1, \ldots, H-K-1\}$ stores a tail node on the stack, then all treehash instances on heights $i+1, \ldots, H-K-1$ have tail nodes of height at least i, otherwise the treehash instance on height i wouldn't have received any updates in the first place. This shows that there is at most one tail node on each height $h = 0, \ldots, H-K-3$ which bounds the number of nodes stored on the stack by $H-K-2$. This bound is tight for round $s = 2^{H-K+1} - 2$, before the update that completes the treehash instance on height $H-K-1$.

Number of hashes required per round. For now we assume that the maximum number of hash function evaluations is required in the following case: $\textsc{Treehash}_{H-K-1}$ receives all $u = (H-K)/2$ updates and is completed in this round. On input an index s, the number of hashes required by the treehash algorithm is equal to the height of the first parent of leaf s which is a left node. On height h, a left node occurs every 2^h leaves, which means that every 2^h updates at least h hashes are required by treehash. During the u available updates, there are $\lceil u/2^h \rceil$ updates that require at least h hashes for $h = 1, \ldots, \lceil \log_2 u \rceil$. The last update requires $H-K-1 = 2u-1$ hashes to complete the treehash instance on height $H-K-1$. So far only $\lceil \log_2 u \rceil$ of these hashes were counted, so we have to add another $2u-1-\lceil \log_2 u \rceil$ hashes. In total, we get the following upper bound for the number of hashes required per round.

$$B = \sum_{h=1}^{\lceil \log_2 u \rceil} \left\lceil \frac{u}{2^h} \right\rceil + 2u - 1 - \lceil \log_2 u \rceil \tag{49}$$

In round $s = 2^{H-K+1} - 2$ this bound is tight. This is the last round before the treehash instance on height $H-K-1$ must be completed and as explained above, all available updates are required in this case. The desired upper bound is estimated as follows:

$$\begin{aligned} B &\leq \sum_{h=1}^{\lceil \log_2 u \rceil} \left(\frac{u}{2^h} + 1 \right) + 2u - 1 - \lceil \log_2 u \rceil \\ &= u \sum_{h=1}^{\lceil \log_2 u \rceil} \frac{1}{2^h} + 2u - 1 = u \left(1 - \frac{1}{2^{\lceil \log_2 u \rceil}} \right) + 2u - 1 \\ &\leq u \left(1 - \frac{1}{2u} \right) + 2u - 1 = 3u - \frac{3}{2} = \frac{3}{2}(H-K-1) \end{aligned}$$

The next step is to show that the above mentioned case is indeed the worst case. If a treehash instance on height $< H-K-1$ receives all updates and is completed in this round, less than B hashes are required. The same holds if the treehash instance receives all updates but is not completed in this round. The last case to consider is the one where the u available updates are spend on treehash instances on different heights. If the active treehash instance has a tail

node on height j, it will receive updates until it has a tail node on height $j+1$, which requires 2^j updates and 2^j hashes. Additional $t \in \{1, \ldots, H-K-j-2\}$ hashes are required to compute the parent of this node on height $j+t+1$, if the active treehash instance stores tail nodes on heights $j+1, \ldots, j+t$ on the stack and in the treehash instance itself. The next treehash instance that receives updates has a tail node of height $\geq j$. Since the stack stores at most one tail node for each height, this instance can receive additional hashes only if there are enough updates to compute a tail node on height $\geq j+t$, the height of the next tail node possibly stored on the stack. But this is the same scenario that appears in the above mentioned worst case, i.e. if a node on height $j+1$ is computed, the tail nodes on the stack are used to compute its parent on height $j+t+1$ and the same instance receives the next update.

Space required to compute left nodes. First we show that whenever an authentication node is stored in KEEP_h, $h = 1, \ldots, H-2$, the node stored in KEEP_{h-1} is removed in the same round. This immediately follows from Steps 2 and 4a in Algorithm 4.6. Second we show that if a node gets stored in KEEP_h, $h = 0, \ldots, H-3$, then KEEP_{h+1} is empty. To see this we have to consider in which rounds a node is stored in KEEP_{h+1}. This is true for rounds $s \in A_a = \{2^{h+1} - 1 + a \cdot 2^{h+3}, \ldots, 2^{h+2} - 1 + a \cdot 2^{h+3}\}, a \in \mathbb{N}_0$. In rounds $s' = 2^h - 1 + b \cdot 2^{h+2}, b \in \mathbb{N}_0$, a node gets stored in KEEP_h. It is straight forward to compute that $s' \in A_a$ implies that $2a + 1/4 \leq b \leq 2a + 3/4$ which is a contradiction to $b \in \mathbb{N}_0$.

As a result, at most $\lfloor H/2 \rfloor$ nodes are stored in KEEP at a time and two consecutive nodes can share one entry. One additional entry is required to temporarily store the authentication node on height h (Step 2) until node on height $h-1$ is removed (Step 4a).

Computing leaves using an PRNG

In Section 3, we showed how a PRNG can be used during MSS key pair and signature generation to reduce the private key size. We will now show how to use this concept in Algorithm 4.6 to compute the required leaves using an PRNG. Let SEED_s denote the seed required to compute the one-time key pair corresponding to the sth leaf.

During the authentication path computation, leaves which are up to $3 \cdot 2^{H-K-1}$ steps away from the current leaf must be computed by the treehash instances. Calling the PRNG that many times to obtain the seed required to compute this leaf is too inefficient. Instead we use the following scheduling strategy that requires $H-K$ calls to the PRNG in each round to compute the seeds. We have to store two seeds for each height $h = 0, \ldots, H-K-1$. The first (SEEDACTIVE) is used to successively compute the leaves for the authentication node currently constructed by TREEHASH_h and the second (SEEDNEXT) is used for upcoming right nodes on this height. SEEDNEXT is updated using the PRNG in each round. During the initialization, we set

$\text{SEEDNEXT}_h = \text{SEED}_{3 \cdot 2^h}$ for $h = 0, \ldots, H - K - 1$. In each round, at first all seeds SEEDNEXT_h are updated using the PRNG. If in round s a new treehash instance is initialized on height h, we copy SEEDNEXT_h to SEEDACTIVE_h. In that case $\text{SEEDNEXT}_h = \text{SEED}_{\varphi+1+3 \cdot 2^h}$ holds and thus is the correct seed to begin computing the next authentication node on height h.

The time and space requirements of Algorithm 4.6 change as follows. We have to store additional $2(H-K)$ seeds and each seed requires n bit of memory. We also require additional $H - K$ calls to the PRNG in each round.

Theorem 3. *Let $H \geq 2$ and $K \geq 2$ such that $H - K$ is even. The memory requirements of Algorithm 4.6 in combination with a PRNG are*

$$\left(5H + \left\lfloor \frac{H}{2} \right\rfloor - 5K - 2 + 2^K\right) \cdot n \text{ bit.} \tag{50}$$

Further, it requires at most $(H - K)/2 + 1$ leaf computations, $3(H - K - 1)/2 + 1$ hash function evaluations, and $H - K$ calls to the PRNG per round to successively compute authentication paths.

5 Tree chaining

In Section 2 we saw that MSS public key generation requires the computation of the full Merkle hash tree. This means that 2^H leaves and $2^H - 1$ inner nodes have to be determined, which is very time consuming when H is large. The *tree chaining* method [4] solves this problem. The basic idea is similar to the Fractal Merkle Tree Traversal described in Section 4.2. However, in contrast to the Fractal Tree Traversal Method, tree chaining does not split the Merkle tree into smaller subtrees, but instead uses smaller Merkle trees that are independent of each other. The Merkle signature scheme that uses tree chaining is referred to as CMSS.

5.1 The idea

We explain the tree chaining idea. CMSS uses $T \geq 2$ *layers* of Merkle trees. Each Merkle tree on each layer is constructed using the Method from Sections 2 and 3. The hashes of a sequence of one-time verification keys are the leafs. We call the corresponding one-time signature keys the *signature keys of the Merkle tree*. Those signature keys are calculated using a pseudo random number generator. We call the respective seed the *seed of the Merke tree*.

The root of the single tree on the top layer 1 is the public CMSS key. The signature keys of the Merkle trees on the bottom layer T are used to sign documents. The signature keys of the Merkle trees on the intermediate layers i, $1 \leq i < T$ sign the roots of the Merkle trees on layer $i + 1$.

This is what a tree chaining signature looks like:

$$\begin{aligned}\sigma = \big(s,\ &\mathrm{SIG}_T, Y_T, \mathrm{AUTH}_T \\ &\mathrm{SIG}_{T-1}, Y_{T-1}, \mathrm{AUTH}_{T-1} \\ &\qquad\vdots \\ &\mathrm{SIG}_1, Y_1, \mathrm{AUTH}_1\big).\end{aligned} \tag{51}$$

SIG_T is the one-time signature of the document to be signed. It is generated using a signature key of a Merkle tree on the bottom layer T. The corresponding verification key is Y_T. Also, AUTH_T is the authentication path that allows a verifier to construct the path from the verification key Y_T to the root of the corresponding Merkle tree on the bottom layer. Now that root is not known to the verifier. Therefore, the one-time signature SIG_{T-1} of that root is also included in the signature σ. It is constructed using a signature key of a Merkle tree on level $T-1$. The corresponding verification key Y_{T-1} and authentication path AUTH_{T-1} are also included in the signature σ. The root of the tree on layer $T-1$ is also not known to the verifier, unless $T=2$ in which case $T-1=1$ and that root is the public key. So further one-time signatures of roots SIG_i, one-time verification keys Y_i, and authentication paths AUTH_i, $i = T-1, \ldots, 1$ are included in the signature σ.

The signature σ is verified as follows. The verifier checks, that SIG_T can be verified using Y_T. Next, he uses Y_T and AUTH_T to construct the root of a Merkle tree on layer T. He verifies the signature SIG_{T-1} of that root using the verification key Y_{T-1} and constructs the root of the corresponding Merkle tree on layer $T-1$ from Y_{T-1} and AUTH_{T-1}. The verifier iterates this procedure until the root of the single tree on layer 1 is constructed. The signature is verified by comparing this root to the public key. If any of those comparisons fails then the signature σ is rejected. Otherwise, it is accepted.

We discuss the advantage of the tree chaining method. For this purpose, we first compute the number of signatures that can be verified using one public key when the tree chaining method is applied. All Merkle trees on layer i have the same height H_i, $1 \le i \le T$. As mentioned already, there is a single Merkle tree on the top layer 1. Since the Merkle trees on layer i are used to sign the roots of the Merkle trees on layer $i+1$, $1 \le i < T$, the number of Merkle trees on layer $i+1$ is $2^{H_1+H_2+\ldots+H_i}$. So the total number of documents that can be signed/verified is 2^H where $H = H_1 + H_2 + \ldots + H_T$.

The advantage of the tree chaining construction is the following. The generation of a public MSS key that can verify 2^H documents requires the construction of a tree of height H, which in turn requires the computation of 2^H one-time key pairs and $2^{H+1}-1$ evaluations of the hash fuction. When tree chaining is used, the construction of a public CMSS key that can verify 2^H documents only requires the construction of the single Merkle tree on the top layer which is of height H_1. Also, in the tree chaining method, signature generation requires knowledge of the one-time signature of the root of one Merkle tree on each layer. Those roots and one-time signatures can be successively computed as they are used, whereas the root of the first tree on each layer is generated during the key generation. Hence, the CMSS key pair

generation requires the computation of $2^{H_1} + \ldots + 2^{H_T}$ one-time key pairs and $2^{H_1+1} + \ldots + 2^{H_T+1} - T$ evaluations of the hash function. This is a drastic improvement compared to the original MSS key pair generation as illustrated in the following example.

Example 5. Assume that the heights of all Merkle trees are equal, so $H_1 = \ldots = H_T = H$. The number of signatures that can be generated with this key pair is 2^{TH}. The CMSS key pair generation requires $T2^H$ one-time key pairs and $T2^{H+1} - T$ evaluations of the hash function. The original MSS key pair generation requires 2^{TH} one-time key pairs and $2^{TH+1} - 1$ evaluations of the hash function.

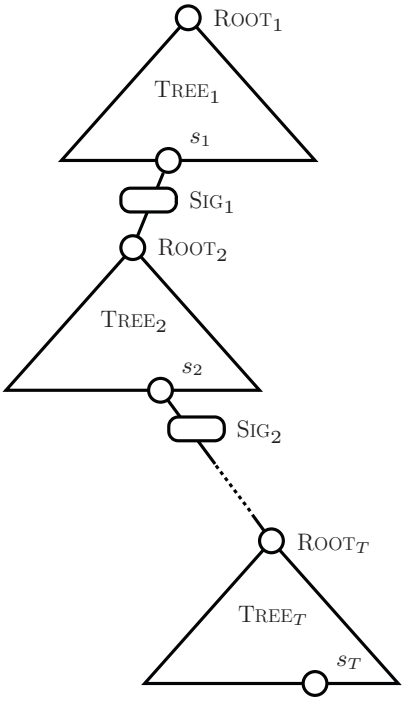

Fig. 8. The tree chaining method. TREE_i denotes the active tree on layer i, ROOT_i its root, and SIG_{i-1} this root's one-time signature generated with the s_{i-1}th signature key of the tree on layer $i-1$.

CMSS key pair generation

For the CMSS key pair generation, the number of layers T and the respective heights H_i, $1 \leq i \leq T$ of the trees on layer i are selected. With $H = H_1 + H_2 +$

$\ldots + H_T$ the number of signatures that can be generated/verified using the key pair to be constructed is 2^H. For each layer, one initial Merkle tree TREE_i is constructed as described in Sections 2 and 3. The CMSS public key is the root of TREE_1. The CMSS secret key is the sequence of the random seeds used to construct the T trees. The signer also stores the one-time signatures of the roots of all those trees generated with the first signature key of the tree on the next layer.

CMSS key pair generation requires the computation of $2^{H_1} + \ldots + 2^{H_T}$ one-time key pairs and $2^{H_1+1} + \ldots + 2^{H_T+1} - T$ evaluations of the hash function.

CMSS signature generation

We use the notation of the previous sections. When a signature is issued, the signer knows one active Merkle tree TREE_i for each layer and the seed SEED_i from which its signature keys can be generated, $i = 1, 2, \ldots, T$. The signer also knows the signature SIG_i of the root of TREE_{i+1}, and the verification key Y_i for that signature, $1 \leq i \leq T-1$. Further, the signer knows the index s_i, $1 \leq i \leq T-1$, of the signature key used to generate the signature SIG_i of the root of the tree TREE_{i+1} and the index s_T of the signature key used to issue the next document signature. The signer constructs the corresponding signature key from the seed SEED_T, he generates the one-time signature SIG_T of the document to be signed and he generates the signature as in Equation (51). The index s in this signature can be recursively computed. Set $t_1 = s_1$ and

$$t_{i+1} = t_i 2^{H_{i+1}} + s_{i+1}, 1 \leq i < T,$$

then $s = t_T$.

After signing, the signer prepares for the next signature by partially constructing the next tree on certain layers using the treehash algorithm of Section 2. He first computes the s_Tth leaf of the next tree on layer T and executes the treehash algorithm with this leaf as input. Then he increments s_T. If $s_T = 2^{H_T}$, then the construction of the next Merkle tree on layer T is completed and its root is available. The signer computes the one-time signature of this root using a signature key of the tree on layer $T-1$ and sets the index s_T to zero. In the same way, the signer constructs the next tree on layer $T-1$ and increments the index s_{T-1}. More generally, the signer partially constructs the next tree on layer i and increments s_i whenever the construction of the next tree on layer $i+1$ is complete, $1 < i < T$. On layer 1, no new tree is required and the signer only increments the index s_1 if the construction of a tree on layer 2 is completed. When $s_1 = 2^{H_1}$, CMSS cannot sign new documents anymore.

Since a CMSS signature consists of T MSS signatures, the signature size increases by a factor T compared to MSS. Also, the computation of the roots of the following trees and their signatures increases the signature generation time.

CMSS verification

The basics of the CMSS signature verification are straight forward and were already explained above.

We now explain how the verifier uses s to determine a positive integer s_i for each layer i, such that Y_i is the s_ith verification key of the active tree on that layer. The verifier uses s_i to construct the path from Y_i to the root of the corresponding tree on layer i (see Section 2). The following formulas show how this can be accomplished.

$$\begin{aligned} j_T &= \lfloor s/2^{H_T} \rfloor, & j_i &= \lfloor j_{i+1}/2^{H_i} \rfloor, i = T-1, \ldots, 1 \\ s_T &= s \bmod 2^{H_T}, & s_i &= j_{i+1} \bmod 2^{H_i}, i = T-1, \ldots, 1 \end{aligned} \tag{52}$$

6 Distributed signature generation

In this section, we describe *distributed signature generation* [4]. This method counteracts the new problems that arise when using the tree chaining method, namely the increased signature size and signature generation time. It is based on the observation that the one-time signatures of the roots and the authentication paths in upper layers change only infrequently. The idea is to distribute the operations required for the generation of these one-time signatures and authentication paths evenly across each step. This significantly improves the worst case signature generation time. Recall Section 1.2, where we showed that the Winternitz one-time signature scheme uses the parameter w to provide a trade-off between the signature generation time and the signature size. Using the method of distributed signature generation it is possible to choose large values of w for upper layers, which in turn results in smaller signatures. The combination of the tree chaining method, the distributed signature generation, and the original MSS is called GMSS.

The idea

Fix a layer $i \geq 2$. Denote the active tree on layer i by TREE_i. It is currently used to sign roots or documents. The preceding tree on that layer is denoted by TREEPREV_i. The next tree on layer i is TREENEXT_i. The idea of the distributed signature generation is the following. When TREE_i is used, the root of TREENEXT_i is known. The root of TREENEXT_i is signed while the signature keys of TREE_i are used. The root of TREENEXT_i was calculated while TREEPREV_i was used to sign documents or roots.

Distributed root signing

We use the notation from above. We explain how the root of TREENEXT_i is signed while TREE_i is used to sign. By construction, the necessary signature key from layer $i-1$ is known.

We distribute the computation of the signature of the root of TREENEXT_i across the leaves of TREE_i. When the first leaf of TREE_i is used we initialize the Winternitz one-time signature generation by calculating the parameters and executing the padding. Then we calculate the number of hash function evaluations and calls to the PRNG required to compute the one-time signature key and the one-time signature. We divide those numbers by 2^{H_i} where H_i is the height of TREE_i to estimate the number of operations required per step. When a leaf of TREE_i is used, the appropriate amount of computation for the signature of the root of TREENEXT_i is performed. The distributed generation of the one-time signatures is visualized in Figure 9.

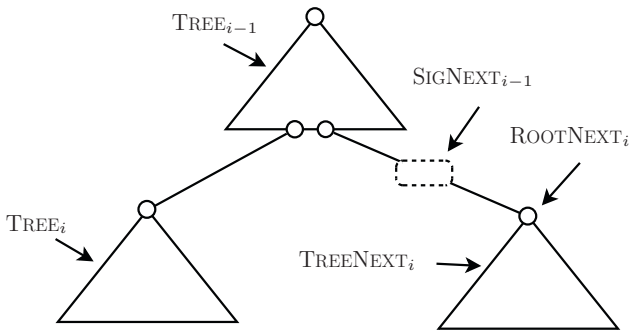

Fig. 9. Distributed generation of SIGNEXT_{i-1}, the one-time signature of the root of TREENEXT_i.

We estimate the running time of the distributed root signing. The one-time signature of a root of a tree on layer i is generated using the Winternitz parameter w_{i-1} of layer $i-1$. According to Section 1.2 the generation of this signature requires $(2^{w_{i-1}}-1)t_{w_{i-1}}$ hash function evaluations in the worst case. As shown in Section 3 the generation of the one-time signature requires $t_{w_{i-1}}+1$ calls to the PRNG. Since each tree on layer i has 2^{H_i} leaves, the computation of its root signature is distributed across 2^{H_i} steps. Therefore, the total number of extra operations for each leaf of TREE_i to compute the root signature of TREENEXT_i is at most

$$c_{\text{sig}}(i) = \left\lceil \frac{(2^{w_{i-1}}-1)t_{w_{i-1}}}{2^{H_i}} \right\rceil c_{\text{HASH}} + \left\lceil \frac{t_{w_{i-1}}+1}{2^{H_i}} \right\rceil c_{\text{PRNG}}. \tag{53}$$

Distributed root computation

We explain, how the root of TREENEXT_i is computed while TREEPREV_i is active. This is quite simple. Both TREEPREV_i and TREENEXT_i have the same number of leaves. When a leaf of TREEPREV_i is used, the leaf with the same

index in TreeNext$_i$ is calculated and passed to the treehash algorithm from Section 2.

If $i < T$, i.e. TreeNext$_i$ is not on the lowest level, the computation of each leaf of TreeNext$_i$ can also be distributed. This is explained next. Suppose that we want to construct the jth leaf of TreeNext$_i$ while we are using the jth leaf of TreePrev$_i$. This computation is distributed across the leaves of the tree TreeLower on layer $i+1$ whose root is signed using the jth leaf of TreePrev$_i$. When the first leaf of TreeLower is used, we determine the number of hash function evaluations and calls to the PRNG required to compute the jth leaf of TreeNext$_i$. Recall that the calculation of this leaf requires the computation of a Winternitz one-time key pair. We divide those numbers by $2^{H_{i+1}}$ to obtain the number of operations we will execute in each leaf of TreeLower. Whenever a leaf of TreeLower is used, the computation of the jth leaf of TreeNext is advanced by executing those operations.

Once the jth leaf of TreeNext$_i$ is generated, it is passed to the treehash algorithm. This contributes to the construction of the root of TreeNext$_i$. This construction is complete, once we switch from TreePrev$_i$ to Tree$_i$. So in fact, when Tree$_i$ is used, the root of TreeNext$_i$ is known. The distributed computation of the roots is visualized in Figure 10. While constructing TreeNext$_i$, we also perform the initialization steps of the authentication path algorithm of Section 4.5. That is, we store the authentication path of leaf 0 and prepare the algorithm state.

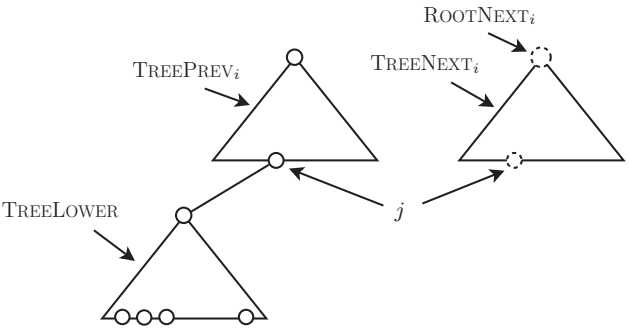

Fig. 10. Distributed computation of RootNext$_i$. Leaf j of tree TreeNext$_i$ is precomputed while using tree TreeLower. It is then used to partially compute RootNext$_i$.

We estimate the extra time required by the distributed root computation. Recall that for the generation of a leaf of TreeNext$_i$ we first determine the corresponding Winternitz one-time key pair. This key pair is constructed using the Winternitz parameter w_i of layer i. The generation of the one-time

signature key requires $t_{w_i} + 1$ calls to the PRNG. The generation of the one-time verification key requires $(2^{w_i} - 1)t_{w_i}$ hash function evaluations and the computation of a leaf of TREENEXT_i requires one additional evaluation of the hash function. This has been shown in Sections 1.2 and 3. Since TREELOWER has $2^{H_{i+1}}$ leaves, the computation of a leaf of TREENEXT_i can be distributed over $2^{H_{i+1}}$ steps. Therefore, the total number of extra operations for each leaf of TREELOWER to compute a leaf of TREENEXT_i is

$$c^1_{\text{leaf}}(i) = \left\lceil \frac{(2^{w_i} - 1)t_{w_i} + 1}{2^{H_{i+1}}} \right\rceil c_{\text{HASH}} + \left\lceil \frac{t_{w_i} + 1}{2^{H_{i+1}}} \right\rceil c_{\text{PRNG}}. \tag{54}$$

Once a leaf of TREENEXT_i is found, it is passed to the treehash algorithm. By the results of Section 2 this costs at most

$$c^2_{\text{leaf}}(i) = H_i \cdot c_{\text{HASH}} \tag{55}$$

additional evaluations of the hash function.

Distributed authentication path computation

Next, we describe the computation of the authentication path of the next leaf of tree TREE_i. We use the algorithm described in Section 4.5. This algorithm requires the computation of $(H_i - K_i)/2 + 1$ leaves per round to generate upcoming authentication paths on layer $i = 1, \ldots, T$. As described above, the computation of these leaves is distributed over the $2^{H_{i+1}}$ leaves (or steps) of tree TREELOWER, the current tree on the next lower layer $i + 1$. Again, this is possible only for leaves in layers $i = 1, \ldots, T - 1$. The computation of the leaves in layer T cannot be distributed.

When we use TREELOWER for the first time we calculate the number of hash function evaluations and calls to the PRNG required to compute the $(H_i - K_i)/2 + 1$ leaves. Recall that we have to compute a Winternitz one-time key pair to obtain this leaf. Then we divide these costs by $2^{H_{i+1}}$ to estimate the number of operations we have to spend for each leaf of tree TREELOWER. At the beginning we don't know which leaves must be computed, we only know how may. Therefore, we have to interact with Algorithm 4.6. We perform the necessary steps to decide which leaf must be computed first. After computing this leaf we pass it to the authentication path algorithm which updates the treehash instance and determines the which leaf must be computed next. This procedure is iterated until all required leaves are computed. The distributed authentication path computation is visualized in Figure 11.

We estimate the cost of the distributed authentication path computation. The algorithm of Section 4.5 requires the computation of $(H_i - K_i)/2 + 1$ leaves for each authentication path. The leaves are computed using the Winternitz parameter w_i of layer i. The generation of one leaf requires $t_{w_i} + 1$ calls to the PRNG and $(2^{w_i} - 1)t_{w_i} + 1$ hash function evaluations, see Sections 1.2 and 3. The computation of the those $(H_i - K_i)/2 + 1$ leaves is distributed

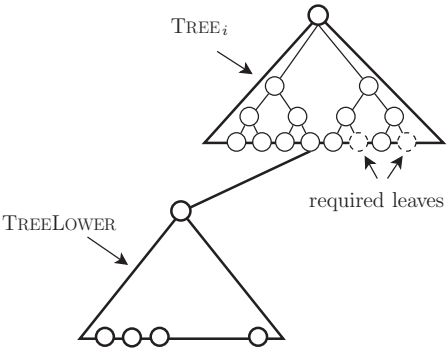

Fig. 11. Distributed computation of the next authentication path. The $(H_i - K_i)/2$ required leaves are computed while using tree TREELOWER.

over the $2^{H_{i+1}}$ steps in the tree on layer $i+1$. Therefore, the total number of operations for each leaf of TREELOWER to compute the $(H_i - K_i)/2 + 1$ leaves is

$$c^1_{\text{auth}}(i) = \frac{H_i - K_i + 2}{2} \cdot c^1_{\text{leaf}}(i). \tag{56}$$

The completed leaves are passed to the treehash algorithm that computes their parent nodes. The algorithm of Section 4.5 requires at most $3(H_i - K_i - 1)/2 + 1$ evaluations of the hash function for the computation of parents. Another $H_i - K_i$ calls to the PRNG are required to prepare upcoming seeds. These operations are not distributed but performed at once. Hence, the total number of operations for each leaf of TREE$_i$ is at most

$$c^2_{\text{auth}}(i) = \frac{3(H_i - K_i) - 1}{2} \cdot c_{\text{HASH}} + (H_i - K_i) \cdot c_{\text{PRNG}}. \tag{57}$$

Example 6. This example illustrates how the distributed signature generation improves the signature generation time. Let $H_1 = \ldots = H_T = H$. Further, all layers use the same Winternitz parameter w and the same value for K. Let c_{sig} denote the worst case cost for generating a one-time signature with Winternitz parameter w, let c_{auth} denote the worst case cost for generating an authentication path in a tree of height H using K, and let c_{tree} denote the cost for partially computing the next tree. The worst case cost for the GMSS signature generation then is

$$c_{\text{sig}} + c_{\text{auth}} + c_{\text{tree}} + \frac{(T-1)c_{\text{sig}} + (T-1)c_{\text{auth}} + (T-2)c_{\text{tree}}}{2^H}.$$

When the signature generation is not distributed, as in the case of CMSS, the worst case cost is

$$Tc_{\text{sig}} + Tc_{\text{auth}} + (T-1)c_{\text{tree}}.$$

GMSS key pair generation

We explain GMSS key pair generation, establish the size of the keys, and the cost for computing them. The following parameters are selected. The number T of layers, the heights $H_1, \ldots, H_T$ of the Merkle trees on each layer, the Winternitz parameters $w_1, \ldots, w_T$ for each layer, and the parameters $K_1, \ldots, K_T$ for the authentication path algorithm of Section 4.5.

We use the approach introduced in Section 3 and use an PRNG for the one-time signature generation. Therefore we must choose initial seeds $\textsc{Seed}_i$, for each layer $i = 1, \ldots, T$. The GMSS public key is the root $\textsc{Root}_1$ of the single tree in layer $i = 1$. The GMSS private key consists of the following entries:

$$\begin{array}{rcr} \textsc{Seed}_i\ , i = 1, \ldots, T & , & \textsc{SeedNext}_i\ , i = 2, \ldots, T \\ \textsc{Sig}_i\ , i = 1, \ldots, T-1\ , & & \textsc{RootNext}_i\ , i = 2, \ldots, T \\ \textsc{Auth}_i\ , i = 1, \ldots, T & , & \textsc{AuthNext}_i\ , i = 2, \ldots, T \\ \textsc{State}_i\ , i = 1, \ldots, T & , & \textsc{StateNext}_i\ , i = 2, \ldots, T \end{array} \tag{58}$$

The seeds $\textsc{Seed}_i$ are required for the generation of the one-time signature keys used to sign the data and the roots. The seeds $\textsc{SeedNext}_i$ are required for the distributed generation of subsequent roots. These seeds are available after the generation of the roots $\textsc{RootNext}_i$. The one-time signatures $\textsc{Sig}_i$ of the roots are required for the GMSS signatures. The signatures $\textsc{Sig}_i$ do not have to be computed explicitly. They are an intermediate value during the computation of the 0th leaf of tree $\textsc{Tree}_{i-1}$. The roots $\textsc{RootNext}_i$ of the next tree in each layer are required for the distributed generation of the one-time signatures $\textsc{SigNext}_{i-1}$. Also, the authentication path for the first leaf of the first and second tree in each layer is stored. $\textsc{State}_i$ and $\textsc{StateNext}_i$ denote the state of the authentication path algorithm of section 4.5 required to compute authentication paths in trees $\textsc{Tree}_i$ and $\textsc{TreeNext}_i$, respectively. This state contains the seeds and the treehash instance and is initialized during the generation of the root.

The construction of a tree on layer i requires the computation of 2^{H_i} leaves and $2^{H_i} - 1$ evaluations of the hash function to compute inner nodes. Each leaf computation requires $(2^{w_i} - 1) \cdot t_{w_i} + 1$ hash function evaluations and $t_{w_i} + 1$ calls to the PRNG. The total cost for one tree on layer i is given as

$$c_{\text{tree}}(i) = \left(2^{H_i}\left(t_{w_i}(2^{w_i} - 1) + 2\right) - 1\right) c_{\textsc{Hash}} + 2^{H_i}\left(t_{w_i} + 1\right) c_{\textsc{Prng}}. \tag{59}$$

Since we construct two trees on layers $i = 2, \ldots, T$ and one on layer $i = 1$, the total cost for the key pair generation is

$$c_{\text{keygen}} = \sum_{i=1}^{T} c_{\text{tree}}(i) + \sum_{i=2}^{T} c_{\text{tree}}(i). \tag{60}$$

The memory requirements of the keys depend on the output size of the used hash function n. A root is a single hash value and requires n bits. A seed

also requires n bits. A one-time signature SIG_i requires $t_{w_{i-1}} \cdot n$ bits. An authentication path together with the algorithm state requires

$$m_{\text{auth}}(i) = \left(3H_i + \left\lfloor \frac{H_i}{2} \right\rfloor - 3K_i - 2 + 2^{K_i}\right) \cdot n \text{ bits.} \tag{61}$$

For each layer $i = 2, \ldots, T$, we store two seeds, two authentication paths and algorithm states, one root and the one-time signature of one root. For layer $i = 1$, we store one seed and one authentication path and algorithm state. The total sizes of the public and the private key are

$$m_{\text{pubkey}} = n \text{ bits,} \tag{62}$$

$$m_{\text{privkey}} = \left(\sum_{i=1}^{T}(m_{\text{auth}}(i) + 1) + \sum_{i=2}^{T}(m_{\text{auth}}(i) + t_{w_{i-1}} + 2)\right) n \text{ bits.} \tag{63}$$

GMSS signature generation

The GMSS signature generation is split in two parts, an online part and an offline part. The online part is equivalent to the CMSS online part. The signer constructs the corresponding signature key from the seed SEED_T and generates the one-time signature SIG_T of the document to be signed. Then he prepares the signature as in Equation (64). The offline part takes care of the distributed computation of upcoming roots, one-time signatures of roots and authentication paths as described above.

$$\begin{aligned} \sigma_s = \big(s,\ & \text{SIG}_T, Y_T, \text{AUTH}_T, \\ & \text{SIG}_{T-1}, Y_{T-1}, \text{AUTH}_{T-1} \\ & \quad \vdots \\ & \text{SIG}_1, Y_1, \text{AUTH}_1\big). \end{aligned} \tag{64}$$

The online part requires the generation of a single one-time signature. This signature is generated using the Winternitz parameter of the lowest layer T. According to Section 1.2, this requires

$$c_{\text{online}} = (2^{w_T} - 1)t_{w_T} \cdot c_{\text{HASH}} + (t_{w_T} + 1)c_{\text{PRNG}}. \tag{65}$$

operations in the worst case. The size of an GMSS signature is computed with the same formula we used for as the CMSS signatures. It consists of T authentication paths ($H_i \cdot n$ bits) and T one-time signatures ($t_{w_i} \cdot n$ bits), one for each layer $i = 1, \ldots, T$. Adding up yields

$$m_{\text{signature}} = \sum_{i=1}^{T}(H_i + t_{w_i}) \cdot n \text{ bits.} \tag{66}$$

To estimate the computational effort required for the offline part we assume the worst case where we have to advance one leaf on all layers

$i = 1, \ldots, T$. The computation of the one-time signature SIGNEXT_i can be distributed for each layers $i = 1, \ldots, T-1$. The computation of the leaves required to construct the root ROOTNEXT_i can be distributed for all layers $i = 2, \ldots, T-1$. For layer $i = T$, the respective leaf of tree TREENEXT_T must be computed at once. Together with the hash function evaluations for the treehash algorithm, this requires at most

$$c^3_{\text{leaf}} = ((2^{w_T} - 1)t_{w_T} + H_T + 1)c_{\text{HASH}} + (t_{w_T} + 1)c_{\text{PRNG}} \tag{67}$$

operations. The leaves required for the computation of upcoming authentication paths can be distributed for all layers $i = 1, \ldots, T-1$. For layer $i = T$, the $(H_T - K_T)/2 + 1$ leaves must be computed at once. Together with the hash function evaluations for the treehash algorithm, this requires at most

$$\begin{aligned} c^3_{\text{auth}} = &\frac{H_T - K_T + 2}{2} \cdot c^3_{\text{leaf}} + \frac{3(H_T - K_T) - 1}{2} \cdot c_{\text{HASH}} \\ &+ (H_T - K_T) \cdot c_{\text{PRNG}} \end{aligned} \tag{68}$$

operations. In summary, the number of operations required by the offline part in the worst case are

$$\begin{aligned} c_{\text{offline}} = &\sum_{i=2}^{T} c_{\text{sig}}(i) + \sum_{i=2}^{T-1} \left(c^1_{\text{leaf}}(i) + c^2_{\text{leaf}}(i)\right) + c^3_{\text{leaf}} \\ &+ \sum_{i=1}^{T-1} \left(c^1_{\text{auth}}(i) + c^2_{\text{auth}}(i)\right) + c^3_{\text{auth}}. \end{aligned} \tag{69}$$

The last step is to estimate the space required by the offline part. We have to store the partially constructed one-time signature SIGNEXT_i for layers $i = 1, \ldots, T-1$ which requires at most $t_{w_{i-1}} \cdot n$ bits. We also have to store the treehash stack for the generation of the root ROOTNEXT_i for layers $i = 2, \ldots, T$ which requires $H_i \cdot n$ bits. We further require memory to store partially constructed leaves. One leaf requires at most $t_{w_i} \cdot n$ bits. For the generation of ROOTNEXT_i we have to store at most one leaf for each layer $i = 2, \ldots, T-1$. For the authentication path, we have to store at most one leaf for each layer $i = 1, \ldots, T-1$. Note that since we compute the leaves required for the authentication path successively, we have to store only one partially constructed leaf at a time. Finally, we need to store the partial state STATENEXT_i of the authentication path algorithm for layers $i = 2, \ldots, T$ which requires at most $m_{\text{auth}}(i)$ bits (see Equation (61)). In summary, the memory required by the offline part in the worst case is

$$m_{\text{offline}} = \left(\sum_{i=2}^{T} \left(t_{w_{i-1}} + H_i + m_{\text{auth}}(i)\right) + \sum_{i=2}^{T-1} t_{w_i} + \sum_{i=1}^{T-1} t_{w_i} \right) \cdot n \text{ bits.} \tag{70}$$

GMSS signature verification

Since the main idea of GMSS is to distribute the signature generation, the signature verification doesn't change compared to CMSS. The verifier successively verifies a one-time signature and uses the corresponding authentication path and Equation (52) to compute the root. This is done until the root of the tree in the top layer is computed. If this root matches the signers public key, the signature is valid.

The verifier must verify T one-time signatures which in the worst case requires $(2^{w_i}-1)t_{w_i}$ evaluations of the hash function, for $i = 1, \ldots, T$. Another H_i evaluations of the hash function are required to reconstruct the path to the root using the authentication path. In total, the number of hash function evaluations required in the worst case is

$$c_{\text{verify}} = \sum_{i=1}^{T} ((2^{w_i} - 1)t_{w_i} + H_i)\, c_{\text{HASH}}. \tag{71}$$

7 Security of the Merkle Signature Scheme

This section deals with the security of the Merkle signature scheme. We will show that the Lamport–Diffie one-time signature scheme is existentially unforgeable under an adaptive chosen message attack (CMA-secure) as long as the used one-way function is preimage resistant. Then we show that the Merkle signature scheme is CMA-secure as long as the used hash function is collision resistant and the underlying one-time signature scheme is CMA-secure. Finally, we estimate the security level of the Merkle signature scheme for a given output length n of the hash function.

7.1 Notations and definitions

We start with some security notions and definitions.

Security notions for hash functions

We present three security notions for hash functions: preimage resistance, second preimage resistance, and collision resistance. The definitions are taken from [30]. We write $x \xleftarrow{\$} S$ for the experiment of choosing a random element from the finite set S with the uniform distribution. Let $\mathcal{G}$ be a family of hash functions, that is, a parameterized set

$$\mathcal{G} = \left\{ g_k : \{0,1\}^* \to \{0,1\}^n | k \in K \right\} \tag{72}$$

where $n \in \mathbb{N}$ and K is a finite set. The elements of K are called *keys*. An *adversary* ADV is a probabilistic algorithm that takes any number of inputs.

We define *preimage resistance*. In fact, our notion of preimage resistance is a special case of the preimage resistance defined in [30] which is useful in our context. Consider an adversary that attempts to find preimages of the hash functions in $\mathcal{G}$. The adversary takes as input a key $k \in K$ and the image $y = g_k(x)$ of a string $x \in \{0,1\}^n$. Both k and x are chosen randomly with the uniform distribution. The adversary outputs a *preimage* x' of y or `failure`. The success probability of this adversary is denoted by

$$\Pr[k \xleftarrow{\$} K, x \xleftarrow{\$} \{0,1\}^n, y \longleftarrow g_k(x), x' \xleftarrow{\$} \textsc{Adv}(k,y) : g_k(x') = y]. \quad (73)$$

Let t, ϵ be positive real numbers. The family $\mathcal{G}$ is called (t,ϵ) preimage resistant, if the success probability (73) of any adversary Adv that runs in time t is at most ϵ.

Next, we define *second preimage resistance*. Consider an adversary that attempts to find second preimages of the hash functions in $\mathcal{G}$. The adversary takes as input a key $k \in K$ and a string $x \in \{0,1\}^n$, both chosen randomly with the uniform distribution. He outputs a *second preimage* x' under g_k of $g_k(x)$ which is different from x or `failure`. The success probability of this adversary is denoted by

$$\Pr[k \xleftarrow{\$} K, x \xleftarrow{\$} \{0,1\}^n, x' \xleftarrow{\$} \textsc{Adv}(k,x) : x \neq x' \wedge g_k(x) = g_k(x')]. \quad (74)$$

Let t, ϵ be positive real numbers. The family $\mathcal{G}$ is called (t,ϵ) second-preimage resistant, if the success probability (74) of any adversary Adv that runs in time t is at most ϵ.

Finally, we define *collision resistance*. Consider an adversary that attemps to find collisions of the hash functions in $\mathcal{G}$. The adversary takes as input a key $k \in K$, chosen randomly with the uniform distribution. He outputs a *collision* of g_k, that is, a pair $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $g(x) = g(x')$ or `failure`. The success probability of this adversary is denoted by

$$\Pr[k \xleftarrow{\$} K, (x,x') \xleftarrow{\$} \textsc{Adv}(k) : x \neq x' \wedge g_k(x) = g_k(x')]. \quad (75)$$

Let t, ϵ be positive real numbers. The family $\mathcal{G}$ is called (t,ϵ) collision resistant, if the success probability (75) of any adversary Adv that runs in time t is at most ϵ.

Signature schemes

Let Sign be a signature scheme. So Sign is a triple (Gen, Sig, Ver). Gen is the key pair generation algorithm. It takes as input 1^n, the string of n successive 1s where $n \in \mathbb{N}$ is a security parameter. It outputs a pair $(\mathsf{sk}, \mathsf{pk})$ consisting of a private key sk and a public key pk. Sig is the signature generation algorithm. It takes as input a message M and a private key sk. It outputs a signature σ for the message M. Finally, Ver is the verification algorithm. Its input is a message M, a signature σ and a public key pk. It checks whether σ is a valid signature for M using the public key pk. It outputs `true` if the signature is valid and `false` otherwise.

Existential unforgeability

Let $\textsc{Sign} = (\textsc{Gen}, \textsc{Sig}, \textsc{Ver})$ be a signature scheme and let $(\mathsf{sk}, \mathsf{pk})$ be a key pair generated by $\textsc{Gen}$. We define *existential unforgeability under an adaptive chosen message attack* of $\textsc{Sign}$. This security model assumes a very powerful forger. The forger has access to the public key and a signing oracle $\mathcal{O}(\mathsf{sk}, \cdot)$ that, in turn, has access to the private key. On input of a message the oracle returns the signature of that message. It is the goal of the forger to win the following game. The forger chooses at most q messages and lets the signing oracle find the signatures of those messages. The maximum number q of queries is also an input of the forger. The oracle queries may be adaptive, that is, a message may depend on the oracles answers to previously queried messages. The forger outputs a pair (M', σ'). The forger wins if M is different from all the messages in the oracle queries and if $\textsc{Ver}(M', \sigma', \mathsf{pk}) = \mathtt{true}$. We denote such a forger by $\textsc{For}^{\mathcal{O}(\mathsf{sk},\cdot)}(\mathsf{pk})$.

Let t and ϵ be positive real numbers and let q be a positive integer. The signature scheme $\textsc{Sign}$ is (t, ϵ, q) *existentially unforgeable under an adaptive chosen message attack* if for any forger that runs in time t, the success probability for winning the above game (which depends on q) is at most ϵ. If $\textsc{Sign}$ has the above property it is also called a (t, ϵ, q) *signature scheme*.

For one-time signatures we must have $q = 1$ since the signature key of a one-time signature scheme must be used only once. For the Merkle signature scheme we must have $q \leq 2^H$.

7.2 Security of the Lamport–Diffie one-time signature scheme

In this section we discuss the security of LD–OTS from Section 1.1. We slightly modify this scheme. Select a security parameter $n \in \mathbb{N}$. Let $K = K(n)$ be a finite set of parameters. Let

$$\mathcal{F} = \left\{ f_k : \{0,1\}^n \to \{0,1\}^n | k \in K \right\}$$

be a family of one-way functions. The key generation of the modified LD–OTS works as follows. On input of 1^n for a security parameter n a key $k \in K(n)$ is selected randomly with the uniform distribution. Then LD–OTS is used with the one-way function f_k. The secret and public keys are generated as described in Section 1.1. The key k is included in the public key. We show that the existential unforgeability under adaptive chosen message attacks of this LD-OTS variant can be reduced to the preimage resistance of the family $\mathcal{F}$.

Suppose that there exists a forger $\textsc{For}^{\mathcal{O}(X,\cdot)}(Y)$ of LD-OTS. Then an adversary $\textsc{Adv}_{\mathsf{Pre}}$ that determines preimages of functions in $\mathcal{F}$ can be constructed as follows. Fix a security parameter n. Input for $\textsc{Adv}_{\mathsf{Pre}}$ are a key k and the image $y = f_k(x)$ of a string $x \in \{0,1\}^n$. Both k and x are selected randomly with the uniform distribution. A LD–OTS key pair

(X, Y) is generated using the one-way function f_k. The public key Y is of the form $Y = (y_{n-1}[0], y_{n-1}[1], \ldots, y_0[0], y_0[1])$. The adversary selects indices $a \in \{0, \ldots, n-1\}$ and $b \in \{0, 1\}$ randomly with the uniform distribution. He replaces the string $y_a[b]$ with the target string y. Next, $\text{ADV}_{\mathsf{Pre}}$ runs the forger $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ with the modified public key. If the forger asks its oracle to sign a message $M = (m_{n-1}, \ldots, m_0)$ and if $m_a = 1 - b$, then the adversary, playing the role of the oracle, signs the message and returns the signature. The adversary can sign this message since he knows the original key pair and because of $m_a = 1 - b$, the modified string in the public key is not used. However, if $m_a = b$ then the adversary cannot sign M. So his answer to the oracle query is `failure` which also causes the forger to abort. If the forger's oracle query was successful or if the forger does not ask the oracle at all the forger may produce a message $M' = (m'_{n-1}, \ldots, m'_0)$ and the signature $(\sigma'_{n-1}, \ldots, \sigma'_0)$ of that message. If $m'_a = b$, then σ'_a is the preimage of y which the adversary returns. Otherwise, the adversary returns `failure`. More formally, the adversary is presented in Algorithm 7.1.

Algorithm 7.1 $\text{ADV}_{\mathsf{Pre}}$

Input: $k \xleftarrow{\$} K$ and $y = f_k(x)$, where $x \xleftarrow{\$} \{0,1\}^n$
Output: x' such that $y = f_k(x)$ or `failure`

1. Generate an LD–OTS key pair (X, Y).
2. Choose $a \xleftarrow{\$} \{0, \ldots, n-1\}$ and $b \xleftarrow{\$} \{0, 1\}$.
3. Replace $y_a[b]$ by y in the LD–OTS verification key Y.
4. Run $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$.
5. When $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ asks its only oracle query with $M = (m_{n-1}, \ldots, m_0)$:
   a) **if** $m_a = (1 - b)$ **then** sign M and respond to the forger $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ with the signature σ.
   b) **else return** `failure`.
6. When $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ outputs a valid signature $\sigma' = (\sigma'_{n-1}, \ldots, \sigma'_0)$ for message $M' = (m'_0, \ldots, m'_{n-1})$:
   a) **if** $m'_a = b$ **then return** σ'_a as preimage of y.
   b) **else return** `failure`.

We now compute the success probability of the adversary $\text{ADV}_{\mathsf{Pre}}$. We denote by ϵ the forger's success probability for producing an existential forgery of the LD–OTS and by t its running time. By t_{GEN} and t_{SIG} we denote the times the LD–OTS requires for key and signature generation, respectively.

The adversary $\text{ADV}_{\mathsf{Pre}}$ is successful in finding a preimage of y if and only if $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ queries the oracle with a message $M = (m_{n-1}, \ldots, m_0)$ with $m_a = (1 - b)$ (Line 5a) or if he queries the oracle not at all and if the forger returns a valid signature for message $M' = (m'_0, \ldots, m'_{n-1})$ with $m'_a = b$ (Line 6a). Since b is selected randomly with the uniform distribution, the probability for $m_a = (1 - b)$ is $1/2$. Since M' must be different from the queried message

M, there exists at least one index c such that $m'_c = 1 - m_c$. $\text{ADV}_{\mathsf{Pre}}$ is successful if $c = a$, which happens with probability at least $1/2n$. Hence, the adversary's success probability for finding a preimage in time $t_{\text{OW}} = t + t_{\text{SIG}} + t_{\text{GEN}}$, is at least $\epsilon/4n$. We have proved the following theorem.

Theorem 4. *Let $n \in \mathbb{N}$, let K be a finite parameter set, let t_{OW}, ϵ_{OW} be positive real numbers, and $\mathcal{F} = \{f_k : \{0,1\}^n \to \{0,1\}^n | k \in K\}$ be a family of $(t_{\text{OW}}, \epsilon_{\text{OW}})$ one-way functions. Then the LD–OTS variant that uses $\mathcal{F}$ is $(t_{\text{OTS}}, \epsilon_{\text{OTS}}, 1)$ existentially unforgeable under an adaptive chosen message attack with $\epsilon_{\text{OTS}} \leq 4n \cdot \epsilon_{\text{OW}}$ and $t_{\text{OTS}} = t_{\text{OW}} - t_{\text{SIG}} - t_{\text{GEN}}$ where t_{GEN} and t_{SIG} are the key generation and signing times of LD–OTS, respectively.*

7.3 Security of the Merkle signature scheme

This section discusses the security of the Merkle signature scheme. We modify the Merkle scheme slightly. Select a security parameter $n \in N$. Let $K = K(n)$ be a finite set of parameters. Let

$$\mathcal{G} = \{g_k : \{0,1\}^* \to \{0,1\}^n | k \in K\}$$

be a family of hash functions. The key generation of the modified MSS works as follows. On input of 1^n for a security parameter n a key $k \in K(n)$ is selected randomly with the uniform distribution. Then the Merkle signature scheme is used with the hash function g_k and some one-time signature scheme. The secret and public keys are generated as described in Section 2. The parameter k is included in the public key. We show that the existential unforgeability of this MSS variant under an adaptive chosen message attack can be reduced to the collision resistance of the family $\mathcal{G}$ and the existential unforgeability of the underlying one-time signature scheme.

We explain how an existential forger for the Merkle signature scheme can be used to construct an adversary that is either an existential forger for the underlying one-time signature scheme or a collision finder for a hash function in $\mathcal{G}$. The input of the adversary is a one-time signature scheme, a key $k \in K$ chosen randomly with the uniform distribution, and the Merkle tree height H. Input is also a verification key Y_{OTS} and a signing oracle $\mathcal{O}_{\text{OTS}}(X_{\mathsf{OTS}}, \cdot)$, where $(X_{\mathsf{OTS}}, Y_{\mathsf{OTS}})$ is a key pair of the one-time signature scheme.

The adversary is allowed to query the oracle $\mathcal{O}_{\text{OTS}}(X_{\mathsf{OTS}}, \cdot)$ once. He aims to output a collision for the hash function g_k or an existential forgery (M', σ') for the one-time signature scheme that can be verified using the verification key Y_{OTS}. He has access to an adaptive chosen message forger $\text{FOR}^{\mathcal{O}(\mathsf{sk},\cdot)}(\mathsf{pk})$ for the MSS with hash function g_k and tree height H. The forger is allowed to ask 2^H queries to its signature oracle. The adversary is supposed to impersonate that oracle.

The adversary selects randomly with the uniform distribution an index c in the set $\{0, \ldots, 2^H - 1\}$. He generates a Merkle key pair in the usual manner with the only exception that as the cth one-time verification key the one-time

verification key Y_{OTS} from the input is used. Then the adversary invokes the adaptive chosen message forger for the Merkle scheme with the hash function g_k and the public Merkle key which he generated before. Without loss of generality, we assume that the forger queries the oracle 2^H times. The oracle answers are given by the adversary. When the forger asks for the ith signature, $i \neq c$, then the adversary produces this signatures using the signature keys which he generated before. However, when the forger asks for the cth signature, the adversary queries the oracle $\mathcal{O}_{\mathrm{OTS}}(X_{\mathsf{OTS}}, \cdot)$. Suppose that the forger is successful and outputs an existential forgery $(M', (s, \sigma', Y', A'))$ where s is the index of the one-time key pair used for this signature, σ' is the one-time signature, Y' is the verification key and A' is the authentication path. The adversary examines the Merkle signature (s, σ, Y, A) of M he returned in response to the forgers sth oracle query.

If $s = c$ and $(Y, A) = (Y', A')$, then the adversary returns (M', σ'). We show that this is an existential forgery of the one-time signature scheme with verification key Y_{OTS}. Since $s = c$ we have $Y = Y' = Y_{\mathsf{OTS}}$. So the verification key in the message returned by the forger is the same as the verification key returned by the oracle when it is queried for the cth time. The same is true for the authentication path. This implies that the message M in the cth oracle query is different from M'. So (M', σ') is an existential forgery.

If $(Y, A) \neq (Y', A')$, then the adversary can construct a collision for the hash function g_k as follows. Consider the path $B = (B_0 = g_k(Y), B_1, \ldots, B_H)$ from Y in the Merkle tree to its root constructed using the hash function g_k and the authentication path $A = (A_0, \ldots, A_{H-1})$. Compare it to the path $B' = (B'_0 = g_k(Y'), B'_1, \ldots, B'_H)$ from Y' in the Merkle tree to its root constructed using the authentication path $A' = (A'_0, \ldots, A'_{H-1})$. First assume that B and B' are different. For example, this is true when $Y \neq Y'$. Since $B_H = B'_H$ is the MSS public key, there is an index $0 \leq i < H$ with $B_{i+1} = B'_{i+1}$ and $B_i \neq B'_i$. Since B_{i+1} is the hash value of the concatenation of B_i and A_i (in the appropriate order), and since B'_{i+1} is the hash value of the concatenation of B'_i and A'_i (in the appropriate order), a collision of g_k is found. Next, assume that B and B' are equal. Therefore $g_k(Y) = B_0 = B'_0 = g_k(Y')$ holds. If $Y \neq Y'$ a collision is found. If $Y = Y'$ then A and A' are different. Assume that $A_i \neq A'_i$ for some index $i < H$. Since B_{i+1} is the hash value of the concatenation of B_i and A_i (in the appropriate order), and since B'_{i+1} is the hash value of the concatenation of B'_i and A'_i (in the appropriate order) again a collision is found. That collision is returned by the adversary. In all other cases the adversary returns failure. Algorithm 7.2 summarizes our description.

We now estimate the success probability of the adversary $\mathrm{ADV}_{\mathsf{CR},\mathsf{OTS}}$. In the following, ϵ denotes the success probability and t the running time of the forger. Also, $t_{\mathrm{GEN}}, t_{\mathrm{SIG}}$, and t_{VER} denote the times MSS requires for key generation, signature generation, and verification, respectively.

If $(Y', A') \neq (Y, A)$, then the adversary returns a collision. His (conditional) probability ϵ_{CR} for returning a collision in time $t_{\mathrm{CR}} = t + 2^H \cdot t_{\mathrm{SIG}} + t_{\mathrm{VER}} + t_{\mathrm{GEN}}$

Algorithm 7.2 $\text{ADV}_{\mathsf{CR},\mathsf{OTS}}$

Input: Key for the hash function $k \xleftarrow{\$} K$, height of the tree $H \geq 2$, one instance of the underlying OTS consisting of a verification key Y_{OTS} and the corresponding signing oracle $\mathcal{O}_{\text{OTS}}(X_{\mathsf{OTS}}, \cdot)$.
Output: A collision of g_k, an existential forgery for the supplied instance of the OTS, or `failure`

1. Set $c \xleftarrow{\$} \{0, \ldots, 2^H - 1\}$.
2. Generate OTS key pairs $(X_j, Y_j), j = 0, \ldots, 2^H - 1, j \neq c$ and set $Y_c \leftarrow Y_{\mathsf{OTS}}$.
3. Complete the Merkle key pair generation and obtain $(\mathsf{sk}, \mathsf{pk})$.
4. Run $\text{FOR}^{\mathcal{O}(\mathsf{sk},\cdot)}(\mathsf{pk})$.
5. When $\text{FOR}^{\mathcal{O}(\mathsf{sk},\cdot)}(\mathsf{pk})$ asks its qth oracle query $(0 \leq q \leq 2^H - 1)$:
   a) **if** $q = c$ **then** query the signing oracle $\mathcal{O}_{\text{OTS}}(X_{\mathsf{OTS}}, \cdot)$.
   b) **else** compute the one-time signature σ using the qth signature key X_q.
   c) Return the corresponding Merkle signature to the forger.
6. If the forger outputs an existential forgery $(M', (s, \sigma', Y', A'))$, examine the Merkle signature (s, σ, Y, A) returned in response to the forgers sth oracle query.
   a) **if** $(Y', A') \neq (Y, A)$ **then return** a collision of g_k.
   b) **else**
      i. **if** $s = c$ **then return** (M', σ') as forgery for the supplied instance of the one-time signature scheme.
      ii. **else return** `failure`.

is at least ϵ. If $(Y', A') = (Y, A)$ the adversary returns an existential forgery if $s = c$. His (conditional) probability ϵ_{OTS} for finding an existential forgery in time $t_{\text{OTS}} = t + 2^H \cdot t_{\text{SIG}} + t_{\text{VER}} + t_{\text{GEN}}$ is at least $\epsilon \cdot 1/2^H$. Since both cases are mutually exclusive, one of them occurs with probability at least $1/2$. So we have proved the following theorem.

Theorem 5. *Let K be a finite set, let $H \in \mathbb{N}$, $t_{\text{CR}}, t_{\text{OTS}}, \epsilon_{\text{CR}}, \epsilon_{\text{OTS}} \in \mathbb{R}_{>0}$, $\epsilon_{\text{CR}} \leq 1/2$, $\epsilon_{\text{OTS}} \leq 1/2^{H+1}$, and let $\mathcal{G} = \{g_k : \{0,1\}^* \to \{0,1\}^n | k \in K\}$ be a family of $(t_{\text{CR}}, \epsilon_{\text{CR}})$ collision resistant hash functions. Consider MSS using a $(t_{\text{OTS}}, \epsilon_{\text{OTS}}, 1)$ signature scheme. Then MSS is a $(t, \epsilon, 2^H)$ signature scheme with*

$$\epsilon \leq 2 \cdot \max\{\epsilon_{\text{CR}}, 2^H \cdot \epsilon_{\text{OTS}}\} \tag{76}$$

$$t = \min\{t_{\text{CR}}, t_{\text{OTS}}\} - 2^H \cdot t_{\text{SIG}} - t_{\text{VER}} - t_{\text{GEN}}. \tag{77}$$

This theorem tell us that if there is no adversary that breaks the collision resistance of the family $\mathcal{G}$ in time at most t_{CR} with probability greater than ϵ_{CR} and there is no adversary that is able to produce an existential forgery for the one-time signature scheme used in MSS in time at most t_{OTS} with probability greater than ϵ_{OTS}, then there exists no forger for MSS running in time at most $\min\{t_{\text{CR}}, t_{\text{OTS}}\} - 2^H \cdot t_{\text{SIG}} - t_{\text{VER}} - t_{\text{GEN}}$ and success probability greater then $2 \cdot \max\{\epsilon_{\text{CR}}, 2^H \cdot \epsilon_{\text{OTS}}\}$.

7.4 The security level of MSS

The goal of this section is to estimate the security level of the Merkle signature scheme when used with the Lamport–Diffie one-time signature scheme for a given output length n of the hash function. Let $b \in \mathbb{N}$. We say that MSS has security level 2^b if the expected number of hash function evaluations required for the generation of an existential forgery is at least 2^b. This security level can be computed as t/ϵ where t is the running time of an existential forger and ϵ is its success probability. We also say that the signature scheme has b *bits of security* or that the *bit security* is b. In this section let $\epsilon_{\text{CR}}, t_{\text{CR}}, \epsilon_{\text{OW}}, t_{\text{OW}} \in \mathbb{R}_{>0}$, let K be a finite set, and let

$$\mathcal{G} = \left\{ g_k : \{0,1\}^* \to \{0,1\}^n | k \in K \right\} \tag{78}$$

be a family of $(t_{\text{CR}}, \epsilon_{\text{CR}})$ collision resistant and $(t_{\text{OW}}, \epsilon_{\text{OW}})$ preimage resistant hash functions.

Since we consider MSS using LD-OTS, we first combine Theorems 4 and 5. This is achieved by substituting the values for ϵ_{OTS} and t_{OTS} from Theorem 4 in Equations (76) and (77) from Theorem 5. This yields

$$\epsilon \leq 2 \cdot \max\left\{ \epsilon_{\text{CR}}, 2^H \cdot 4n \cdot \epsilon_{\text{OW}} \right\} \tag{79}$$

$$t = \min\left\{ t_{\text{CR}}, t_{\text{OW}} \right\} - 2^H \cdot t_{\text{SIG}} - t_{\text{VER}} - t_{\text{GEN}}. \tag{80}$$

Note that we can replace t_{OTS} by t_{OW} rather than $t_{\text{OW}} - t_{\text{SIG}} - t_{\text{GEN}}$, since the time LD-OTS requires for signature and key generation is already included in the signature and key generation time of the MSS in Theorem 5. We also require $\epsilon_{\text{CR}} \leq 1/2$ and $\epsilon_{\text{OW}} \leq 1/(2^{H+1} \cdot 4n)$ to ensure $\epsilon \leq 1$.

To estimate the security level, we need explicit values for the key pair generation, signature generation and verification times of MSS using LD-OTS. We will use the following upper bounds.

$$t_{\text{GEN}} \leq 2^H \cdot 6n, \quad t_{\text{SIG}} \leq 4n(H+1), \quad t_{\text{VER}} \leq n + H$$

We also make assumptions for the values of $(t_{\text{CR}}, \epsilon_{\text{CR}})$ and $(t_{\text{OW}}, \epsilon_{\text{OW}})$. We distinguish between attacks that use classic computers only and attacks with quantum computers.

Using classical computers

In our security analysis of MSS we assume that the hash functions under consideration have output length n and only admit generic attacks against their preimage and collision resistance. Those generic attacks are exhaustive search and the birthday attack. When classical computers are used, then a birthday attack that inspects $2^{n/2}$ hash values has a success probability of approximately $1/2$. Also, an exhaustive search of $2^{n/2}$ random strings yields

a preimage of a given hash value with probability $1/2^{n/2}$. Therefore, we assume that the hash function family $\mathcal{G}$ is $(2^{n/2}, 1/2)$ collision resistant and $(2^{n/2}, 1/2^{n/2})$ preimage resistant. In this situation, we prove the following theorem.

Theorem 6 (Classic case). *The security level of the Merkle signature scheme combined with the Lamport-Diffie one-time signature scheme is at least*

$$b = n/2 - 1 \tag{81}$$

if the height of the Merkle tree is at most $H \leq n/3$ and the output length of the hash function is at least $n \geq 87$.

To prove Theorem 6 we use our assumption and Equations (79) and (80) and obtain the following estimate for the security level.

$$\frac{t}{\epsilon} \geq \frac{2^{n/2} - 2^H \cdot t_{\text{SIG}} - t_{\text{VER}} - t_{\text{GEN}}}{2 \cdot \max\{1/2, 2^H \cdot 4n \cdot 1/2^{n/2}\}}. \tag{82}$$

Using $H \leq n/3$, the maximum in the denominator is $1/2$ as long as

$$n/3 \leq n/2 - \log_2 4n - 1 \tag{83}$$

which holds for $n \geq 53$. Using the upper bounds for t_{SIG}, t_{VER}, and t_{GEN} estimated above, Equation (82) implies

$$\frac{t}{\epsilon} \geq 2^{n/2} - 2^H \cdot 4n(H+1) - (n+H) - 2^H \cdot 6n. \tag{84}$$

Using $H \leq n/3$, the desired lower bound for the security level of $2^{n/2-1}$ holds as long as

$$2^{n/3}(4/3 \cdot n^2 + 4n) + 4/3 \cdot n + 2^{n/3} \cdot 6n \leq 2^{n/2-1} \tag{85}$$

which is true for $n \geq 87$.

Using quantum computers

Again, we assume that our hash functions only admit generic attacks against their collision and preimage resistance. However, when quantum computers are available, the Grover algorithm [13] can be used in those generic attacks. Grovers algorithm requires $2^{n/3}$ evaluations of the hash function to find a collision with probability at most $1/2$. So we assume that our hash functions are $(2^{n/3}, 1/2)$ collision resistant. Also as explained in Remark 3 of Section 5 in Chapter 2 "Quantum computing", we may by virtue of Grover's algorithm assume that our hash functions are $(2^{n/3}, 1/2^{n/3})$ preimage resistant. In this situation, we prove the following theorem.

Theorem 7 (Quantum case). *The security level of the Merkle signature scheme combined with the Lamport-Diffie one-time signature scheme is at least*

$$b = n/3 - 1 \tag{86}$$

if the height of the Merkle tree is at most $H \leq n/4$ *and the output length of the hash function is at least* $n \geq 196$.

To prove Theorem 7 we use the same approach as for the proof of Theorem 6. We use our assumption on the hash function and Equations (79) and (80) and obtain the following estimate for the security level.

$$\frac{t}{\epsilon} \geq \frac{2^{n/3} - 2^H \cdot t_{\text{SIG}} - t_{\text{VER}} - t_{\text{GEN}}}{2 \cdot \max\{1/2, 2^H \cdot 4n \cdot 1/2^{n/3}\}}. \tag{87}$$

Using $H \leq n/4$, the maximum in the denominator is $1/2$ as long as

$$n/4 \leq n/3 - \log_2 4n - 1 \tag{88}$$

which holds for $n \geq 119$. Using the upper bounds for t_{SIG}, t_{VER}, and t_{GEN} estimated above, Equation (87) implies

$$\frac{t}{\epsilon} \geq 2^{n/3} - 2^H \cdot 4n(H+1) - (n+H) - 2^H \cdot 6n. \tag{89}$$

Using $H \leq n/4$, the desired lower bound for the security level of $2^{n/3-1}$ holds as long as

$$2^{n/4}(n^2 + 4n) + 5/4 \cdot n + 2^{n/4} \cdot 6n \leq 2^{n/3-1} \tag{90}$$

which is true for $n \geq 196$.

Comparison of the bit security

Table 2 shows the security level for some output lenghts n of the hash function. This table also shows the maximum value for H such that the security level holds.

Table 2. Security level of the Merkle signature scheme combined with the Lamport–Diffie one-time signature scheme in bits.

Output length n	128	160	224	256	384	512
Classic case						
bit security b	63	79	111	127	191	255
Maximum value for H	42	53	74	85	128	170
Quantum case						
bit security b	–	–	73	84	127	169
Maximum value for H	–	–	56	64	96	128

This table shows, that state-of-the-art hash functions can be used to ensure a high security level of the Merkle signature scheme, even against attacks by quantum computers. For all practical applications the maximum height of the Merkle tree and the resulting number of messages that can be signed with one key pair is sufficiently large.

References

1. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT'94*, LNCS 950, pages 92–111. Springer, 1995.
2. Berman, P., Karpinski, M., Nekrich, Y.: Optimal Trade-Off for Merkle Tree Traversal. Theoretical Computer Science, volume 372, issue 1, pages 26–36, 2007.
3. Buchmann, J., Coronado, C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS – an improved Merkle signature scheme. In *Progress in Cryptology - INDOCRYPT 2006*, LNCS 4329, pages 349–363. Springer-Verlag, 2006.
4. Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle signatures with virtually unlimited signature capacity. In *Applied Cryptography and Network Security - ACNS 2007*, LNCS 4521, pages 31–45. Springer, 2007.
5. Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. 2nd International Workshop on Post-Quantum Cryptography - PQCrypto 2008, LNCS 5299, pages 63–77. Springer, 2008.
6. Boneh, D., Mironov, I., Shoup, V.: A secure signature scheme from bilinear maps. In *Topics in Cryptology - CT-RSA 2003*, LNCS 2612, pages 98–110. Springer, 2003.
7. Coppersmith, D., Jakobsson, M.: Almost Optimal Hash Sequence Traversal. Financial Crypto '02. Available at `www.markus-jakobsson.com`.
8. Coronado, C.: On the security and the efficiency of the Merkle signature scheme. Cryptology ePrint Archive, Report 2005/192, 2005. `http://eprint.iacr.org/`.
9. Dahmen, E., Okeya, K., Takagi, T., Vuillaume, C.: Digital Signatures out of Second-Preimage Resistant Hash Functions. 2nd International Workshop on Post-Quantum Cryptography - PQCrypto 2008, LNCS 5299, pages 109–123. Springer, 2008.
10. Dods, C., Smart, N., Stam, M.: Hash based digital signature schemes. In *Cryptography and Coding*, LNCS 3796, pages 96–115. Springer, 2005.
11. ElGamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. Advances in Cryptology – CRYPTO '84, LNCS 196, pages 10–18. Springer, 1985.
12. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. In SIAM Journal on Computing, 17(2), pages 281–308, 1988.
13. Grover, L. K.: A fast quantum mechanical algorithm for database search. Proceedings of the Twenty-Eighth Annual Symposium on the Theory of Computing, pages 212–219, New York, 1996. ACM Press.
14. Jakobsson, M.: Fractal Hash Sequence Representation and Traversal. ISIT '02, p. 437. Available at `www.markus-jakobsson.com`.

15. Johnson, D. and Menezes, A.: The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical Report CORR 99-34, University of Waterloo, 1999. Available at http://www.cacr.math.uwaterloo.ca.
16. Jakobsson, M., Leighton, T., Micali, S., Szydlo, M.: Fractal Merkle Tree Representation and Traversal. In RSA Cryptographers Track, RSA Security Conference 2003.
17. Jutla, C., Yung, M.: PayTree: Amortized-Signature for Flexible Micropayments. 2nd USENIX Workshop on Electronic Commerce, pp. 213–221, 1996.
18. Lamport, L.: Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.
19. Lipmaa, H.: On Optimal Hash Tree Traversal for Interval Time-Stamping. In Proceedings of Information Security Conference 2002, LNCS 2433, pp. 357–371, Springer, 2002. Available at www.tcs.hut.fi/ ~ helger/papers/lip02a/.
20. Malkin, T., Micciancio, D., Miner, S.: Efficient Generic Forward-Secure Signatures With An Unbounded Number Of Time Periods. Proceedings of Eurocrypt '02, pages 400–417.
21. Merkle, R.C.: Secrecy, Authentication, and Public Key Systems. UMI Research Press, 1982. Also appears as a Stanford Ph.D. thesis in 1979.
22. Merkle, R.C.: A Digital Signature Based on a Conventional Encryption Function. Proceedings of Crypto '87, pp. 369–378.
23. Merkle, R.C.: A certified digital signature. Advances in Cryptology - CRYPTO '89 Proceedings, LNCS 435, pages 218–238, Springer, 1989.
24. Micali, S.: Efficient Certificate Revocation. In RSA Cryptographers Track, RSA Security Conference 1997, and U.S. Patent No. 5,666,416.
25. Naor, D., Shenhav, A., Wool, A.: One-time signatures revisited: Have they become practical. Cryptology ePrint Archive, Report 2005/442, 2005. http://eprint.iacr.org/.
26. Naor, D., Shenhav, A., Wool, A.: One-time signatures revisited: Practical fast signatures using fractal merkle tree traversal. IEEE – 24th Convention of Electrical and Electronics Engineers in Israel, pages 255–259, 2006.
27. Perrig, A., Canetti, R., Tygar, D., Song, D.: The TESLA Broadcast Authentication Protocol. Cryptobytes, Volume 5, No. 2 (RSA Laboratories, Summer/Fall 2002), pages 2–13. Available at www.rsasecurity.com/rsalabs/cryptobytes/.
28. Rompel, J.: One-way Functions are Necessary and Sufficient for Secure Signatures. Proceedings of ACM STOC'90, pages 387–394, 1990.
29. Rivest, R., Shamir, A.: PayWord and MicroMint–Two Simple Micropayment Schemes. CryptoBytes, Volume 2, No. 1 (RSA Laboratories, Spring 1996), pp. 7–11. Available at www.rsasecurity.com/rsalabs/cryptobytes/.
30. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption - FSE 2004*, LNCS 3017, pages 371–388. Springer, 2004.
31. Rivest, R. L., Shamir, A., and Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21(2):120–126, 1978.
32. FIPS PUB 180-1, Secure Hash Standard, SHA-1. Available at www.itl.nist.gov/fipspubs/fip180-1.htm.
33. Szydlo, M.: Merkle Tree Traversal in Log Space and Time. Advances in Cryptology - EUROCRYPT 2004, LNCS 3027, pages 541–554, Springer, 2004

34. Szydlo, M.: Merkle Tree Traversal in Log Space and Time. Preprint, available at www.szydlo.com, 2003.

Code-based cryptography

Raphael Overbeck[1] and Nicolas Sendrier[2]

[1] EPFL, I&C, LASEC.
[2] INRIA Rocquencourt, projet SECRET.

1 Introduction

In this chapter, we consider the theory and the practice of code-based cryptographic systems. By this term, we mean the cryptosystems in which the algorithmic primitive (the underlying one-way function) uses an error correcting code $\mathcal{C}$. This primitive may consist in adding an error to a word of $\mathcal{C}$ or in computing a syndrome relatively to a parity check matrix of $\mathcal{C}$.

The first of those systems is a public key encryption scheme and it was proposed by Robert J. McEliece in 1978 [48]. The private key is a random binary irreducible Goppa code and the public key is a random generator matrix of a randomly permuted version of that code. The ciphertext is a codeword to which some errors have been added, and only the owner of the private key (the Goppa code) can remove those errors. Three decades later, some parameter adjustment have been required, but no attack is known to represent a serious threat on the system, even on a quantum computer.

Similar ideas have been used to design other cryptosystems. Among others, let us mention some public key systems, like the Niederreiter encryption scheme [52] or the CFS signature scheme [14], and also identification schemes [73, 76], random number generators [19, 30] or a cryptographic hash function [3]. Some of the most important of those proposals are reviewed in §2.

As for any class of cryptosystems, the practice of code-based cryptography is a trade-off between security and efficiency. Those issues are well understood, at least for McEliece's scheme. Even though, no practical application of code-based cryptography is known to us. This might partly be due to the large size of the public key (100 kilobytes to several megabytes), but maybe also to a lack of publicity in a context were alternative solution were not urgently needed. Anyway, apart from the key size that we already mentioned, the McEliece encryption scheme has many strong features. First, the security reductions are tight (see [38] for instance). Also, the system is very fast, as both encryption and decryption procedures have a low complexity.

We will discuss in details the two aspects of security in §3 and §4. The first security assumption is the hardness of decoding in a random linear code [6]. This is an old problem of coding theory for which only exponential time solutions are known [4]. The second security assumption, needed only for public key systems, is the indistinguishability of Goppa codes [66]. Though it is not as old, in this form, as the first one, it relates to old problems of algebraic coding theory and is believed to be valid.

We will conclude this chapter with some practical aspects, first on the implementation, then on the key size issue, and we finish with a key point for the practicality of McEliece and related systems: how to efficiently construct a semantically secure (IND-CCA2) variant.

2 Cryptosystems

The first cryptosystem based on coding theory was a public key encryption scheme, presented in 1978 by McEliece [48]. Nearly all subsequently proposed asymmetric cryptographic schemes based on coding theory have a common disadvantage: the large memory requirements. Several other schemes followed, as the identification scheme by Stern [73], hash functions [3], random number generators [19] and efforts to build a signature scheme. The latter however all failed (compare [79], [32], [1] and [74]), until finally in 2001 Courtois, Finiasz and Sendrier made a promising proposal [14]. However, even if the latter is not broken, it is not suited for standard applications since besides the public key sizes the signing costs are large for secure parameter sets.

In 1986, Niederreiter proposed a knapsack-type PKC based on error correcting codes. This proposal was later shown to have a security equivalent to McEliece's proposal [42]. Among others, Niederreiter estimated GRS codes as suitable codes for his cryptosystem which were assumed to allow smaller key sizes than Goppa codes. Unfortunately, in 1992 Sidelnikov and Shestakov were able to show that Niederreiter's proposal to use GRS codes is insecure. In the following a couple of proposals were made to modify McEliece's original scheme (see e.g. [27], [26], [28], [70] and [35]) in order to reduce the public key size. However, most of them turned out to be insecure or inefficient compared to McEliece's original proposal (see e.g. [54] or [38]). The most important modifications for McEliece's scheme are the conversions by Kobara and Imai in 2001. These are CCA2-secure, provably as secure as the original scheme [37] and have almost the same transmission rate as the original system.

The variety of possible cryptographic applications provides sufficient motivation to have a closer look at cryptosystems based on coding theory as an serious alternative to established PKCs like the ones based on number theory. In this section we will concentrate on the most important cryptographic schemes based on coding theory.

2.1 McEliece PKC

The McEliece cryptosystem we are going to present in this section remains unbroken in its original version, even if about 15 years after it's proposal security parameters had to be adapted. Although the secret key of the McEliece PKC is a Goppa code (see §6.2) in the original description, the secret key could be drawn from any subclass of the class of alternant codes. However, such a choice might not reach the desired security as we will see in the following sections. The trapdoor for the McEliece cryptosystem is the knowledge of an efficient error correcting algorithm for the chosen code class (which is available for each Goppa code) together with a permutation. The McEliece PKC is summarized in Algorithm 2.1.

Algorithm 2.1 The McEliece PKC

- **System Parameters:** $n, t \in \mathbb{N}$, where $t \ll n$.
- **Key Generation:** Given the parameters n, t generate the following matrices:

  G : $k \times n$ generator matrix of a code $\mathcal{G}$ over $\mathbb{F}$ of dimension k and minimum distance $d \geq 2t+1$. (A binary irreducible Goppa code in the original proposal.)
  S : $k \times k$ random binary non-singular matrix
  P : $n \times n$ random permutation matrix

  Then, compute the $k \times n$ matrix $\mathsf{G}^{\text{pub}} = \mathsf{SGP}$.
- **Public Key:** $\left(\mathsf{G}^{\text{pub}}, t\right)$
- **Private Key:** $(\mathsf{S}, D_{\mathcal{G}}, \mathsf{P})$, where $D_{\mathcal{G}}$ is an efficient decoding algorithm for $\mathcal{G}$.
- **Encryption ($E_{(\mathsf{G}^{\text{pub}},t)}$):** To encrypt a plaintext $\mathbf{m} \in \mathbb{F}^k$ choose a vector $\mathbf{z} \in \mathbb{F}^n$ of weight t randomly and compute the ciphertext $\mathbf{c}$ as follows:

$$\mathbf{c} = \mathbf{m}\mathsf{G}^{\text{pub}} \oplus \mathbf{z} \ .$$

- **Decryption ($D_{(S,D_{\mathcal{G}},P)}$):** To decrypt a ciphertext $\mathbf{c}$ calculate

$$\mathbf{c}\mathsf{P}^{-1} = (\mathbf{m}\mathsf{S})\,\mathsf{G} \oplus \mathbf{z}\mathsf{P}^{-1}$$

  first, and apply the decoding algorithm $D_{\mathsf{G}^{\text{pub}}}$ for $\mathcal{G}$ to it. Since $\mathbf{c}\mathsf{P}^{-1}$ has a hamming distance of t to $\mathcal{G}$ we obtain the codeword

$$\mathbf{m}\mathsf{S}\mathsf{G} = D_{\mathcal{G}}\left(\mathbf{c}\mathsf{P}^{-1}\right) \ .$$

  Let $J \subseteq \{1, \cdots, n\}$ be a set, such that $\mathsf{G}^{\text{pub}}_{\cdot J}$ is invertible, then we can compute the plaintext $\mathbf{m} = (\mathbf{m}\mathsf{S}\mathsf{G})_J \left(\mathsf{G}_{\cdot J}\right)^{-1} S^{-1}$

The choice of security parameters for the McEliece PKC has to be taken in respect to the known attacks. The optimal choice of parameters for a given security level (in terms of the public key size) unfortunately can not be given as a closed formula. We are going to discuss the latter later on. The problem

to attack the McEliece PKC differs from the general decoding problem, which we will examine in §3:

Problem 1. (McEliece Problem) Let $\mathbb{F} = \{0,1\}$ and $\mathcal{G}$ be a binary irreducible Goppa code in Algorithm 2.1.

- Given a McEliece public key $\left(\mathsf{G}^{\text{pub}}, t\right)$ where $\mathsf{G}^{\text{pub}} \in \{0,1\}^{k\times n}$ and a ciphertext $\mathbf{c} \in \{0,1\}^n$,
- Find the (unique) message $\mathbf{m} \in \{0,1\}^k$ s.t. $\text{wt}\left(\mathbf{m}\mathsf{G}^{\text{pub}} - \mathbf{c}\right) = t$.

It is easy to see that someone who is able to solve the Syndrome Decoding Problem (compare §3) is able to solve the McEliece problem. The reverse is presumably not true, as the code $\mathcal{G} = \left\langle \mathsf{G}^{\text{pub}} \right\rangle$ is not a random one, but permutation equivalent to a code of a known class (a Goppa code in our definition). We can not assume that the McEliece-Problem is $\mathcal{NP}$-hard. Solving the McEliece-Problem would only solve the General Decoding Problem in a certain class of codes and not for all codes.

In the case of McEliece's original proposal, Canteaut and Chabaud state the following: "The row scrambler S has no cryptographic function; it only assures for McEliece's system that the public matrix is not systematic otherwise most of the bits of the plain-text would be revealed" [11]. However, for some variants of McEliece's PKC, this statement is not true, as e.g. in the case of CCA2-secure variants (see §5.1 and §5.3) or in the case, where the messages are seeds for PRNGs. The matrix P is indispensable because for most codes the code positions are closely related to the algebraic structure of the code. We will come back to this in §4.3.

The Niederreiter variant

The dual variant of the McEliece PKC is a knapsack-type cryptosystem and is called the Niederreiter PKC. In difference to the McEliece cryptosystem, instead of representing the message as a codeword, Niederreiter proposed to encode it into the error vector by a function $\phi_{n,t}$:

$$\phi_{n,t} : \{0,1\}^{\ell} \to \mathcal{W}_{n,t}, \tag{1}$$

where $\mathcal{W}_{n,t} = \{\mathbf{e} \in \mathbb{F}_2^n \mid \text{wt}(\mathbf{e}) = t\}$ and $\ell = \lfloor \log_2 |\mathcal{W}_{n,t}| \rfloor$. Such a mapping is presented, e.g., in [19] and is summarized in Algorithm 2.2. This algorithm is quite inefficient and has complexity $\mathcal{O}(n^2 \cdot \log_2 n)$. Its inverse is easy to define: $\phi_{n,t}^{-1}(\mathbf{e}) = \sum_{i=1}^{n} e_i \cdot \binom{i}{\sum_{j=0}^{i} e_j}$. We discuss efficient alternatives in §5.1. Representing the message by the error vector, we get the dual variant of McEliece's cryptosystem, given in Algorithm 2.3. The security of the Niederreiter PKC and the McEliece PKC are equivalent. An attacker who can break one is able to break the other and vice versa [42]. In the following, by "Niederreiter PKC" we refer to the dual variant of the McEliece PKC and to the proposal by Niederreiter to use GRS codes by "GRS Niederreiter PKC".

Algorithm 2.2 $\phi_{n,t}$: Mapping bit strings to constant weight codewords

Input: $\mathbf{x} \in \{0,1\}^{\ell}$
Output: a word $\mathbf{e} = (e_1, e_2, \cdots, e_n)$ of weight w and length n.

$c \leftarrow \binom{n}{w}$, $c' \leftarrow 0$, $j \leftarrow n$.
$i \leftarrow$ Index of $\mathbf{x}$ in the lexicographic order (an integer).
while $j > 0$ **do**
  $c' \leftarrow c \cdot \frac{j-w}{j}$
  if $i \leq c'$ **then**
    $e_j \leftarrow 0$, $c \leftarrow c'$
  else
    $e_j \leftarrow 1$, $i \leftarrow i - c'$, $c \leftarrow c \cdot \frac{w}{n}$
  $j \leftarrow j - 1$

Algorithm 2.3 Niederreiter's PKC

- **System Parameters:** n, $t \in \mathbb{N}$, where $t \ll n$.
- **Key Generation:** Given the parameters n, t generate the following matrices:

  H: $(n-k) \times n$ check matrix of a code $\mathcal{G}$ which can correct up to t errors.
  P: $n \times n$ random permutation matrix

  Then, compute the systematic $n \times (n-k)$ matrix $\mathsf{H}^{\text{pub}} = \mathsf{MHP}$, whose columns span the column space of HP, i.e. $\mathsf{H}^{\text{pub}}_{\{1,\cdots,n-k\}.} = \mathsf{Id}_{(n-k)}$.
- **Public Key:** $\left(\mathsf{H}^{\text{pub}}, t\right)$
- **Private Key:** $(\mathsf{P}, D_{\mathcal{G}}, \mathsf{M})$, where $D_{\mathcal{G}}$ is an efficient syndrome decoding algorithm[1] for $\mathcal{G}$.
- **Encryption:** A message $\mathbf{m}$ is represented as a vector $\mathbf{e} \in \{0,1\}^n$ of weight t, called plaintext. To encrypt it, we compute the syndrome

$$\mathbf{s} = \mathsf{H}^{\text{pub}}\mathbf{e}^{\top} \ .$$

- **Decryption:** To decrypt a ciphertext $\mathbf{s}$ calculate

$$\mathsf{M}^{-1}\mathbf{s} = \mathsf{HPe}^{\top}$$

  first, and apply the syndrome decoding algorithm $D_{\mathcal{G}}$ for $\mathcal{G}$ to it in order to recover $\mathsf{Pe}^{\top}$. Now, we can obtain the plaintext $\mathbf{e}^{\top} = \mathsf{P}^{-1}\mathsf{Pe}^{\top}$

[1] A syndrome decoding algorithm takes as input a syndrome – not a codeword. Each syndrome decoding algorithm leads immediately to an decoding algorithm and vice versa.

The advantage of this dual variant is the smaller public key size since it is sufficient to store the redundant part of the matrix H^{pub}. The disadvantage is the fact, that the mapping $\phi_{n,t}$ slows down en- and decryption. In a setting, where we want to send random strings, only, this disadvantage disappears as we can take $h(\mathbf{e})$ as random string, where h is a secure hash function.

Modifications for the trapdoor of McEliece's PKC

From McEliece's scheme one can easily derive a scheme with a different trapdoor by simply replacing the irreducible binary Goppa codes by another code class. However, such attempts often proved to be vulnerable against structural attacks. In §4.3 we will sketch a few of those attacks. To prevent structural attacks not only McEliece's proposal, but others exist as well. In Table 1 we give an overview of the principal modifications. McEliece's proposal can thus

1. **Row Scrambler [48]:** Multiply G with a random invertible matrix $\mathsf{S} \in \mathbb{F}^{k\times k}$ from the right. As $\langle \mathsf{G} \rangle = \langle \mathsf{SG} \rangle$, one can use the known error correction algorithm. Publishing a systematic generator matrix provides the same security against structural attacks as a random S.
2. **Column Scrambler / Isometry [48]:** Multiply G with a random invertible matrix $\mathsf{T} \in \mathbb{F}^{n\times n}$ from the left, where T preserves the norm, see §4.1. Obviously one can correct errors of norm up to t in $\langle \mathsf{GT} \rangle$, if G and T are known.
3. **Subcode [52]:** Let $0 < l < k$. Multiply G with a random matrix $\mathsf{S} \in \mathbb{F}^{l\times k}$ of full rank from the right. As $\langle \mathsf{SG} \rangle \subseteq \langle \mathsf{G} \rangle$, the known error correction algorithm may be used.
4. **Subfield Subcode [48]:** Take the $\mathbb{F}_{\mathrm{SUB}}$-subfield subcode of the secret code for a subfield $\mathbb{F}_{\mathrm{SUB}}$ of $\mathbb{F}$. As before, one can correct errors by the error correcting algorithm for the secret code. However, sometimes one can correct errors of larger norm in the subfield subcode than in the original code, compare Definition 9 and following.
5. **Matrix Concatenation [70]:** Take the code $\langle [\, \mathsf{G} | \mathsf{SG} \,] \rangle$ for an invertible matrix $\mathsf{S} \in \mathbb{F}^{k\times k}$. In Hamming norm, the secret key holder can correct $2t+1$ errors in this code, as he can correct errors in the first or the second n columns.[2]
6. **Random Redundancy [22]:** Add a number l of random columns at the left side of the matrix G. Errors can be corrected in the last n columns.
7. **Artificial Errors [27]:** One can choose to modify the matrix G at a small number of positions. These positions will be treated as erasures on decryption and thus change the norm t of the errors that can be decoded.
8. **Reducible Codes [26]:** Choose some matrices $\mathsf{Y} \in \mathbb{F}^{k\times n}$ and $\mathsf{S} \in \mathbb{F}^{l\times k}$ with $l \leq k$. Then take the code generated by

$$\left[\begin{array}{c|c} \mathsf{SG} & 0 \\ \hline \mathsf{Y} & \mathsf{G} \end{array}\right].$$

   Error correction by the algorithm for the secret code is possible if one corrects errors in sections, beginning from the right.[2] However, for correcting errors in Hamming metric, this approach does not seem to be suitable [56].

Table 1. Strategies for hiding the structure of a code

[2] One might generalize this approach by replacing one of the matrices G by a second secret code.

be seen as a combination of the strategies 1,2 and 4. Nevertheless, we have to remark, that all strategies have to be used with care, as they can but do not necessarily lead to a secure cryptosystem (compare e.g. [54, 78] and §4.3).

2.2 CFS signature

The only unbroken signature scheme based on the McEliece, or rather on the Niederreiter PKC was presented by Courtois, Finiasz and Sendrier in [14]. The security of the CFS scheme (against universal forgery) can be reduced to the hardness of Problem 1. The knowledge of the private key allows the decoder to solve this problem for a certain fraction of random words $\mathbf{c}$. The idea of the CFS algorithm is to repeatedly hash the document, randomized by a counter of bit-length r, until the output is a decryptable ciphertext. The signer uses his secret key to determine the corresponding error-vector. Together with the current value of the counter, this error vector will then serve as signature. The signature scheme is summarized in Algorithm 2.4.

The average number of attempts needed to reach a decodable syndrome can be estimated by comparing the total number of syndromes to the number of efficiently correctable syndromes:

$$\frac{\sum_{i=0}^{t}\binom{n}{t}}{2^{n-k}} = \frac{\sum_{i=0}^{t}\binom{n}{t}}{2^{mt}} \approx \frac{n^t/t!}{n^t} = \frac{1}{t!}$$

Thus each syndrome has a probability of $\frac{1}{t!}$ to be decodable, which can be tested in about t^2m^3 binary operations, see §6.1. The CFS scheme needs about $t^2m^3t!$ operations to generate a signature [14] and produces signatures of length $\log_2(r\binom{n}{t}) \approx \log_2(n^t)$. Thus, r has to be be larger than $\log_2(t!)$. The signature length $(n+r)$ can be reduced considerably, by employing a mapping like $\phi_{n,t}$.

With the parameters suggested by Courtois, Finiasz and Sendrier ($m = 16, t = 9$) the number of possible error-vectors is approximately given by $\binom{n}{t} = \binom{2^{16}}{9} \approx 2^{125.5}$ so that a 126-bit counter suffices to address each of them. However, these parameters are too low to prevent a generalized collision attack, see §3.4. As the CFS scheme does not scale well with growing parameters, secure instances of the CFS scheme require huge public keys.

2.3 Stern's identification scheme

Stern's identification scheme presented in 1994 is closely related to the Niederreiter cryptosystem. There exists a variant of this scheme by Pascal Véron [76]. However, we will explain the original scheme: Let H^{pub} be a $(n-k)\times n$ matrix common to all users. If H^{pub} is chosen randomly, it will provide a parity check matrix for a code with asymptotically good minimum distance given by the Gilbert-Varshamov (GV) bound, see Definition 1. The private key for a user

Algorithm 2.4 CFS digital signature

- **System parameters:** $m, t \in \mathbb{N}$.
- **Key Generation:** Generate a Niederreiter PKC key pair with a code drawn from the class of $[n = 2^m, k = n - mt, 2t + 1]$ binary irreducible Goppa codes.
- **Signing:**

  Input: h a public hash function, $\phi_{n,t}$, $D_{(S,D_{\mathcal{G}},P)}$, $r \in \mathbb{N}_+$ and the document d to be signed
  Output: A CFS-signature $\mathbf{s}$.

  $\mathbf{z} = h(d)$
  choose a r-bit Vector $\mathbf{i}$ at random
  $\mathbf{s} = h(\mathbf{z}||\mathbf{i})$
  while $\mathbf{s}$ is not decodable **do**
    choose a r-bit Vector $\mathbf{i}$ at random
    $\mathbf{s} = h(\mathbf{z}||\mathbf{i})$
  $\mathbf{e} = D_{(S,D_{\mathcal{G}},P)}(\mathbf{s})$
  $\mathbf{s} = (\phi_{n,t}^{-1}(\mathbf{e})||\mathbf{i})$
- **Verification:**

  Input: A signature $\mathbf{s} = (\phi_{n,t}^{-1}(\mathbf{e})||\mathbf{i})$, the document d and $\mathsf{H}^{\mathrm{pub}}$
  Output: accept or **reject**

  $\mathbf{e} = \phi_{n,t}(\phi_{n,t}^{-1}(\mathbf{e}))$
  $\mathbf{s}_1 = \mathsf{H}^{\mathrm{pub}}(\mathbf{e}^\top)$
  $\mathbf{s}_2 = h(h(d)||\mathbf{i})$
  if $\mathbf{s}_1 = \mathbf{s}_2$ **then**
    accept $\mathbf{s}$
  else
    reject $\mathbf{s}$

will thus be a word $\mathbf{e}$ of low weight w (e.g. $w \approx$ GV bound), which sums up to the syndrome $\mathbf{e}\mathsf{H} = \mathbf{s}$, the public key. By Stern's 3-pass zero-knowledge protocol (Algorithm 2.5), the secret key holder can prove his knowledge of $\mathbf{e}$ using two blending factors: a permutation and a random vector. However, a dishonest prover not knowing $\mathbf{e}$ can cheat the verifier in the protocol with probability 2/3. Thus, the protocol has to be run several times to detect cheating provers.

The security of the scheme relies on the difficulty of the general decoding problem, that is on the difficulty of determining the preimage $\mathbf{e}$ of $\mathbf{s} = \mathsf{H}^{\mathrm{pub}}\mathbf{e}^\top$. Without the secret key, an adversary has three alternatives to deceive the verifier:

1. To be able to answer the challenges $b \in \{1, 2\}$, the attacker commits to $c_1 = (\Pi, \mathsf{H}^{\mathrm{pub}}\mathbf{y}^\top + \mathbf{s})$ and selects a random vector $\hat{\mathbf{e}}$ of the same weight as $\mathbf{e}$. Now, he computes $c_2 = (\mathbf{y} + \hat{\mathbf{e}})\Pi$ and $c_3 = \Pi(\mathbf{y})$.
2. He can work with a random $\hat{\mathbf{e}}$ of weight w instead of the secret key while computing c_1, c_2, c_3. He will succeed if he is asked $b \in \{0, 2\}$ but in case

Algorithm 2.5 Stern's identification scheme

- **System parameters :** $n, k, q, w \in \mathbb{N}_+$ and $\mathsf{H}^{\text{pub}} \in \mathbb{F}_q^{(n-k)\times n}$.
- **Public key :** $\mathsf{H}^{\text{pub}}\mathbf{e}^\top = \mathbf{s} \in \mathbb{F}_q^{n-k}$
- **Private key :** $\mathbf{e} \in \mathbb{F}_q^n$ of weight w.

Prover	**Verifier**
Choose random n-bit vector $\mathbf{y}$ and random permutation $\mathsf{\Pi}$, to compute $c_1 = (\mathsf{\Pi}, \mathsf{H}^{\text{pub}}\mathbf{y}^\top)$, $c_2 = \mathbf{y}\mathsf{\Pi}$, $c_3 = (\mathbf{y} + \mathbf{e})\mathsf{\Pi}$. Send commitments for (c_1, c_2, c_3)	
	Send random request $b \in \{0, 1, 2\}$
If $b = 0 \Rightarrow$ reveal $c_2, \mathsf{\Pi}$ If $b = 1 \Rightarrow$ reveal $c_3, \mathsf{\Pi}$ If $b = 2 \Rightarrow$ reveal c_2, c_3	
	If $b = 0 \Rightarrow$ check c_1, c_2 If $b = 1 \Rightarrow$ check c_1, c_3 with $\mathsf{H}^{\text{pub}}\mathbf{y}^\top = \mathsf{H}^{\text{pub}}(\mathbf{y} + \mathbf{e})^\top + \mathbf{s}$ If $b = 2 \Rightarrow$ check c_2, c_3 and the weight of $\mathbf{e}\mathsf{\Pi}$.

$b = 1$ he will not be able to produce the correct c_1, c_3 since $\mathsf{H}^{\text{pub}}\hat{\mathbf{e}}^\top \neq \mathsf{H}^{\text{pub}}\mathbf{e}^\top = \mathbf{s}$.

3. He can choose $\hat{\mathbf{y}}$ of arbitrary weight from the set of all possible preimages of $\mathbf{s}$ and replaces $\mathbf{e}$ by $\hat{\mathbf{y}}$ while computing c_1, c_2, c_3. This time he will fail to answer the request $b = 2$ since $\text{wt}(\hat{\mathbf{y}}) \neq w$.

The communication cost per round is about $n(\log_2(q) + \log_2(n))$ plus three times the size of the employed commitments (e.g. a hash function).

The standard method to convert the identification procedure into a procedure for signing, is to replace verifier-queries by values suitably derived from the commitments and the message to be signed. This leads to a blow-up of each (hashed) plaintext bit to more than $(n[\log_2(q) + \log_2(n)])/\log_2(3)$ signature bits and is therefore of theoretical interest as a signature. However, the security of the resulting signature scheme can be reduced to the average-case hardness of the $\mathcal{NP}$-hard general decoding problem in the random oracle model.

2.4 Cryptosystems based on the syndrome one-way function

Besides the classical code based PKCs there exist other cryptographic primitives with security reductions to coding theoretic problems. For symmetric cryptosystems we do not need a trapdoor and can take the computation of a syndrome of a random code as a one-way function. In this section we want to give a way of obtaining cryptographic strong hashing and generation of pseudorandom sequences using coding theoretic primitives.

Code based hashing

If in Stern's identification scheme parameters are chosen properly, one has the following inequality:

$$\binom{n}{w}(q-1)^{w-1} \cdot q^{k-n} \geq 1.$$

Thus, there are more vectors of weight w and length n than syndromes of an $[n, k]$ code. If it is still hard to recover vectors of weight w in the set of vectors with a certain syndrome, then, computing syndromes can serve as a compression function. Based on this compression function, a hash function can be constructed [3]. The compression function is realized by $\mathbf{x} \mapsto \phi_{n,w}(\mathbf{x})\mathsf{H}$, with $\phi_{n,w}$ given in Algorithm 2.2. In Figure 1 we give an intuition of the way the hash function works.

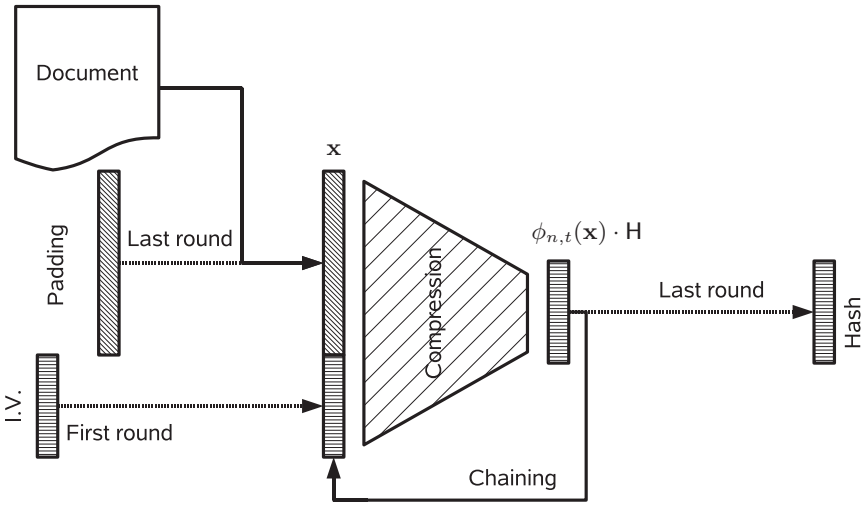

Fig. 1. Merkle-Damgård scheme of hash functions

The performance of such a hash function depends on the time needed to compute the one-to-one mapping $\phi_{n,w}$. In order to speed-up such hash function, one can for example limit the set $\mathcal{W}_{n,t}$ to the set of w'*-regular words*

$$\mathcal{W}'_{n,t} = \left\{(\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_{w/w'}) \in \mathbb{F}_q^n \mid \mathbf{e}_i \in \mathbb{F}_q^{n \cdot w'/w}, \mathrm{wt}(\mathbf{e}) = w'\right\}$$

if $w'|w$ and $(w'/w)|n$. The modified mapping $\phi_{n,w}^{w'}$ is easy to compute, if $w' = 1$. The resulting compression function is $\mathbf{x} \mapsto \phi_{n,w}^{w'}(\mathbf{x})\mathsf{H}$. Nevertheless, using regular words changes the problem of inverting the compression function. Even if it was proved, that inverting $\phi_{n,t}^{w'}(\mathbf{x})\mathsf{H}$ is $\mathcal{NP}$-hard in general, there

is no evidence if it is weak or hard in the average case [3]. Further, it was only proved, that finding preimages for $\phi_{n,t}^{w'}(\mathbf{x})\mathsf{H}$ is $\mathcal{NP}$-hard in the cases $w' \in \{1, 2\}$, but not the problem to find collisions.

For the chaining step of a hash function one possibility is obviously to concatenate the syndrome obtained with the input of the next round and to apply $\phi_{n,w}$ afterwards. In the case of w'-regular words with blocklengths $n \cdot w'/w$, there exists a second possibility: One can simply concatenate two such words of length $< n$ to obtain a new w'-regular word of length n and weight w.

One possible choice is to use $q = 2, w' = 1$ with parameters $n = 2^{14}$, $n - k = 160$ and $w = 64$ for a moderately ($2^{62.2}$) secure hash function and $n = 3 \cdot 2^{13}$, $n - k = 224$ and $w = 96$ for a ($2^{82.3}$) secure version. For more parameter proposals and comparison with other hash functions we refer to [3]. An attack against the collision resistance of the hash function is presented in §3.4.

Cryptographically strong random numbers

If in Stern's identification scheme parameters are chosen such that

$$\binom{n}{w}(q-1)^{w-1} \cdot q^{k-n} \leq 1,$$

there are less vectors of weight w and length n than syndromes of an $[n, k]$ code. If it is still hard to recover vectors of weight w in the set of vectors with a certain syndrome, then, computing syndromes can serve as a expansion function and thus to generate pseudorandom sequences [19]. Figure 2 gives an intuition of the way the pseudo random number generator (PRNG) works. For security reasons, we propose to use the same parameters as in Stern's

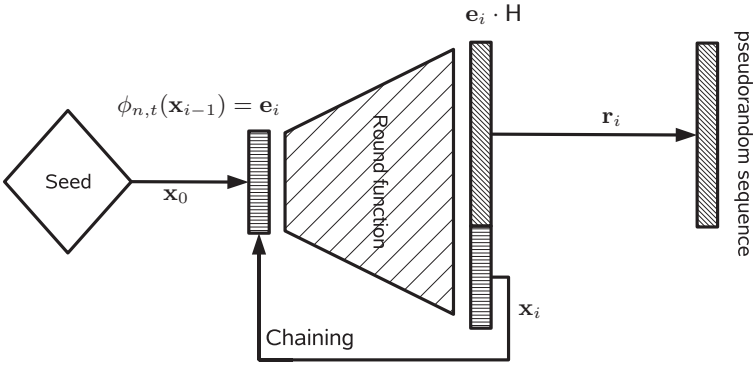

Fig. 2. Scheme of code based PRNG

identification scheme, see §2.3. Here again, w'-regular words can be used to speed-up the PRNG.

3 The security of computing syndromes as one-way function

In this section we consider the message security (opposed to key security) of code-based cryptosystems. We assume the attacker has no information on the algebraic structure of the underlying error correcting code, either because the trapdoor is sufficiently well hidden (public key systems) or because there is no trapdoor (code-based one way functions). This means correcting errors in a linear code for which one knows only a generator (or a parity check) matrix.

Unless specified otherwise, the codes we consider in this section have a binary alphabet. It is sufficient for most cryptosystems of interest. Moreover, most statements can be generalized to a larger alphabet.

3.1 Preliminaries

We consider a binary linear code $\mathcal{C}$ of length n and dimension k. We denote $r = n - k$ the codimension of $\mathcal{C}$ and H a parity check matrix of $\mathcal{C}$. We define a syndrome mapping relatively to H

$$\begin{array}{rccc} S_{\mathsf{H}} : & \{0,1\}^n & \longrightarrow & \{0,1\}^r \\ & \mathbf{y} & \longmapsto & \mathbf{y}\mathsf{H}^\top \end{array}$$

For any $\mathbf{s} \in \{0,1\}^r$, we denote the set of words of $\{0,1\}^n$ with syndrome $\mathbf{s}$ by

$$S_{\mathsf{H}}^{-1}(\mathbf{s}) = \left\{\mathbf{y} \in \{0,1\}^n \mid \mathbf{y}\mathsf{H}^\top = \mathbf{s}\right\}.$$

By definition, we have $S_{\mathsf{H}}^{-1}(\mathbf{0}) = \mathcal{C}$ for any parity check matrix H of $\mathcal{C}$. The sets $\mathbf{y} + \mathcal{C}$, for all $\mathbf{y}$ in $\{0,1\}^n$, are called the *cosets* of $\mathcal{C}$. There are exactly 2^r different cosets which form a partition of $\{0,1\}^n$ (*i.e.* pairwise disjoint). For any parity check matrix H of $\mathcal{C}$, there is a one to one correspondence between cosets and syndromes relatively to H.

Proposition 1. *For any syndrome* $\mathbf{s} \in \{0,1\}^r$ *we have*

$$S_{\mathsf{H}}^{-1}(\mathbf{s}) = \mathbf{y} + \mathcal{C} = \{\mathbf{y} + \mathbf{x} \mid \mathbf{x} \in \mathcal{C}\},$$

where $\mathbf{y}$ *is any word of* $\{0,1\}^n$ *of syndrome* $\mathbf{s}$*. Moreover, finding such a word* $\mathbf{y}$ *from* $\mathbf{s}$ *(and* H*) can be achieved in polynomial time.*

For any $\mathbf{y}$ and $\mathbf{z}$ in $S_{\mathsf{H}}^{-1}(\mathbf{s})$, we have $\mathbf{y}\mathsf{H}^\top = \mathbf{z}\mathsf{H}^\top$, thus $(\mathbf{y} + \mathbf{z})\mathsf{H}^\top = 0$ and $\mathbf{y} + \mathbf{z} \in \mathcal{C}$. It follows that $S_{\mathsf{H}}^{-1}(\mathbf{s}) = \mathbf{y} + \mathcal{C}$.

To compute one particular element of $S_{\mathsf{H}}^{-1}(\mathbf{s})$, given $\mathbf{s}$, we will consider a systematic form H_0 of the parity check matrix H. That is a $r \times n$ binary matrix

H_0 of the form $[\mathsf{Id} \mid \mathsf{X}]$ (where Id is the $r \times r$ identity matrix and X is some $r \times k$ matrix) such that $\mathsf{H}_0 = \mathsf{UH}$, with U a $r \times r$ non-singular matrix. One can obtain such a matrix U in time $\mathcal{O}(r^3)$ by inverting the first r columns[3] of H. Let $\mathbf{y} = [\mathbf{s}\mathsf{U}^\top \mid \mathbf{0}] \in \{0,1\}^n$, since $(\mathsf{U}^\top)^{-1} = (\mathsf{U}^{-1})^\top$, we have

$$\mathbf{y}\mathsf{H}^\top = \mathbf{y}(\mathsf{U}^{-1}\mathsf{H}_0)^\top = \mathbf{y}\mathsf{H}_0^\top(\mathsf{U}^{-1})^\top = (\mathbf{s}\mathsf{U}^\top \mid \mathbf{0}) \begin{bmatrix} \mathsf{Id} \\ \mathsf{X}^\top \end{bmatrix} (\mathsf{U}^\top)^{-1} = \mathbf{s}.$$

The word $\mathbf{y}$ is in $S_{\mathsf{H}}^{-1}(\mathbf{s})$ and is obtained in polynomial time.

3.2 Decoding problems

Let $\mathcal{C}$ be a binary linear code of parity check matrix H. We are given a word $\mathbf{y} \in \{0,1\}^n$ and its syndrome $\mathbf{s} = \mathbf{y}\mathsf{H}^\top \in \{0,1\}^r$. Decoding consists of solving one of the following equivalent problems:

(i) Find a codeword $\mathbf{x} \in \mathcal{C}$ closest to $\mathbf{y}$ for the Hamming distance.
(ii) Find an error $\mathbf{e} \in \mathbf{y} + \mathcal{C}$ of minimal Hamming weight.
(iii) Find an error $\mathbf{e} \in S_{\mathsf{H}}^{-1}(\mathbf{s})$ of minimal Hamming weight.

In practice, given an instance of a decoding problem, it is difficult to check if the error $\mathbf{e}$ is really of *minimal* weight in the coset (or if the codeword $\mathbf{x}$ is really the *closest* to $\mathbf{y}$). Because of that, the decoding problem as stated above is not in $\mathcal{NP}$. Instead, we will consider a slightly different abstraction of the problem, called *syndrome decoding*:

Problem 2 (Computational Syndrome Decoding). Given a binary $r \times n$ matrix H, a word $\mathbf{s}$ in $\{0,1\}^r$ and an integer $w > 0$, find a word $\mathbf{e}$ in $S_{\mathsf{H}}^{-1}(\mathbf{s})$ of Hamming weight $\leq w$.

The value of the additional parameter w will significantly affect the difficulty (see §3.3) of the resolution. In the theory of error correcting codes the problem is meaningful only if w is such that the problem has a single solution with high probability (*i.e.* w is not greater than the Gilbert-Varshamov bound (Definition 1)). For cryptographic applications, any value of w such that the problem is hard may produce a one-way function.

Decades of practice indicate that syndrome decoding in an arbitrary linear code is difficult (see [4] for instance). In addition, the associated decision problem was proved $\mathcal{NP}$-complete in [6].

We will denote $\mathrm{CSD}(\mathsf{H}, w, \mathbf{s})$ a specific instance of the computational syndrome decoding problem. Note that there is no "gap" (as for problems related to Diffie-Hellman) between the decisional and the computational problems. In fact an attacker can solve any instance of CSD with a linear number of access to a decisional syndrome decoding oracle (this is the basis for the reaction attack, see §5.3).

[3] w.l.o.g. we can assume that the first r columns of H are full rank

The problem of finding non-zero words of small Hamming weight (say $\leq w$) in a given linear code is very similar, but not identical, to decoding. We can state it as follows

Problem 3 (Codeword Finding). Given a binary $r \times n$ matrix H and an integer $w > 0$, find a non-zero word of Hamming weight $\leq w$ in $S_{\mathsf{H}}^{-1}(\mathbf{0})$.

Though it looks similar, this is not a particular instance of CSD, because of the non-zero condition. In fact, if $\mathcal{C}$ is the linear code of parity check matrix H, then any solution of $\mathrm{CSD}(\mathsf{H}, w, \mathbf{y}\mathsf{H}^\top)$ is also a solution to $\mathrm{CF}(\mathsf{H}', w)$, where H' is a parity check matrix of the code $\mathcal{C}' = \langle \mathbf{y} + \mathcal{C} \rangle$ spanned by $\mathbf{y}$ and $\mathcal{C}$. The converse is true only if $w < \mathrm{dmin}(\mathcal{C})$, the minimum distance of $\mathcal{C}$.

The minimum distance is usually unknown. However most binary linear codes of length n and codimension r have a minimum distance very close to the Gilbert-Varshamov distance $d_0(n, r)$.

Definition 1. *[4] The Gilbert-Varshamov distance $d_0(n, r)$ (or simply d_0 when there is no ambiguity) is defined as the largest integer such that*

$$\sum_{i=0}^{d_0-1} \binom{n}{i} \leq 2^r.$$

Let H, $\mathbf{y}$ and H' be defined as above. Let $\mathbf{e}$ be a solution to $\mathrm{CF}(\mathsf{H}', w)$, independently of w we have

- if $\mathrm{wt}(\mathbf{e}) < d_0$, then $\mathbf{e}$ is very likely a solution to $\mathrm{CSD}(\mathsf{H}, w, \mathbf{y})$,
- if $\mathrm{wt}(\mathbf{e}) \geq d_0$, then $\mathbf{e}$ is a solution to $\mathrm{CSD}(\mathsf{H}, w, \mathbf{y})$ with probability $\approx 1/2$.

Those informal statements hold "in average" and come from the fact that $\mathcal{C}'$ the code spanned by $\mathbf{y}$ and $\mathcal{C}$ is equal to $\mathcal{C} \cup (\mathbf{y} + \mathcal{C})$. If the weight of $\mathbf{e} \in \mathcal{C}' = \mathcal{C} \cup (\mathbf{y} + \mathcal{C})$ is smaller than the minimum distance of $\mathcal{C}$ (which is likely to be close to d_0), then it belongs to the coset $\mathbf{y}+\mathcal{C}$. On the other hand if the weight of $\mathbf{e}$ is higher than d_0, then, if it is a random solution to $\mathrm{CF}(\mathsf{H}', w)$, it is equally likely[4] to be in $\mathcal{C}$ and in $\mathbf{y} + \mathcal{C}$. In practice, most general purpose decoders, and in particular those used in cryptanalysis, are in fact searching for small weight codewords.

In the problems we have stated so far, the target weight w is an input. In many cases of interest, the target weight will instead depend of the code parameters (length and dimension). This will happen in particular in two cases that we detail below: *Complete Decoding* and *Bounded Decoding*.

As we have seen earlier, decoding will consist in finding a word of minimal weight that produces a given syndrome. If the syndrome is random, then the solution is very likely to have a weight equal to the Gilbert-Varshamov distance. Decoding is thus likely to be as hard as the following problem.

[4] Metric properties of a random code are in practice indistinguishable from those of a random set with the same cardinality

Problem 4 (Complete Decoding). Given a binary $r \times n$ matrix H and a word $\mathbf{s}$ in $\{0,1\}^r$, find a word of Hamming weight $\leq d_0(n,r)$ in $S_{\mathsf{H}}^{-1}(\mathbf{s})$.

This is in fact the most general and the most difficult computational problem for given parameters n and r.

In a public key encryption scheme, like McEliece or Niederreiter, the target weight is much smaller as it will be equal to the error correcting capability of the underlying code. A Goppa code of length $n = 2^m$ and correcting t errors has codimension $r = tm$. A message attack on the McEliece encryption scheme will thus correspond to the following computational problem

Problem 5 (Goppa Bounded Decoding). Given a binary $r \times n$ matrix H and a word $\mathbf{s}$ in $\{0,1\}^r$, find a word of Hamming weight $\leq r/\log_2 n$ in $S_{\mathsf{H}}^{-1}(\mathbf{s})$.

The associated decision problem is $\mathcal{NP}$-complete [18]. This demonstrates that the above computational problem is $\mathcal{NP}$-hard, that is difficult in the worst case. Even though this doesn't say anything on the average case complexity, at least this proves that if we reduce the target weight to the error correcting capability of a Goppa code, we do not fall into an easy case.

3.3 Decoding algorithms

Information set[5] decoding is undoubtedly the technique that has attracted most of the cryptographer's attention. The best known decoding attacks on McEliece and Niederreiter are all derived from it. There have been other attempts but with a mitigated success (iterative decoding [20] or statistical decoding [34, 55]).

Algorithm 3.1 presents a generalized version of information set decoding. Lee and Brickell [39] were the first to use it to analyze the security of

---

Algorithm 3.1 Information set decoding (for parameter p)

- **Input**: a $k \times n$ matrix G, an integer w
- **Output**: a non-zero codeword of weight $\leq w$
- **Repeat**
  - Pick a $n \times n$ permutation matrix P.
  - Compute $\mathsf{G}' = \mathsf{UGP} = (\mathsf{Id} \mid \mathsf{R})$ (w.l.o.g. we assume the first k positions form an information set).
  - Compute all the sum of p rows or less of G', if one of those sums has weight $\leq w$ then stop and return it.

McEliece's PKC. In another context, computing the minimum distance of a

[5] An information set for a given code of dimension k, is a set of k positions such that the restriction of the code to those positions contains all the k-tuples exactly once. In particular, it means that the corresponding columns in any generator matrix are independent.

code, Leon [40] proposed an improvement by looking for codewords containing zeroes in a windows of size ℓ in the redundancy (right) part of the codeword. It was further optimized by Stern [72] by dividing the information set in two parts, allowing to speed-up the search for codewords with zeroes in the window by a birthday attack technique.

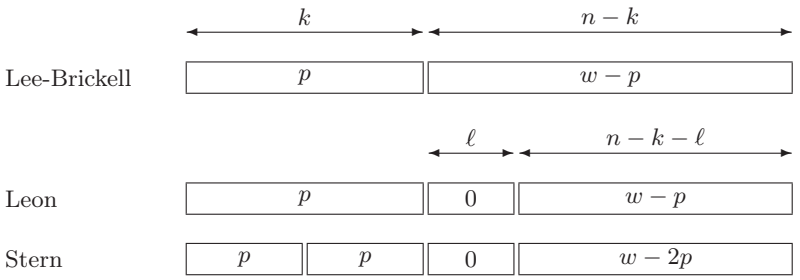

Fig. 3. Weight profile of the codewords sought by the various algorithms (the number inside the boxes is the Hamming weight of the corresponding tuples)

In Figure 3 we present the different weight profiles corresponding to a success, the probability of success of a given iteration is respectively

$$P_{LB} = \frac{\binom{k}{p}\binom{n-k}{w-p}}{\binom{n}{w}}, P_L = \frac{\binom{k}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}}, P_S = \frac{\binom{k/2}{p}^2\binom{n-k-\ell}{w-2p}}{\binom{n}{w}}.$$

The total cost of the algorithm is usually expressed as a *binary work factor*. It is equal to the cost (in binary operation) of an iteration divided by the above probability (*i.e.* multiplied by the expected number of iterations).

The Canteaut-Chabaud decoding algorithm

The best known variant was proposed by Canteaut and Chabaud [12] and is the Stern algorithm with another improvement due to van Tilburg [75] consisting in changing only one element of the information set at each iteration. The overall binary work factor is smaller, but it is much more difficult to evaluate as for every value of the parameters p and ℓ, the probability of success is obtained by computing the stationary distribution of a Markov process. It is nevertheless possible to exhibit a rather tight lower bound on its complexity. The probability of success of an iteration is upper bounded by the success probability P_S of Stern's algorithm and for any p the best value for ℓ is close to $\log_2 \binom{k/2}{p}$. Finally, we get the following lower bound on the binary work factor for Canteaut-Chabaud algorithm:

$$WF(n,k,w) \geq \min_p \left(\frac{K\ell}{2^\ell} \frac{\binom{n}{w}}{\binom{n-k-\ell}{w-2p}} \right) \text{ where } \ell = \log_2 \binom{k/2}{p}. \qquad (2)$$

In the above formula, K is a small constant (see the remark below) which also appears in the cost of Canteaut-Chabaud's algorithm. The space complexity in bits is lower bounded by $\ell 2^\ell$. The lower bound defined by (2) is close in practice (a factor 10 at most, see Figures 5 and 6) to the estimation given in [12] which requires the computation of the fundamental matrix of a Markov chain (inversion of a real matrix of size $t + p + 1$) for every value of p and ℓ.

Remark 1. The binary work factor gives a measure of the cost of the algorithm. It is in fact a lower bound on the average number of binary operations needed to solve a problem of given size. Dividing by 32 or 64 (minus 5 or 6 on the exponent) will give a lower bound on the number of CPU operations.

The actual computation time will depend on the relative cost of the various operations involved (sorting, storing, fetching, xoring, popcounting, ...) for a particular implementation and a particular platform. In formula (2), all of this is hidden in the constant K (for practical purposes we took $K = 3$).

Let us consider now how the work factor evolves with the error weight w. For fixed values of the code length n and dimension k, the maximal cost is obtained when w is equal to the Gilbert-Varshamov distance. When $w \leq d_0(n, n-k)$, the decoding cost is $2^{w(c+o(1))}$, where the constant c depends of the ratio k/n. Typical behavior for fixed n and k when t grows is given in Figure 4.

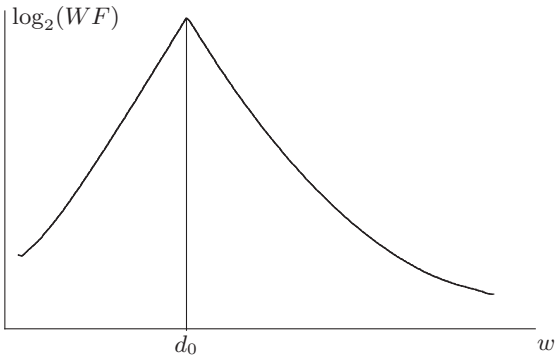

Fig. 4. Information set decoding running time (log scale) for fixed length and dimension when the error weight w varies

When w gets larger, the number of solutions grows very quickly. Formula (2) gives the cost for finding one specific codeword of weight w, the decoding cost is obtained by dividing this value by the expected number of solutions. For those values of w, information set decoding is not always the best technique, and other algorithms, like the generalized birthday attack [10, 77], may be more efficient (see §3.4).

Decoding attacks against McEliece

We consider binary Goppa codes. The length is $n = 2^m$ and the dimension is related with the error weight t, as $k = n - tm$.

In practice, the best value for parameter p in formula (2) is small. For length 2048 it is always equal to 2, and for length 4096, the best value of p varies between 2 and 5. Figures 5 and 6 (see also Table 2) give an estimate of the practical security of the McEliece cryptosystem when binary Goppa codes of length 2048 and 4096 are used. Note finally that the decoding (message) attacks are always more efficient than the structural (key) attacks (see §4.3).

3.4 Collision attacks against FSB and CFS

The fastest attacks on the CFS scheme and the FSB hash function are based on Wagner's solution for the generalized Birthday paradox. Wagner's main theorem can be seen as a generalization of the search part of Stern's algorithm for low weight code words or the algorithm of Patarin and Camion [10] and can be summarized as follows:

Theorem 1. (Generalized Birthday Problem) *Let $r, a \in \mathbb{N}$ with $(a+1)|r$ and $\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_{2^a} \subseteq \mathbb{F}_{2^r}$ be sets of cardinality $2^{\frac{r}{a+1}}$, then, a solution of the equation*

$$\sum_{i=1}^{2^a} x_i = \mathbf{0} \text{ where } x_i \in \mathcal{L}_i, \tag{3}$$

can be found in $\mathcal{O}(2^a 2^{\frac{r}{a+1}})$ Operations (over $\mathbb{F}_{2^r}$).

The algorithm proposed by Wagner is iterative: First, one searches for partial collisions of the sets $\mathcal{L}_i$ and $\mathcal{L}_{i+2^{a-1}}$, $i = 1, \cdots, 2^{a-1}$, that is, such pairs $(x_i, x_{i+2^{a-1}})$ that $LSB_{\frac{r}{a+1}}(x_i + x_{i+2^{a-1}}) = 0$. This way, one obtains 2^{a-1} Lists with approximately $2^{\frac{r}{a+1}}$ pairs, where the last $\frac{r}{a+1}$ entries are zero and can be omitted in the next step. A recursive application of this step leads to a solution of Equation (3).

As shown by J.-S. Coron and A. Joux, Wagner's solution for the generalized Birthday Paradox can be used to find collisions for the FSB hash. This is due to the fact, that the compression function of the FSB hash is inherently different to the one used by other hash functions: If we consider $\phi'_{n,t}(\mathbf{x}) \cdot \mathsf{H}$ (or $\phi_{n,t}(\mathbf{x}) \cdot \mathsf{H}$), we can see, that one collision ($\phi'_{n,t}(\mathbf{x}_1) \cdot \mathsf{H} = \phi'_{n,t}(\mathbf{x}_2) \cdot \mathsf{H}$) leads to up to $\binom{w}{w/2}$ further collisions. As the mapping $\phi_{n,t}$ can be easily inverted and the second part of the compression function is linear, we can apply Wagner's theorem employing a "divide and conquer"-strategy.

Applied to the FSB hash function we obtain the following attack against the collision resistance in the case of $\phi'_{n,t}$ as compression function: Each list $\mathcal{L}_1, \cdots, \mathcal{L}_{2^a}$ is designed to contain the syndromes of 2-quasiregular words $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_w)$, such that for all $i \neq j$ and γ:

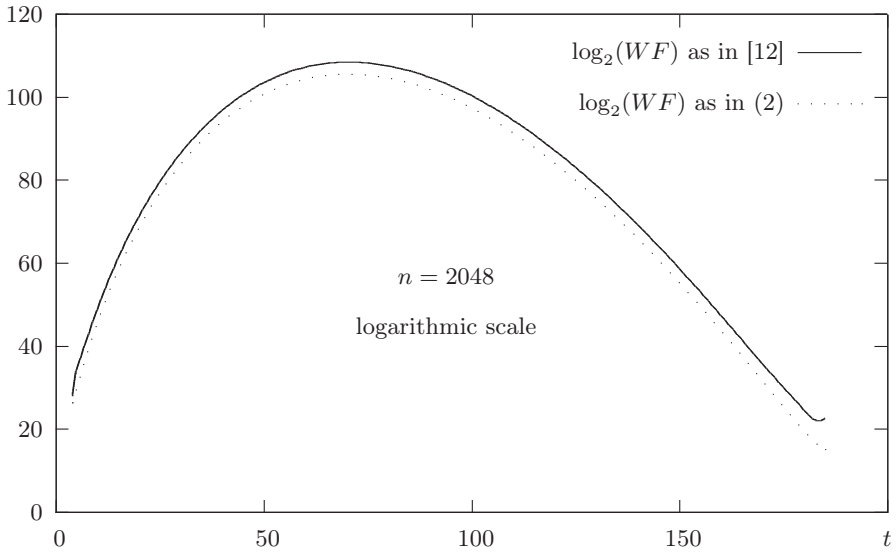

Fig. 5. Binary workfactor ($\log_2$) for finding words of weight t in a binary code of length 2048 and dimension $2048 - 11t$ (Goppa codes parameters)

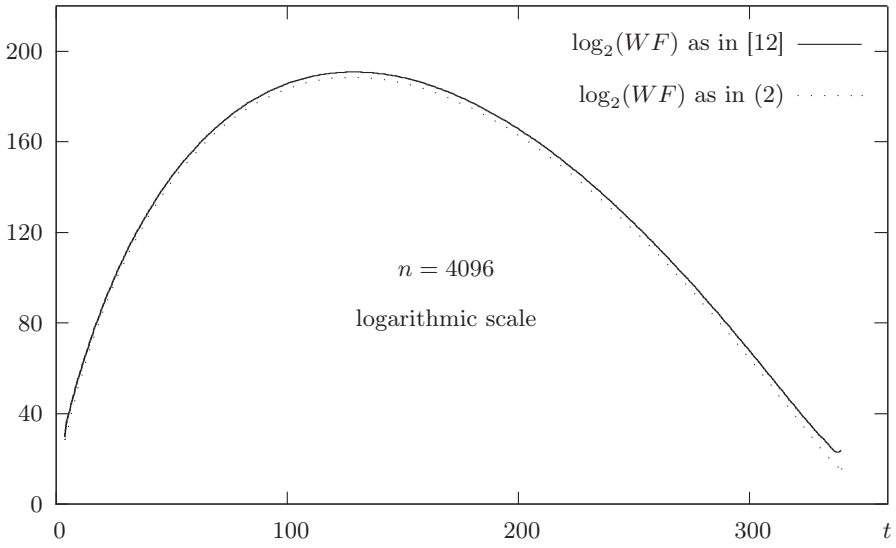

Fig. 6. Binary workfactor ($\log_2$) for finding words of weight t in a binary code of length 4096 and dimension $4096 - 12t$ (Goppa codes parameters)

$$(\exists_{\phi'_{n,t}(\mathbf{e})\mathsf{H}\in\mathcal{L}_i} : \mathbf{e}_\gamma \neq 0) \Rightarrow (\forall_{\phi'_{n,t}(\mathbf{e})\mathsf{H}\in\mathcal{L}_j} : \mathbf{e}_\gamma = 0).$$

If the lists are not of the desired cardinality $2^{\frac{r}{a+1}}$ ($r = n - k$ is the hash width), we can modify the attack accordingly, (compare [3]). We omit details and conclude that the size of the lists implies the following restriction to the attacker:

$$\frac{2^a}{a+1} \leq \frac{w}{r} \log_2 \left(\frac{n}{w}\right).$$

Therefore, the authors of [3] conclude that the work factor for an attacker grows exponentially with $n - k$ if we choose two constants $\alpha, \beta \in \mathbb{R}$ and then compute $(n, w) = (\alpha(n - k), \beta(n - k))$ as then a is upper bounded by a constant.

Likewise, Wagner's algorithm can be used to generate a valid signature in the CFS scheme (existential forgery): The attacker generates four lists: One with possible hash values, and the remaining three as syndromes of weight $t/3$ vectors. The dominating term for the cost of the attack is $2^{mt/3}$. For the $m = 16, t = 9$ CFS parameter set, this leads to an attack that can be performed in about 2^{59} operations.[6]

3.5 The impact of quantum computers

To our knowledge there is no connection between coding theory and the "Hidden Subgroup Problem" as in the case of number theoretic cryptosystems. However, there is still the possibility to employ Grover's algorithm to speed up searching the secret key or the space of possible plaintexts. In this chapter we give an intuition, why Grover's algorithm is not able give a significant speed-up for the existing attacks on code based cryptosystem.

In the following we make the simplifying assumption that by Grover's algorithm we are able to search a set of size N in $\mathcal{O}(\sqrt{N})$ operations on a quantum computer with at least $\log_2(N)$ QuBits. However, a consecutive call of Grover's algorithm is not possible, i.e. if the set to be searched is defined by the output of Grover's algorithm, we can not search this space with Grover's algorithm before writing it in complete to the (classic) memory, see Section 5 of Chapter 1 "Quantum computing".

Solving the generalized birthday problem

The iterative step of Wagner's algorithm can be realized by sorting algorithms, which can not be sped-up with quantum computers so far: Instead of searching $\mathcal{L}_i \times \mathcal{L}_{i+2^{a-1}}$ for all pairs $(x_i, x_{i+2^{a-1}})$ that $LSB_{\frac{r}{a+1}}(x_i + x_{i+2^{a-1}}) = 0$ we can sort the lists $\mathcal{L}_i$ after $LSB_{\frac{r}{a+1}}(x_i)$ and $\mathcal{L}_{i+2^{a-1}}$ after $LSB_{\frac{r}{a+1}}(x_{i+2^{a-1}}) = 0$. The merged list of pairs can now be directly read from the sorted lists (the

[6] Although we do not have a reference, we attribute this attack to Bleichenbacher

halves of the pairs are sorted into the same positions/boxes). If both lists have the same size $\sqrt{N}$, this means that the merging can be done in $\sqrt{N}$ operations instead of N, which is the same speed-up that can be achieved by Grover's algorithm. Thus, even with a quantum computer we can not expect to get attacks for FSB or CFS more efficient than the existing ones.

Algorithms for searching low weight codewords

The crucial point of algorithms for finding low weight codewords is to guess part of the structure at the beginning and then search for the vector in the remaining space. This can be seen as a "divide-and-conquer" strategy. However, this particular strategy of the attacks prevents an effective use of Grover's algorithm - or to be more precise - achieves the same speed-up as Grover's algorithm would achieve: The search step in the algorithms for finding low weight codewords is realized in the same way as in Wagner's algorithm for the generalized birthday paradox. Thus there is no possibility to significantly speed-up the search step by Grover's algorithm.

One might argue, that the guessing phase can be seen as a search phase, too. However, as mentioned before, this would either require an iterative application of Grover's algorithm (which is not possible) or a memory of size of the whole search space, as the search function in the second step depends on the first step. This would clearly ruin the "divide-and-conquer" strategy and is thus not possible either.

Table 2 gives an overview for the advantage of quantum computers over classical computers in attacking the McEliece PKC. One can see, that the

McEliece parameters m, t	Workload Cryptanalysis (in binary operations) classic	quantum computer	Minimal number of Qubits	Quantum-computer bit security[7]
$11, 32$	2^{91}	2^{86}	25	80
$11, 40$	2^{98}	2^{94}	50	88
$12, 22$	2^{93}	2^{87}	29	80
$12, 45$	2^{140}	2^{133}	28	128

[7] Compare remark 1

Table 2. Attacking the McEliece PKC

expected advantage does not lead to significantly different security estimations for the McEliece PKC.

4 Codes and structures

In this section we will consider structural attacks, i.e. attacks on the private key of code based PKCs. All codes with an efficient error correction algorithm have either an algebraic structure or are specially designed. For most codes, the knowledge of the canonical generator matrix allows efficient error correction. This is true for all codes one could consider for cryptographic applications (i.e. the ones of large dimension):

- Goppa/alternant codes [48]
- GRS codes [52]
- Gabidulin codes [27]
- Reed-Muller codes [70]
- Algebraic geometric codes [35]
- BCH codes [28]
- Graph based codes (LDPC-, expander-, LT- or turbo-codes)

While graph based codes almost immediately reveal their structure because of their sparse check matrix, this is not obvious for the algebraic codes. In this chapter we thus view how algebraic structures or permutations of a code can be recovered by an attacker.

4.1 Code equivalence

In code-based public key cryptography, one may try to hide a secret code $\mathcal{C}$ by applying an isometry f to it and publish a basis of the code $\mathcal{C}' = f(\mathcal{C})$. If the isometry f is known, a decoder for $\mathcal{C}'$ can be obtained. Hopefully, the isometry will scramble the code structure, making the decoding intractable. In the binary case (the most common) the isometry is "just" a permutation of the support.

An *isometry* of a metric space is a mapping which preserves the distance. Thus, codes that are images of one another by an isometry share all their metric properties and will be functionally equivalent. When the metric space is a vector space, we define the *semi-linear* isometries as those which preserve vector subspaces (i.e. the image of any vector subspace is another vector subspace). The semi-linear isometries of the Hamming space $\mathbb{F}_q^n$ are of the form

$$\begin{array}{rccc} \Psi_{V,\pi,\sigma}: & \mathbb{F}_q^n & \to & \mathbb{F}_q^n \\ & (x_i)_{i\in\mathcal{I}} & \mapsto & (v_i \pi(x_{\sigma^{-1}(i)}))_{i\in\mathcal{I}} \end{array} \tag{4}$$

where $V = (v_i)_{i\in\mathcal{I}}$ is a sequence of non zero elements of $\mathbb{F}_q$, π is a field automorphism of $\mathbb{F}_q$ and σ a permutation of the code support $\mathcal{I}$ (unless otherwise specified, we will now consider codes of length n and support $\mathcal{I}$).

Note that if $\mathcal{C}$ and $\mathcal{C}'$ are linear codes over $\mathbb{F}_q$ with $\mathcal{C}' = f(\mathcal{C})$ for some isometry f of the Hamming space $\mathbb{F}_q^n$, then there exists a semi-linear isometry g such that $\mathcal{C}' = g(\mathcal{C})$ (except in the degenerate case where $\mathcal{C}$ is decomposable,

that is the direct sum of two codes with disjoint support, see [51]). So as long as we only consider linear codes there is no loss of generality if we restrict ourselves to semi-linear isometries.

In the binary case ($q = 2$) semi-linear isometries are reduced to the support permutations (the v_i are all equal to 1 and the only field automorphism is the identity).

Definition 2. *Two linear codes $\mathcal{C}$ and $\mathcal{C}'$ are* equivalent *if one is the image of the other by a semi-linear isometry.*

Definition 3. *Two linear codes $\mathcal{C}$ and $\mathcal{C}'$ are* permutation-equivalent *if there exists a permutation σ such that*

$$\mathcal{C}' = \sigma(\mathcal{C}) = \{(x_{\sigma^{-1}(i)})_{i\in\mathcal{I}} \mid (x_i)_{i\in\mathcal{I}} \in \mathcal{C}\}.$$

The two definitions coincide in the binary case. Note also that the use of σ^{-1} in the index is consistent as we have $\pi(\sigma(\mathcal{C})) = \pi \circ \sigma(\mathcal{C})$.

Code equivalence relates with the ability of a code to correct errors. Two equivalent codes will have the same correcting capability.

Let $\mathcal{C}$ be a code equipped with a t-error correcting decoder $D_{\mathcal{C}}$. For any isometry f, the mapping $f \circ D_{\mathcal{C}} \circ f^{-1}$ is a t-error correcting decoder for $\mathcal{C}' = f(\mathcal{C})$.

4.2 The support splitting algorithm

The support splitting algorithm aims at solving the Code Equivalence problem:

Problem 6 (Code Equivalence).

Instance: Two matrices G_1 and G_2 defined over a finite field.
Question: Are the linear codes $\mathcal{C}_1$ and $\mathcal{C}_2$ spanned respectively by the rows of G_1 and G_2 permutation-equivalent?

This problem was introduced by Petrank and Roth [59], who proved that it was harder than the graph isomorphism problem but not $\mathcal{NP}$-complete unless $\mathcal{P} = \mathcal{NP}$.

Invariants and signatures

Let $\mathcal{L}_n$ denote the set of all linear codes of length n, and let $\mathcal{L} = \bigcup_{n>0} \mathcal{L}_n$ be the set of all linear codes.

Definition 4. *An* invariant *over a set E is defined to be a mapping $\mathcal{L} \to E$ such that any two permutation-equivalent codes take the same value.*

For instance the length, the cardinality or the minimum Hamming weight are invariants over the integers. The weight enumerator polynomial is an invariant over the polynomials with integer coefficients.

Applying an invariant, for instance the weight enumerator, may help us to decide whether two codes are equivalent or not. Two codes with different weight enumerators cannot be equivalent. Unfortunately we may have inequivalent codes with the same weight enumerator, though this only occurs with a small probability.

Any invariant is a global property of a code. We need to define a local property, that is a property of a code and of one of its positions.

Definition 5. *A* signature S *over a set* E *maps a code* $\mathcal{C}$ *of length* n *and an element* $i \in \mathcal{I}$ *into an element of* E *and is such that for all permutations* σ *on* $\mathcal{I}$, $S(\mathcal{C}, i) = S(\sigma(\mathcal{C}), \sigma(i))$.

A signature can be obtained, for instance, by applying an invariant on punctured codes. To an invariant V, we associate the signature $S_V : (\mathcal{C}, i) \mapsto V(\mathcal{C}_{\cdot\mathcal{I}\setminus\{i\}})$ ($\mathcal{C}_{\cdot\mathcal{J}}$ denotes the code restricted to $\mathcal{J} \subset \mathcal{I}$).

Now, if we have a signature S, and wish to answer the question: *"Are* $\mathcal{C}$ *and* $\mathcal{C}'$ *permutation-equivalent?"*, we can compute the sets $S(\mathcal{C}, \mathcal{I}) = \{S(\mathcal{C}, i), i \in \mathcal{I}\}$ and $S(\mathcal{C}', \mathcal{I}) = \{S(\mathcal{C}', i), i \in \mathcal{I}\}$. If $\mathcal{C}$ and $\mathcal{C}'$ are permutation-equivalent, then those sets must be equal (and for every signature value obtained more than once, the multiplicity must be the same). Moreover, for each distinct value in the sets $S(\mathcal{C}, \mathcal{I})$ and $S(\mathcal{C}', \mathcal{I})$, some information on the permutation between $\mathcal{C}$ and $\mathcal{C}'$ is revealed. The number of distinct values taken by a given signature for a given code $\mathcal{C}$ is thus of crucial importance to measure its efficiency.

Definition 6. *Let* $\mathcal{C}$ *be a code of length* n.

- *A signature* S *is said to be* discriminant *for* $\mathcal{C}$ *if there exist* i *and* j *in* $\mathcal{I}$ *such that* $S(\mathcal{C}, i) \neq S(\mathcal{C}, j)$.
- *A signature* S *is said to be* fully discriminant *for* $\mathcal{C}$ *if for all* i *and* j *distinct in* $\mathcal{I}$, $S(\mathcal{C}, i) \neq S(\mathcal{C}, j)$.

If $\mathcal{C}' = \sigma(\mathcal{C})$ and if S is fully discriminant for $\mathcal{C}$, then, for all i in $\mathcal{I}$, there exists a unique element j in $\mathcal{I}$ such that $S(\mathcal{C}, i) = S(\mathcal{C}', j)$, and we have $\sigma(i) = j$ and we thus obtain the permutation σ.

Description of the algorithm

If we assume the existence of a procedure `find_fd_signature` which returns for any generator matrix G a signature which is fully discriminant for $\mathcal{C} = \langle \mathsf{G} \rangle$, then Algorithm 4.1 will recover the permutation between permutation-equivalent codes. In fact it is easy to produce a procedure `find_fd_signature`, but the signature it returns has an exponential complexity.

Algorithm 4.1 The support splitting algorithm

Input: G_1 and G_2 two $k \times n$ matrices
Output: a permutation
$S \leftarrow$ `find_fd_signature`$(T[\,] = \emptyset)$
for $i \in \mathcal{I}$ **do**
  $T[S(\mathsf{G}_1, i)] \leftarrow i$
for $i \in \mathcal{I}$ **do**
  $\sigma[i] \leftarrow T[S(\mathsf{G}_2, i)]$

The difficulty is to obtain, for as many codes as possible, a fully discriminant signature which can be computed in polynomial time. The hull [2] of a linear code $\mathcal{C}$ is defined as its intersection with its dual $\mathcal{H}(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^\perp$. It has some very interesting features:

(i) It commutes with permutations: $\mathcal{H}(\sigma(\mathcal{C})) = \sigma(\mathcal{H}(\mathcal{C}))$
(ii) The hull of a random code is almost always of small dimension [69].
(iii) For all $i \in \mathcal{I}$, exactly one of the three sets $\mathcal{H}(\mathcal{C}_{.\mathcal{I}\setminus\{i\}})$, $\mathcal{H}(\mathcal{C}^\perp{}_{.\mathcal{I}\setminus\{i\}})$ and $\mathcal{H}(\mathcal{C})_{.\mathcal{I}\setminus\{i\}}$ is strictly greater than the other two, which are equal [68].

We consider the following signature

$$S(\mathcal{C}, i) = \left(W(\mathcal{H}(\mathcal{C}_{.\mathcal{I}\setminus\{i\}})), W(\mathcal{H}(\mathcal{C}^\perp{}_{.\mathcal{I}\setminus\{i\}}))\right)$$

where $W(\mathcal{C})$ denotes the weight enumerator polynomial of $\mathcal{C}$. Because of (i), the mapping $S()$ is a signature, because of (ii) it is almost always computable in polynomial time, and because of (iii), it is discriminant (but not always fully discriminant).

We apply $S()$ to all positions of a code and group those with the same value. We obtain a partition of the support. Using that partition we can refine the signature and eventually obtain a fully discriminant signature in a (conjectured) logarithmic number of refinements. When used on two codes of length n, the heuristic complexity for the whole procedure is

$$\mathcal{O}\left(n^3 + 2^h n^2 \log n\right)$$

where h is the dimension of the hull.

The first term is the cost of the Gaussian elimination needed to compute the hull. The second term is the (heuristic) number of refinements, $\log n$, multiplied by the cost of one refinement (n weight enumerator of codes of dimension h and length n). In practice, for random codes, the hull has a small dimension with overwhelming probability [69] and the dominant cost for the average case is $\mathcal{O}(n^3)$. The worst case happen when the hull's dimension is maximal: weakly-self dual codes ($\mathcal{C} \subset \mathcal{C}^\perp$) are equal to their hulls. The algorithm becomes intractable with a complexity equal to $\mathcal{O}(2^k n^2 \log n)$. This is the case in particular of the Reed-Muller codes used in Sidelnikov's system [70]. For more details on the support splitting algorithm, see [68].

4.3 Recognizing code structures

Only for alternant and algebraic geometric codes it is sufficient to publish a systematic generator matrix of a code permutation equivalent to the secret one in order to hide the private key from an attacker. In this section we want to give the reader an intuition, in which cases and how the structure of an algebraic code can be recognized or not.

GRS Codes

In 1992 V.M. Sidelnikov and S.O. Shestakov proposed an attack on the GRS Niederreiter PKC (compare §2.1) which reveals an alternative private key in polynomial time [71]. We consider this attack to be worth mentionable, as Goppa codes are subfield subcodes of GRS codes. Even though, the results from [71] do not affect the security of the original McEliece PKC.

In their attack, Sidelnikov and Shestakov take advantage of the fact, that the check matrix of GRS code is of the form (see §6.2)

$$\bar{\mathsf{H}} = \begin{pmatrix} z_1 a_1^0 & z_1 a_1^1 & \cdots & z_1 a_1^s \\ z_2 a_2^0 & z_2 a_2^1 & \cdots & z_2 a_2^s \\ \vdots & & \ddots & \vdots \\ z_n a_n^0 & z_n a_n^1 & \cdots & z_n a_n^s \end{pmatrix}^\top \in \mathbb{F}_q^{n\times(s+1)} \,. \tag{5}$$

A public key is of the form $\mathsf{H}' = \mathsf{M}\bar{\mathsf{H}}\mathsf{P}$, where M is a non-singular matrix and P a permutation matrix. The permutation matrix P does not change the structure of $\bar{\mathsf{H}}$, so we don't have to worry about P. Sidelnikov and Shestakov use the fact, that each entry of the row $\mathsf{H}'_{i\cdot}$ can be expressed by a polynomial f of degree $\leq s$ in a_i. From this observation one can derive a system of polynomial equations whose solution yields the private key.

To perform the attack, it is necessary to see, that we can assume that a_1, a_2, a_3 are distinguished elements, so we extend $\mathbb{F}_q$ by ∞: $\mathbb{F} := \mathbb{F}_q \cup \infty$ with $1/\infty = 0$ with $1/0 = \infty$ and $f(\infty) = f_s$ for every polynomial $f(x) = \sum_{j=0}^{s} f_j x^j$ of degree $\leq s$ over $\mathbb{F}_q$. Sidelnikov and Shestakov show that for every birational transformation, i.e. $\mathbb{F}$-automorphism

$$\phi(x) = \begin{cases} \frac{a}{c} & c \neq 0, x = \infty \\ \frac{ax+b}{cx+d} & \text{otherwise} \end{cases} \quad \text{with } a,b,c,d \in \mathbb{F}_q,\ ad - bc \neq 0$$

there exist $z'_1, \cdots, z'_1$ and a matrix M' such that

$$\mathsf{H}' = \mathsf{M}\left(\mathsf{M}'\right)^{-1} \cdot \begin{pmatrix} z'_1\phi(a_1)^0 & z'_1\phi(a_1)^1 & \cdots & z'_1\phi(a_1)^s \\ z'_2\phi(a_2)^0 & z'_2\phi(a_2)^1 & \cdots & z'_2\phi(a_2)^s \\ \vdots & & \ddots & \vdots \\ z'_n\phi(a_n)^0 & z'_n\phi(a_n)^1 & \cdots & z'_n\phi(a_n)^s \end{pmatrix}^\top .$$

Thus, without loss of generality, we can assume that H' defines the (dual) code with codewords

$$(z'_1 f(1), z'_2 f(0), z'_3 f(\infty), z'_4 f(a_4), z'_5 f(a_5), \cdots, z'_n f(a_n)),$$

where f varies over the polynomials of degree $\leq s$ over $\mathbb{F}_{2^m}$. This means, H' defines an extended GRS code (see Definition 9) with $a_1 = 1, a_2 = 0$ and $a_3 = \infty$. Note that because $a_3 = \infty$ we have $a_i \neq \infty$ for all $i \neq 3$.

The general idea of the attack is the following: If we take two codewords with $s-1$ common zeroes, then the corresponding polynomials π_1, π_2 have $s-1$ common factors, while each polynomial is of degree $\leq s$. As we have noted above, we can assume that $\pi_1(0) = 1 = \pi_2(1)$ and $\pi_1(1) = 0 = \pi_2(0)$, which leads to

$$\frac{\pi_1(x_j)}{\pi_2(x_j)} = \frac{\pi_1(\infty)}{\pi_2(\infty)} \cdot \frac{x_j - 1}{x_j} = \frac{\pi_1(a_3)}{\pi_2(a_3)} \cdot \frac{x_j - 1}{x_j},$$

and thus reveals a_j on all positions where neither π_1 nor π_2 are zero. We can repeat this procedure with other pairs of polynomials to obtain the whole vector $(a_1, a_2, \cdots, a_n)$. Taking a birational transform ϕ, such that $(\phi(a_1), \phi(a_2), \cdots, \phi(a_n))$ does not contain the ∞ element, we can recover $(z_1, \cdots, z_n)$ by setting $z_1 = 1$ and employing Gauss's algorithm afterwards. As pairs of codewords with $s-1$ common zeroes can be found by computing a systematic check matrix, the algorithm has a running time of $\mathcal{O}\left(s^4 + sn\right)$.

Remark 2. There were two proposals to modify the GRS Niederreiter cryptosystem: The first one is by E. Gabidulin and consists in adding artificial errors to the generator matrix [24] whereas the second by P. Loidreau uses a subcode of a GRS code (compare Table 1). While the first proposal did not receive much attention so far, the second one was cryptanalyzed by C. Wieschebrink [78], who showed how to attack that modification for small parameter sets by finding pairs of code words with $s-1-i$ common zeroes and guessing i elements from $(x_4, \cdots, x_n)$. This attack can be applied to the Niederreiter PKC variant proposed in [24] in certain cases, e.g., by puncturing the public code. Even if these attacks have exponential runtime, we are not sure if secure parameter sets have a better performance than McEliece's PKC with Goppa codes.

Remark 3. The attack on the GRS Niederreiter PKC can not be applied to McEliece/Niederreiter cryptosystems using Goppa codes. Even though for every Goppa code there is a check matrix H which has the same structure as the check matrix $\bar{\mathsf{H}}$ for GRS codes in equation (5) (see [47]), there is no analogous interpretation of H' for the Niederreiter cryptosystem using Goppa codes. We are able to view H as a matrix over $\mathbb{F}_2$ if we are using Goppa codes, whereas this doesn't work for GRS codes. Thus we have different matrices M: $\mathsf{M} \in \mathbb{F}_{2^m}^{(s+1)\times(s+1)}$ for the GRS case and $\mathsf{M} \in \mathbb{F}_2^{m(s+1)\times m(s+1)}$ for Goppa codes. Thus, in the latter case, H' has no obvious structure, as long as M is unknown.

Rank metric codes

So called Gabidulin codes are a subclass of Srivastava codes, which are MDS codes (i.e., they have a check matrix in form of Equation (5) and their minimum distance is $d = n-k+1$) [47], for which an efficient decoding algorithm exists [27]. These codes were introduced into cryptography together with the notion of rank metric (see Definition 11). The class of Gabidulin codes is the only class of codes for which an algorithm is known, which can correct errors in Hamming and rank metric. For now, however, we omit the interesting notion of rank metric, but give a general intuition, why one can recognize the structure of a Gabidulin code even better then the one of a GRS code and why the modifications proposed in Table 1 do not serve to hide their structure sufficiently for cryptographic purposes.

We will define Gabidulin codes by their generator matrix. For ease of notation we introduce the operator λ_f, which maps a matrix $\mathsf{M} = (m_{ij})$ to a blockmatrix:

$$\begin{aligned} \lambda_f : \mathbb{F}_{q^m}^{m\times n} &\to \mathbb{F}_{q^m}^{m(f+1)\times n} \\ \mathsf{M} &\mapsto \begin{bmatrix} \mathsf{M} \\ \mathsf{M}^{[q]} \\ \vdots \\ \mathsf{M}^{[q^f]} \end{bmatrix}, \end{aligned} \tag{6}$$

where $\mathsf{M}^{[x]} := (m_{ij}^x)$.

Definition 7. *Let $\mathbf{g} \in \mathbb{F}_{q^m}^n$ be a vector s.t. the components g_i, $i = 1, \cdots, n$ are linearly independent over $\mathbb{F}_q$. This implies that $n \leq m$. The $[n,k]$ Gabidulin code $\mathcal{G}$ is the rank distance code with generator matrix*

$$\mathsf{G} = \lambda_{k-1}(\mathbf{g}). \tag{7}$$

The vector $\mathbf{g}$ is said to be a *generator vector* of the Gabidulin code $\mathcal{G}$ (It is not unique, as all vectors $a\mathbf{g}$ with $0 \neq a \in \mathbb{F}_{q^m}$ are generator vectors of $\mathcal{G}$). Further, if $\mathsf{T} \in \mathbb{F}_q^{n\times n}$ is an invertible matrix, then $\mathsf{G}\cdot\mathsf{T}$ is the generator matrix of the Gabidulin code with generator vector $\mathbf{g}\mathsf{T}$. An error correction algorithm based on the "right Euclidian division algorithm" runs in $\mathcal{O}\left(d^3 + dn\right)$ operations over $\mathbb{F}_{q^m}$ for $[n,k,d]$ Gabidulin codes [27]. The property, that a matrix G generates a Gabidulin code is invariant under the operator $\Lambda_f(\mathsf{M})$:

Lemma 1. *If G is a generator matrix of an $[n,k]$ Gabidulin code $\mathcal{G}$ with $k < n$, then $\Lambda_f(\mathsf{G}^{\mathrm{pub}})$ is a generator matrix of the Gabidulin code with the same generator vector as $\mathcal{G}$ and dimension* $\min\{n, k+f\}$.

Another nice property of Gabidulin codes is, that the dual code of an $[n,k]$ Gabidulin code is an $[n,n-k]$ Gabidulin code (see [27]):

Lemma 2. *Let $\mathcal{G}$ be an $[n,k]$ Gabidulin code over $\mathbb{F}_{q^m}$ with generator vector g. Then $\mathcal{G}$ has a check matrix of the form*

$$\mathsf{H} = \lambda_{n-k-1}\left(\mathbf{h}^{[1/q^{n-k-1}]}\right) \in \mathbb{F}_{q^m}^{n-k\times n} .$$

Further, the vector $\mathbf{h}$ *is uniquely determined by* $\mathbf{g}$ *(independent from k) up to a scalar factor $\gamma \in \mathbb{F}_{q^m} \setminus \{0\}$. We will call* $\mathbf{h}$ *a check vector of $\mathcal{G}$.*

The major disadvantage of Gabidulin codes, is the fact, that one can easily distinguish a random $k \times n$ matrix M from an arbitrary generator matrix G of an $[n,k]$ Gabidulin code by a quite simple operation: The matrix $\lambda_1(\mathsf{G})$ defines an $[n,k+1]$ code, while the matrix $\lambda_1(\mathsf{M})$ will have rank $> k+1$ with overwhelming probability [45].

Remark 4. Unlike for GRS codes, it is not sufficient to take the generator matrix $\mathsf{G}_{\mathrm{SUB}}$ of an $[n,k-l]$ subcode of an secret $[n,k]$ Gabidulin code $\mathcal{G} = \langle \mathsf{G} \rangle$ to hide the structure as it was proposed in [5]. It is easy to verify, that $\lambda_1(\mathsf{G}_{\mathrm{SUB}})$ defines a subcode of $\langle \lambda_1(\mathsf{G}) \rangle$ and thus any full rank vector in the dual of $\lambda_{n-k-2}(\mathsf{G}_{\mathrm{SUB}})$ gives a Gabidulin check vector which allows to decode in $\mathsf{G}_{\mathrm{SUB}}$.

There were plenty of other proposals on how to use Gabidulin codes for cryptography, most with the notation of rank metric, see §6.3. However, as mentioned before, all these variants proved to be insecure [53, 57].

Reed-Muller Codes

Reed-Muller codes were considered for cryptographic use by Sidelnikov [70]. His basic proposal is to replace the Goppa code in McEliece's scheme by a Reed-Muller code, which can be defined as follows:

The Reed-Muller code in m variables of degree r consists of all codewords which can be obtained by evaluating some polynomial in $\mathbb{F}_2[x_1, \cdots, x_m]$ of degree at most r at all possible variable assignments, see [47]. Lexicographic ordering of the 2^m possible assignments leads to the following recursive description of the canonical generator matrix $\mathsf{R}(r,m)$, which is reducible:

$$\mathsf{R}(r,m) = \left[\begin{array}{c|c} \mathsf{R}(r,m-1) & \mathsf{R}(r,m-1) \\ \hline 0 & \mathsf{R}(r-1,m-1) \end{array}\right], \tag{8}$$

where $\mathsf{R}(r,m) = \mathsf{R}(m,m)$ for $r > m$ and $\mathsf{R}(0,m)$ is the codeword of length 2^m which is one at all positions. The code $\langle \mathsf{R}(r,m) \rangle$ is a $[2^m, \sum_{i=0}^{r} \binom{m}{i}, d]$ code with $d = 2^{m-r}$ and is a subcode of $\langle \mathsf{R}(r+1,m) \rangle$ [47].

From the construction of a Reed-Muller code it is easy to see, that each low weight codeword in $\mathsf{R}(r,m)$ can be represented as a product of $m-r$ pairwise different linear factors. Due to this large number of low weight codewords

(there exist about $2^{mr-r(r-1)}$ of them), Stern's algorithm [72] and its variants allow to find low weight codewords in Reed-Muller codes efficiently, compare §3.3.

Now, let $\mathsf{P} \in \mathbb{F}_2^{n\times n}$ be a permutation matrix. The main observation, which allows to recover P from some generator matrix $\mathsf{G}^{\mathrm{pub}}$ of $\langle \mathsf{R}\,(r,m)\,\mathsf{P}\rangle$ is that each low weight codeword of $\left\langle \mathsf{G}^{\mathrm{pub}}\right\rangle$ can be "factored". Indeed, each low weight codeword $\mathbf{v}$ in $\left\langle \mathsf{G}^{\mathrm{pub}}\right\rangle$ can be written as the "product" of a low weight codeword $\bar{\mathbf{v}}$ in $\mathsf{R}(r-1,m)\mathsf{P}$ and a low weight codeword $\hat{\mathbf{v}}$ in $\mathsf{R}(1,m)\mathsf{P}$:

$$\mathbf{v} := \bar{\mathbf{v}} \odot \hat{\mathbf{v}} := (\bar{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1, \bar{\mathbf{v}}_2 \cdot \hat{\mathbf{v}}_2, \cdots, \bar{\mathbf{v}}_n \cdot \hat{\mathbf{v}}_n)$$

The goal is to find the factor $\bar{\mathbf{v}}$ of $\mathbf{v}$. If a sufficiently large number of low weight codewords of $\left\langle \mathsf{G}^{\mathrm{pub}}\right\rangle$ have been factored, the code $\mathsf{R}(r-1,m)\mathsf{P}$ can be reconstructed. Iteratively reducing the problem it remains to solve the problem to recover P from $\mathsf{R}(1,m)\mathsf{P}$, which is trivial [50].

Remark 5. The application of N. Sendrier's "Support Splitting Algorithm" (SSA, see §4.2) for finding the permutation between permutation equivalent codes is not efficient for Reed-Muller codes. The runtime of SSA is exponential in the dimension of the hull of a code $\mathcal{C}$, i.e. the dimension of $\mathcal{C} \cup \mathcal{C}^{\perp}$, which is large, if $\mathcal{C}$ is a Reed-Muller code. Thus, Sidelnikov's proposal can not be attacked via the SSA.

In [50] L. Minder gives an algorithm to deduce the factor $\bar{\mathbf{v}}$ of $\mathbf{v}$ efficiently. We will assume that $\mathsf{P} = \mathsf{Id}$ in the following, since the algorithm does not depend on P. Assume, that $\mathbf{v}$ is a low weight codeword in $\langle \mathsf{R}(r,m)\rangle$, then we may well assume, that the corresponding polynomial can be written as $v = v_1 \cdot v_2 \cdots v_r$ (after a change of basis). Now, the code $\mathcal{C}$ consisting of all codewords with support disjoint from $\mathbf{v}$ can be represented as a polynomial

$$f = f(v_1, v_2, \cdots, v_m) = \sum_{I\subseteq\{1,2,\cdots,r\}} f_I \cdot \prod_{i\in I} v_i$$

with $f_I \in \mathbb{F}_2[v_{r+1}, v_{r+2}, \cdots, v_m]$. Further, since f and v have disjoint support, we have $f(\underbrace{1,1,\cdots,1}_{r \text{ times}}, v_{r+1}, v_{r+2}, \cdots, v_m) = 0$ and thus

$$\sum_{I\subseteq\{1,2,\cdots,r\}} f_I = 0.$$

We can see, that restricting the codewords of disjoint support to the ones with a fixed value for $(v_1, \cdots, v_r) \neq (1,1,\cdots,1)$ we obtain an permuted version of $\mathsf{R}(r-1, m-r-1)$ (after puncturing). This shows, that the codewords with disjoint support from $\mathbf{v}$ form a code which is a permuted concatenated code build of $2^r - 1$ blocks, each a Reed-Muller code of degree $r-1$ in $m-r-1$ variables, i.e. there is a permutation Π such that

$$\mathcal{C}\Pi \subseteq \underbrace{(0,0,\cdots,0)}_{2^{m-r} \text{ times}} \otimes \left(\bigotimes_{i=1}^{2^r-1} \langle \mathsf{R}(r-1, m-r-1) \rangle \right).$$

Thus, each of this inner blocks together with the support of $\mathbf{v}$ gives a low weight codeword in $\langle \mathsf{R}(r-1,m)\mathsf{P} \rangle$.

Even if the identification of the inner blocks of a concatenated code has been studied in [64], Minder proposes to identify the different blocks by statistical analysis: For a low weight codeword $\mathbf{y}$, he states, that the probability, that $\mathbf{y}_i = 1$ and $\mathbf{y}_j = 1$ is independent if and only if i and j do not belong to the same inner block.

Remark 6. The code $\langle \mathsf{R}(r,m)\mathsf{P} \rangle$ is a permutation of a concatenated code $\subseteq \bigotimes_{i=1}^{2^r} \langle \mathsf{R}(r-1, m-r-1) \rangle$, too. Thus, one might think of applying the statistical analysis directly to $\mathsf{R}(r,m)\mathsf{P}$ in order to partition the code. However, Minder states that the support of the low weight code words of $\mathsf{R}(r,m)$ is too large (i.e. twice the length of each block) to allow sampling from the desired space.

Minder's runtime analysis shows, that the crucial point is to find the low weight codewords in $\mathcal{C}$, which however, due to the large number of low weight codewords is practical for reasonable parameter sets, turning Sidelnikov's cryptosystem inefficient. For $r = 3$ and $m = 11$ for example, his algorithm allows to recover the permutation P in less than one hour on a desktop PC.

Structural attacks on the McEliece cryptosystem

Binary Goppa codes were proposed by McEliece in the original version of his system. So far, all known structural attacks on Goppa codes have an exponential cost.

We assume t-error correcting binary irreducible Goppa codes of length $n = 2^m$ over $\mathbb{F}_{2^m}$ are used for the key generation. The secret key is the code $\Gamma(\mathbf{L}, g)$ which consists of

- a *generator*, a monic irreducible polynomial $g(z)$ of degree t over $\mathbb{F}_{2^m}$
- a *support*, a vector $\mathbf{L} \in \mathbb{F}_{2^m}^n$ with distinct coordinates (in fact, with $n = 2^m$, this defines a permutation).

If either the support or the generator is known, the other part of secret can be recovered in polynomial time from the public key G^{pub}.

1. If the support $\mathbf{L}$ is known, then a multiple of $g(z)$ can be obtained from any codeword by using equation (12) page 139. Codewords can easily be obtained from G^{pub}, and after a few gcds (usually one is enough) the generator polynomial is obtained.
2. If the generator polynomial $g(z)$ is known, we construct a generator matrix G of the Goppa code of generator $g(z)$ and support $\mathbf{L}_0$ (where $\mathbf{L}_0$ is fixed

and chosen arbitrarily), and we obtain the secret vector $\mathbf{L}$ by applying the support splitting algorithm to G and G^{pub} (the permutation between G and G^{pub} will also be the permutation between $\mathbf{L}_0$ and $\mathbf{L}$).

In both cases, we obtain an exhaustive search attack, either by enumerating the permutations (proposed by Gibson in [31]) or by enumerating the irreducible polynomials [46]. There are $\approx 2^{tm}/t = n^t/t$ irreducible polynomials compared to $n! = \mathcal{O}(\sqrt{n}(n/\mathrm{e})^n)$ permutations. The second attack is always more efficient. To evaluate the cost of this attack we consider

- the number of monic irreducible polynomials of degree t over $\mathbb{F}_{2^m}$ [43, p. 93], equal to $\approx 2^{tm}/t = n^t/t$.
- the cost of the support splitting algorithm, equal to $\mathcal{O}(n^3)$, because Goppa codes behave like random codes and have a small hull.
- the number of distinct pairs support/generator that produce the same Goppa code, which is almost always equal to $m2^m = n \log_2 n$ [31].

We multiply the first two numbers and divide by the third and we get $\mathcal{O}(n^{t+2}/t \log n)$. In fact, it is possible to do slightly better by considering extended codes (an overall parity check bit is appended). The number of distinct pairs support/generator that produce the same extended Goppa code is almost always equal to $m2^m(2^{2m} - 1)$ (see [47, p. 347]). The support splitting algorithm can be applied on extended code and the complexity of the attack is reduced to

$$\mathcal{O}\left(\frac{n^t}{t \log n}\right) = \mathcal{O}\left(\frac{2^{tm}}{tm}\right).$$

This is currently the best known structural attack on McEliece encryption scheme using Goppa codes. As the best decoding attack is upper bounded by $\mathcal{O}(2^{(n-k)/2}) = \mathcal{O}(2^{tm/2})$ (see [4] for instance), structural attacks are never better than decoding attacks.

Choosing the secret codes: general pitfalls

Beyond the existence of an efficient structural attack today, what kind of assumptions do we want to (or have to) make for arguing of McEliece's scheme security? First, obviously, the family of codes used to produce the keys is critical. Binary Goppa codes are safe (or seem to be), but not Reed-Solomon codes [71], concatenated codes [64], elliptic codes [49], Reed-Muller codes [50] (to some extend), and many other unpublished attempts.

Indistinguishability is the strongest security assumption related with structural attacks. Informally, it says that it is not computationally feasible to tell apart a generator matrix of a random code from a generator matrix of a particular family. When it holds, the security of the corresponding public-key system can be reduced to the hardness of decoding, for which very strong arguments exist.

Indistinguishability is conjectured for binary Goppa codes, and in practice, no property is known that can be computed from a generator matrix in polynomial time and which behaves differently for binary Goppa codes and for binary linear codes. To our knowledge, this is the only such family of codes with an efficient decoding algorithm.

Using other families of codes in public key cryptography should be considered with great care. There are at least two possible pitfalls

- Families with high performance decoding, like concatenated codes, turbo-codes or LDPC codes, have many low weight codewords in their duals. Those low weight codewords may be easy to find and are likely to leak some of the code structure.
- As we have seen previously in this section (§4.3 and §4.3), families with optimal or sub-optimal combinatorial properties are dangerous too. For instance, (generalized) Reed-Solomon codes are MDS (the highest possible minimum distance), elliptic codes are almost MDS (minimum distance is just one less), in both case minimum weight codewords are not hard to find and reveal a lot of information on the code structure. Reed-Muller codes are highly structured, and though they have an optimal resistance to the support splitting algorithm (they are weakly self-dual), Lorenz Minder has exhibited a structural attack which is more efficient than the decoding attack.

Finally, let us mention algebraic geometry codes, proposed for cryptography by Janwa and Moreno [35]. They are probably insecure for small genus (Minder's work) but otherwise, their security status is unknown.

5 Practical aspects

The practice of McEliece's PKC or more generally of a code-based PKC raises many questions. We address here a few of them in this section. The main advantage of McEliece's scheme is a low algorithmic complexity for encryption and decryption and its main drawback is a large public key size. We will stress the first point and examine what can be done for the second. Also, for practical purposes, the system suffers from many weaknesses, most of them related to malleability. We will examine the generic and ad-hoc semantically secure conversions that solve those issues.

5.1 Fast en- and decryption for the McEliece PKC

We describe here the implementation of the McEliece encryption scheme. The error correcting code will be a binary irreducible t error-correcting Goppa code $\mathcal{G}$ of length $n = 2^m$ and dimension $k = n - tm$. We denote $D_{\mathcal{G}} : \{0,1\}^n \to \mathcal{G}$ a t-error correcting procedure for $\mathcal{G}$ (see §6.1). The private key is the decoder $D_{\mathcal{G}}$ and the public key is a generator matrix G of $\mathcal{G}$.

We assume the existence of an injective mapping $\phi_{n,t} : \{0,1\}^\ell \to \mathcal{W}_{n,t}$ easy to compute and to invert (see §5.1). The key features of the implementation we describe are presented in Algorithm 5.1. The two main differences from the original proposal are:

1. The public key is chosen in systematic form $\mathsf{G}^{\text{syst}} = (\mathsf{Id} \mid \mathsf{R})$.
2. The mapping $\phi_{n,t}$ will be used to encrypt ℓ additional information bits.

Those modifications do not alter the security of the system as long as a semantically secure conversion is used (such a conversion is needed anyway). Moreover, those conversions (see §5.3) require the use of $\phi_{n,t}$, so, for practical purpose, that part of the computation has to be done anyway.

Algorithm 5.1 Modified McEliece encryption scheme

- **Public key:** a $k \times (n-k)$ binary matrix R
- **Private key:** a decoder $D_{\mathcal{G}}$ for the code $\mathcal{G}$ spanned by $(\mathsf{Id} \mid \mathsf{R})$
- **Encryption:** the plaintext is $(\mathbf{m}_1, \mathbf{m}_2) \in \{0,1\}^k \times \{0,1\}^\ell$
  the ciphertext is $\mathbf{y} = (\mathbf{m}_1, \mathbf{m}_1\mathsf{R}) + \phi_{n,t}(\mathbf{m}_2) \in \{0,1\}^n$
- **Decryption:** the ciphertext is $\mathbf{y} \in \{0,1\}^n$
  compute the codeword $\mathbf{x} = D_{\mathcal{G}}(\mathbf{y})$, with $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_1\mathsf{R})$
  the plaintext is $(\mathbf{m}_1, \mathbf{m}_2) = (\mathbf{x}_1, \phi_{n,t}^{-1}(\mathbf{y} - \mathbf{x}))$

The algorithmic complexity of the encryption and decryption procedures are relatively easy to analyse.

- The encryption complexity is dominated by the vector/matrix multiplication (k times $k \times (n-k)$) and the call to $\phi_{n,t}$. In practice those two costs are comparable.
- The decryption complexity is dominated by the decoding $D_{\mathcal{G}}(\mathbf{y})$ and the call to $\phi_{n,t}^{-1}$. In practice the decoding is much more expensive.

McEliece with a systematic public key

Let G be the public key of an instance of McEliece cryptosystem with parameters (n, k, t). Let $\mathsf{G}^{\text{syst}} = (\mathsf{Id} \mid \mathsf{R}) = \mathsf{UG}$ be a systematic generator matrix of the same code (w.l.o.g. the first k column of G are non-singular and U is a $k \times k$ matrix which can be computed from G in polynomial time).

For any G, we denote $\Psi_{\mathsf{G}}(\mathbf{m}, \mathbf{e}) = \mathbf{m}\mathsf{G} + \mathbf{e}$. Using $\Psi_{\mathsf{G}^{\text{syst}}}$ instead of Ψ_{G} for the encryption has many advantages:

- the public key is smaller, as it has a size of $k(n-k)$ bits instead of kn,
- the encryption is faster, as we multiply the plaintext by a smaller matrix,
- the decryption is faster, as the plaintext is a prefix of the ciphertext cleared of the errors.

The drawback is a "decrease" of the semantic security. The following example is taken from [65, p. 34], and is the beginning of a ciphertext for an instance of McEliece using a systematic public key:

```
Le{ cryptosystèmas0basés suv les code{‘corveãteurs soît-ils sýòs?
```

Obviously, there is a leak of information. However, since we have $\Psi_{\mathsf{G}}(\mathbf{m}, \mathbf{e}) = \Psi_{\mathsf{G}^{\text{syst}}}(\mathbf{m}\mathsf{U}^{-1}, \mathbf{e})$, any inversion oracle for $\Psi_{\mathsf{G}^{\text{syst}}}$ can be transformed in an inversion oracle for Ψ_{G}. Thus, if the plaintext $\mathbf{m}$ is uniformly distributed, both versions are equally secure. In practice, this means that a semantically secure conversion (see §5.3) will enable us to use G^{syst} without loss of security.

Encoding constant weight words

The problem here is to exhibit, for given n and t, an efficient injective mapping into the set of binary words of length n and weight t, $\phi_{n,t} : \{0,1\}^{\ell} \to \mathcal{W}_{n,t}$. This mapping is needed for implementing Niederreiter scheme and is also used in most semantically secure conversions. In practice we want ℓ to be close to $\lfloor \log_2 \binom{n}{t} \rfloor$. Else, we risk a loss of security.

Enumerative method.

This method is optimal in terms of information rate and can be traced back to [15, 62]. It is based on the following bijective mapping

$$\begin{aligned} \theta : \quad \mathcal{W}_{n,t} &\longrightarrow \left[0, \tbinom{n}{t}\right[\\ (i_1, \ldots, i_t) &\longmapsto \tbinom{i_1}{1} + \tbinom{i_2}{2} + \cdots + \tbinom{i_t}{t} \end{aligned}$$

where the element of $\mathcal{W}_{n,t}$ is represented by its non-zero positions in increasing order $0 \le i_1 < i_2 < \ldots < i_t < n$. Computing θ requires the computation of t binomial coefficients. When t is not too large, computing the inverse θ^{-1} is not significantly more expensive thanks to the following inversion formula

$$x = \binom{i}{t} \Leftrightarrow i = X + \frac{t-1}{2} + \frac{t^2-1}{24}\frac{1}{X} + \mathcal{O}\left(\frac{1}{X^3}\right), X = (t!x)^{1/t}. \quad (9)$$

We can define $\phi_{n,t}$ as the restriction of θ^{-1} to the interval $\left[0, 2^{\ell}\right[$ where $\ell = \lfloor \log_2 \binom{n}{t} \rfloor$. Both $\phi_{n,t}$ and $\phi_{n,t}^{-1}$ can be obtained by computing t binomial coefficients and have a cost of $\mathcal{O}(t\ell^2) = \mathcal{O}(t^3 m^2)$ binary operations.

The decoding procedure is described in Algorithm 5.2. It uses formula (9) for inverting the binomial coefficients. In fact, this inversion does not require a great precision as the result we seek is an integer, not a floating point number. In practice `invert_binomial` has a negligible cost compared with the computation of the binomial coefficients.

Algorithm 5.2 Enumerative decoding

Input: $x \in \left[0, \binom{n}{t}\right[$
Output: t integers $0 \leq i_1 < i_2 < \ldots < i_t < n$
$j \leftarrow t$
while $j > 0$ **do**
  $i_j \leftarrow \texttt{invert\_binomial}(x, j)$
  $x \leftarrow x - \binom{i_j}{j}$
  $j \leftarrow j - 1$

where $\texttt{invert\_binomial}(x, t)$ returns the integer i such that $\binom{i}{t} \leq x < \binom{i+1}{t}$

Recursive source coding methods.

Those methods consist, as for the enumerative method, in finding a binary encoder for the source $\mathcal{W}_{n,t}$ equipped with the uniform probability (i.e. a compression algorithm). The idea is to consider a simpler approximative source model which allows a faster encoding and decoding. Linear time methods were proposed in [63,67]. It consists in a (variable length) encoder $\mathcal{W}_{n,t} \to \{0,1\}^*$, with the additional requirement that any (long enough) binary sequence can be decoded into a sequence of words of $\mathcal{W}_{n,t}$. For instance in [67], an element of $\mathcal{W}_{n,t}$ is first represented by a t-tuple $(\delta_1, \ldots, \delta_t)$ of integers where δ_i is the number of '0's between the $(i-1)$-th and the i-th '1' (the 0-th '1' is the beginning of the word). The encoding is recursively defined as:

$$\Psi_{n,t}(\delta_1, \delta_2, \ldots, \delta_t) = (f_{n,t}(\delta_1), \Psi_{n-\delta_1-1,t-1}(\delta_2, \ldots, \delta_t))$$

where $f_{n,t}$ is a source encoder for the set of integers $\{0, 1, \ldots, n-t\}$ equipped with the probability distribution

$$P_{n,t}(i) = \frac{\binom{n-i-1}{t-1}}{\binom{n}{t}}, i = 0, \ldots, n-t.$$

The model is then simplified. We choose d an integer such that

$$\sum_{i<d} P_{n,t}(i) = 1 - \frac{\binom{n-d}{t}}{\binom{n}{t}} \approx \frac{1}{2} \;\Leftrightarrow\; d \approx \frac{2^{1/t} - 1}{2^{1/t}} \left(n - \frac{t-1}{2}\right)$$

and we define $f_{n,t}$ as

$$f_{n,t}(i) = \begin{cases} 0, B_2(i) & \text{if } 0 \leq i < d \\ 1, f_{n-d,t}(i-d) & \text{if } i \geq d \end{cases}$$

where $B_2()$ encodes the set $\{0, \ldots, d-1\}$ equipped with the uniform distribution (easily derived from the integers binary expansion). The best value of d depends of n and t (it is thus different for every recursive call). Choosing a different value of d is possible but suboptimal in terms of compression rate.

There is a good trade-off when one uses only powers of 2 for d, there is a small loss in average, but a significant advantage in speed.

The recursive method is significantly faster than the enumerative method: the computation time is linear in ℓ instead of quadratic. However the encoder $\Psi_{n,t} : \mathcal{W}_{n,t} \to \{0,1\}^*$ produces a variable length output.

Comments and implementation.

The enumerative method allows constant length encoding with a minimal loss ($\ell = \lfloor \binom{n}{t} \rfloor$). On the other hand, it is relatively slow, even when the binomial coefficients are precomputed. The recursive method can be much faster, however the encoder $\mathcal{W}_{n,t} \to \{0,1\}^*$ has an important length variation. This is unpractical and not recommendable, as it raises some security issues that need to be studied further. For instance if an adversary knows how many bits were used to produce the error, he might be able to use this information. The Table 3 gives the average running time for a $\phi_{n,t}$ (and for its inverse $\phi_{n,t}^{-1}$) build from both methods.

(n,t)	(2048,32)		(2048,40)		(4096,22)		(4096,45)	
	$\phi_{n,t}$	$\phi_{n,t}^{-1}$	$\phi_{n,t}$	$\phi_{n,t}^{-1}$	$\phi_{n,t}$	$\phi_{n,t}^{-1}$	$\phi_{n,t}$	$\phi_{n,t}^{-1}$
enumerative	1980	1550	2530	2090	1440	1080	3160	2750
enumerative[(1)]	560	200	580	210	490	200	620	290
recursive	240	250	250	250	240	230	230	240
recursive[(2)]	150	150	150	150	135	130	140	140

(1) enumerative method with precomputation of the binomial coefficients
(2) recursive method optimized for speed (*vs.* average length)

Table 3. Performance (cycles/byte, Intel Core 2) for various encoding methods

Remark 7. There is another proposal [61] which uses arithmetic coding. It is essentially the same as the enumerative method. It is not clear whether or not this algorithm allows a faster implementation.

Remark 8. A new approach has been considered very recently[8] which allows linear time encoding (around 300 cycles/byte on a processor Intel Core 2) with an optimal constant length. At the time of writing, this work was at a too early stage to be detailed here.

[8] see `http://www-rocq.inria.fr/secret/MCE`

Niederreiter's encryption scheme

Using a systematic public key for Niederreiter's scheme was already known to be harmless [13]. The decoder $D'_{\mathcal{G}} : \{0,1\}^k \to \mathcal{W}_{n,t}$ is slightly different, it takes as argument a syndrome (for $\mathsf{H} = (\mathsf{R}^\top \mid \mathsf{Id})$) and returns an error pattern. The implementation is presented in Algorithm 5.3.

Algorithm 5.3 Modified Niederreiter encryption scheme

- **Public key:** a $k \times (n-k)$ binary matrix R
- **Private key:** a decoder $D'_{\mathcal{G}}$ for the code $\mathcal{G}$ spanned by $(\mathsf{Id} \mid \mathsf{R})$
- **Encryption:** the plaintext is $\mathbf{m} \in \{0,1\}^\ell$
  compute the error $\mathbf{e} = \phi_{n,t}(\mathbf{m}) = (\mathbf{e}_1, \mathbf{e}_2) \in \{0,1\}^k \times \{0,1\}^{n-k}$
  the ciphertext is $\mathbf{s} = \mathbf{e}\left(\mathsf{R}^\top \middle| \mathsf{Id}\right)^\top = \mathbf{e}_1\mathsf{R} + \mathbf{e}_2 \in \{0,1\}^{n-k}$
- **Decryption:** the ciphertext is $\mathbf{s} \in \{0,1\}^{n-k}$
  the plaintext is $\mathbf{m} = \phi_{n,t}^{-1}(D'_{\mathcal{G}}(\mathbf{s}))$

Timings and sizes

In Table 4, numbers for McEliece and Niederreiter encryption schemes are given. They come from `http://www-rocq.inria.fr/secret/MCE`. Implementation uses a systematic public key and information is encoded in the error.

5.2 Reducing storage requirements

Reducing the key size for the McEliece PKC has a long history. Besides the approaches to use different codes than Goppa codes, there were two different attempts: The first uses the automorphism group of Goppa codes [44] and the second the quasi-cyclicity of codes [28]. While the first method was broken [38], the second reduces the number of possible secret keys. However, the quasi-cyclic approach has an interesting application in Stern's ID scheme, reducing the RAM requirements of the scheme. However, the proposal is too recent and further research is probably needed to establish secure parameter sets.

Definition 8. *An $[n,k,d]$ code $\mathcal{G}$ over $\mathbb{F}$ is called s-quasi cyclic if for all $\mathbf{c} \in \mathcal{G}$ the vector $\sigma_s(\mathbf{c})$ is in $\mathcal{G}$, where*

$$\begin{aligned} \sigma_s : \mathbb{F}^n &\to \mathbb{F}^n \\ (c_1, \cdots, c_n) &\mapsto (c_{n-s+1}, \cdots, c_n, c_1, \cdots, c_{n-s}) \end{aligned}$$

denotes a cyclic shift by s positions. If $s = 1$ or $s \times n$ the code is cyclic. A set of vectors G is called generating set if the vectors $\left\{\sigma_s^i(\mathbf{c}) \mid \mathbf{c} \in \mathsf{G}, i \in \mathbb{N}_+\right\}$ span $\mathcal{G}$, where $\sigma_s^i(\mathbf{c}) = \sigma_s(\sigma_s^{i-1}(\mathbf{c}))$.

	(n,t)	(2048,32)	(2048,40)	(4096,22)	(4096,45)
McEliece scheme	plaintext size[(1)]	1928	1888	4024	3904
	ciphertext size[(1)]	2048	2048	4096	4096
	encryption rate[(2)]	176	222	145	192
	decryption rate[(2)]	1780	2260	600	1650
Niederreiter scheme	plaintext size[(1)]	232	280	192	352
	ciphertext size[(1)]	352	440	264	540
	encryption rate[(2)]	360	370	320	340
	decryption rate[(2)]	13600	16700	9800	16900
	public key size[(3)]	73 KB	86 KB	123 KB	234 KB
	key generation[(3)]	$6.70\,10^7$	$9.55\,10^7$	$7.93\,10^7$	$23.1\,10^7$
	security bits[(3)(4)]	91	98	93	140

[(1)] plaintext and ciphertext sizes in bits
[(2)] in cycles per plaintext bytes, Intel Core 2
[(3)] common to both schemes (number of cycles on a processor Intel Core 2)
[(4)] $\log_2$ of the non-quantum binary workfactor

Table 4. McEliece and Niederreiter encryption scheme

Every cyclic code is s-quasi cyclic for all $s \in \mathbb{N}_+$ and the dual of a s-quasi cyclic code is s-quasi cyclic, too. Each cyclic code has a s-cyclic subcode that is not s'-cyclic for all $s' < s$ if $s|n$.

If one chooses to use a secret s-quasi cyclic $[n,k]$ code with $s|n$ for McEliece's scheme and restricts the possible choice of permutation matrices P to the ones which are of the form

$$\mathsf{P} = \begin{bmatrix} \pi & 0 & \cdots & 0 \\ 0 & \pi & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \pi \end{bmatrix},$$

where π is a $s \times s$ matrix. Then, a systematic generator matrix of $\mathcal{G}$ can be reconstructed from $\mathsf{G}^{\text{pub}} = \mathsf{G} \cdot \mathsf{P}$. Thus, the public key size can reduced by a certain factor.

However, this technique holds some risks, as the number of possible permutations is reduced and part of the structure is revealed (a first approach to attack such a system was reported by A. Otmani, J.P. Tillich and L. Dallot [16]). Second, general decoding algorithms could take advantage of the structure of the code, as it is e.g. the case for iterative or statistical decoding [20, 55]. Third, let $\mathbf{e}$ be an error vector of weight t, H be the generating set of the dual of $\mathcal{G}$ and $\mathbf{s} = \mathbf{e}\mathsf{H}$. Then a cyclic shift of $\mathbf{s}$ by one corresponds to the vector $\sigma_s(\mathbf{e})$.

For Stern's ID scheme, one could chose to use 2-quasi cyclic codes with a single generating vector as proposed in [29]. This reduces drastically the size of memory needed to execute the scheme (from kn to n). However, as for Stern's scheme only a random code is required, one could build the generating matrix G from a random string as well, if a cryptographic strong random number generator is used. This has the same effect of reducing the size of memory needed but does not come with the disadvantage of a quasi cyclic code.

5.3 Semantic security for the McEliece scheme

The McEliece PKC and the Niederreiter scheme are subject to several attacks if not completely random bit-strings are sent. Thus, the schemes as they are only serve for key-agreement protocols and not for encrypting messages. In this section we will point out the weaknesses of the McEliece scheme (and thus the Niederreiter version) against attacks on the semantic security and how to get a semantically secure cryptosystem.

A cryptosystem is called *secure against adaptive chosen ciphertext attacks* (CCA2 secure) if an attacker with access to a decryption oracle (which does not decrypt the ciphertext $\mathbf{c}$) has no advantage in deciphering a given ciphertext $\mathbf{c}$. A PKC is *indistinguishable in the CCA2-model* if the attacker has no advantage in determining for a given ciphertext and two plaintexts which of them was encrypted.

Weaknesses of the McEliece PKC

The main weakness of the McEliece PKC results from the malleability of its ciphertexts. Adding codewords, i.e. rows of $\mathsf{G}^{\mathrm{pub}}$ to a ciphertext yields another valid ciphertext. Therefore, the original McEliece cryptosystem does not satisfy non-malleability. A CCA2 attack can be derived immediately as the adversary can add a second message $\mathbf{m}'$ to $\mathbf{c}$ by computing $\mathbf{c}' = \mathbf{c} \oplus \mathbf{m}'\mathsf{G}^{\mathrm{pub}}$, which will be decrypted by the oracle. Note that malleability is not such a problem in the Niederreiter case, as we can not create new decodable syndromes from old ones with probability significantly larger than t/n.

As a consequence from the malleability, an adversary for the McEliece scheme may use the relation between two encrypted messages to determine error bits [8]. This attack can not be adapted to the Niederreiter cryptosystem. Let $\mathbf{m}_1, \mathbf{m}_2$ be two messages with a known relation Λ, e.g. $\Lambda(\mathbf{m}_1, \mathbf{m}_2) = \mathbf{m}_1 \oplus \mathbf{m}_2$ and $\mathbf{c}_1, \mathbf{c}_2$ the corresponding ciphertexts. Then $\mathbf{c}_1 \oplus \mathbf{c}_2 \oplus \Lambda(\mathbf{m}_1, \mathbf{m}_2)$ will be of weight $\leq 2t \leq n - k$ and at least k error-free positions of $\mathbf{m}_1 \oplus \mathbf{m}_2$ are revealed. This enables an adversary to efficiently guess error bits. A special case of related messages occurs in the *message-resend attack*, where the attacker can recover $\mathbf{z}_1 \oplus \mathbf{z}_2 = \mathbf{c}_1 \oplus \mathbf{c}_2$.

A reaction attack is a weaker version of an adaptively chosen ciphertext attack, in that the attacker does not have access to a full decryption oracle, but

can only observe the receiver's reaction on potential ciphertexts. An adversary may intercept ciphertexts, change a few bits, and watch the reaction of the designated receiver on these modified ciphertexts. Sending modifications of an authentic ciphertext amounts to adding further error bits. If the receiver cannot decode (reaction: repeat request), the corresponding bits were not in error originally. This enables the attacker to recover an error-free information set in at most k iterations. Observe, that such an attack is well possible for the Niederreiter PKC as it does not require the malleable property.

CCA2-secure versions of the McEliece scheme

In [37] Kobara and Imai review possible conversions to turn the McEliece PKC CCA2-secure. Not all generic conversions can be applied to the McEliece PKC, since the McEliece PKC encryption function is not a OWTP (one-way-trapdoor permutation) and it is vulnerable against message-resend attacks.

However, there are two generic conversions, which are applicable to the McEliece PKC: One presented by Pointcheval [60] and the other by Fujisaki and Okamoto [21]. These conversions are valid for all encryption schemes, which are *partially trapdoor one-way* (PTOWF), i.e., the encryption is a function $f : X \times Y \to Z,\ (x, y) \mapsto z$ where it is impossible to recover x or y from their image z alone, but the knowledge of secret enables a partial inversion, i.e. finding x from z. Pointcheval [60] demonstrated how any PTOWF can be converted to a public-key cryptosystem that is indistinguishable against CCA2, while the conversion of Fujisaki and Okamoto is applicable to those schemes which are one-way encryptions (OWE), which includes PTOWF and OWTP.

We omit giving details on generic conversions, since they add a large amount of redundancy to the cipher texts. Instead we focus on the McEliece-specific conversions presented by Kobara and Imai, whose main concern is to decrease data overhead. As an example we present the "γ-conversion" based on Algorithm 2.1. For the ease of presentation we introduce the notations given in Table 5. The γ-conversion is summarized in Algorithm 5.4. It is assumed that $\text{length}(\mathbf{m}) \geq \log_2 \lfloor \binom{n}{t} \rfloor + k - \text{length}(const) - \text{length}(r)$.

For large messages Kobara and Imai achieve a reduction in data redundancy even below the values for the original McEliece PKC for large parameters. For example, for $m = 11, t = 70$ the message size is expanded by 655 bits instead of 770 in the original McEliece scheme. The security of the γ-conversion can be reduced to the one of the original scheme [37]:

Theorem 2. *Breaking indistinguishability in the CCA2 model using any of the conversions presented above, is as hard as breaking the original McEliece public key system.*

Symbol	Function
ℓ	$\lfloor \log_2 \binom{n}{t} \rfloor$.
H	Cryptographic secure hashing to a ℓ-bit string
R	Cryptographically secure pseudo random number generator from fixed length seeds
$E_{(\mathsf{G}^{\mathrm{pub}},t)}$	McEliece encryption function, taking as first argument the message to be encrypted and as second one the error vector: $E_{(\mathsf{G}^{\mathrm{pub}},t)}(\mathbf{m},\mathbf{z}) = \mathbf{c}$
$D_{(S,D_\mathcal{G},P)}$	McEliece decryption function: $D_{(S,D_\mathcal{G},P)}(c) = (\mathbf{m},\mathbf{z})$
$MSB_n(\mathbf{m})$	The n rightmost bits of $\mathbf{m}$.
$LSB_n(\mathbf{m})$	The n leftmost bits of $\mathbf{m}$.

Table 5. Notation for Algorithm 5.4.

Algorithm 5.4 Kobara-Imai's γ Conversion

- **Additional System Parameters:** length($\mathbf{r}$), the length of the random seed and a constant *const*.
- **Encryption $E^{\gamma}_{(\mathsf{G}^{\mathrm{pub}},t)}$:**

  Generate a random seed $\mathbf{r}$ of length length($\mathbf{r}$).
  Set
  $\mathbf{c}_1 = \mathrm{PRG}(\mathbf{r}) \oplus (\mathbf{m}, const)$, $\mathbf{c}_2 = \mathbf{r} \oplus H(\mathbf{c}_1)$,
  $\mathbf{c}_3 = LSB_{\ell+k}(\mathbf{c}_2,\mathbf{c}_1)$, $\mathbf{c}_4 = LSB_k(\mathbf{c}_3)$,
  $\mathbf{c}_5 = MSB_\ell(\mathbf{c}_3)$, $\mathbf{z} = \phi_{n,t}(\mathbf{c}_5)$
  if $\mathrm{length}(\mathbf{c}_2,\mathbf{c}_1) - \ell - k > 0$ **then**
    $\mathbf{c}_6 = MSB_{\mathrm{length}(\mathbf{c}_2,\mathbf{c}_1)-\ell-k}(\mathbf{c}_2,\mathbf{c}_1)$
    $\mathbf{c} = (\mathbf{c}_6, E_{(\mathsf{G}^{\mathrm{pub}},t)}(\mathbf{c}_4,\mathbf{z}))$
  else
    $\mathbf{c} = E_{(\mathsf{G}^{\mathrm{pub}},t)}(\mathbf{c}_4,\mathbf{z})$
- **Decryption $D^{\gamma}_{(S,\boldsymbol{D}_\mathcal{G},P)}$:**

  Set
  $\mathbf{c}_6 = MSB_{Len(\mathbf{c})-n}(\mathbf{c})$, $(\mathbf{c}_4,\mathbf{z}) = D_{(S,D_\mathcal{G},P)}(LSB_n(\mathbf{c}))$,
  $\mathbf{c}_5 = \phi^{-1}_{n,t}(\mathbf{z})$, $\mathbf{c}_2 = MSB_{\mathrm{length}(\mathbf{r})}(\mathbf{c}_6,\mathbf{c}_5,\mathbf{c}_4)$,
  $\mathbf{c}_1 = LSB_{\mathrm{length}(\mathbf{c})-\mathrm{length}(\mathbf{r})}(\mathbf{c}_6,\mathbf{c}_5,\mathbf{c}_4)$,
  $(\mathbf{m}, const') = (\mathbf{c}_1) \oplus \mathrm{PRG}(\mathbf{c}_2 \oplus H(\mathbf{c}_1))$
  if $const' = const$ **then**
    return m
  else
    reject c

Furthermore, all adaptive attacks become impossible, since relations among plaintexts do no longer result in relations among ciphertexts. Already the simple hashing of messages before encryption prevents this.

6 Annex

6.1 Algebraic coding theory

Hamming distance and linear codes.

Let $\mathbb{F}_q$ be a finite field. The Hamming distance between two words $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{F}_q^n$ is defined to be the number of coordinates in which $\mathbf{x}$ and $\mathbf{y}$ differ. The Hamming weight $\mathrm{wt}(\mathbf{x})$ of $\mathbf{x}$ is the number of non-zero coordinates of $\mathbf{x}$. A *code* is a non-empty subset of the Hamming space $\mathbb{F}_q^n$. A k-dimensional subspace of $\mathbb{F}_q^n$ is called a $[n, k]$ *linear code* over $\mathbb{F}_q$.

Generator and parity check matrices.

Let $\mathcal{C}$ denote an $[n, k]$ linear code over $\mathbb{F}_q$.

- A *generator matrix* G for $\mathcal{C}$ is a matrix over $\mathbb{F}_q$ such that $\mathcal{C} = \langle \mathsf{G} \rangle$, where $\langle \mathsf{G} \rangle$ denotes the vector space spanned by the rows of G. Usually, the rows of G are independent and the matrix is $k \times n$. A generator matrix G is said to be in *systematic form*, if its first k columns form the identity matrix.
- The *dual code* $\mathcal{C}^\perp$ of $\mathcal{C}$ is the orthogonal of $\mathcal{C}$ for the usual scalar product over $\mathbb{F}_q$. It is a $[n, n-k]$ linear code over $\mathbb{F}_q$.
- A *parity check matrix* H of $\mathcal{C}$ is a generator matrix of $\mathcal{C}^\perp$.

Minimum distance and weight.

Let $\mathcal{C}$ denote an $[n, k]$ linear code over $\mathbb{F}_q$. The *minimum distance* $d = \mathrm{dmin}(\mathcal{C})$ of $\mathcal{C}$ is the smallest Hamming distance between distinct codewords. For a linear code, it is equal to the *minimum weight*, the smallest non-zero weight of a codeword. We will speak of an $[n, k, d]$ code.

Decoder.

A decoder for $\mathcal{C}$ is a mapping $D_\mathcal{C} : \mathbb{F}_q^n \to \mathcal{C}$. It is t-error correcting if for all $\mathbf{e} \in \mathbb{F}_q^n$ and all $\mathbf{x} \in \mathcal{C}$

$$\mathrm{wt}(\mathbf{e}) \leq t \Rightarrow D_\mathcal{C}(\mathbf{x} + \mathbf{e}) = \mathbf{x}$$

For any $[n, k, d]$ linear code, there exist a t-error correcting decoder if and only if $t < d/2$.

Weight enumerator polynomial.

For a linear code $\mathcal{C}$, it is defined as

$$W(\mathcal{C})(X) = \sum_{\mathbf{c} \in \mathcal{C}} X^{\mathrm{wt}(\mathbf{c})} = \sum_{i=0}^{n} A_i X^i$$

where A_i is the number of codewords of Hamming weight i.

Support.

The *support* $\mathcal{I}$ of a code of length n is an ordered set of cardinality n used to index the coordinates. Typically $\mathcal{I} = \{1, \ldots, n\}$, but it is sometimes convenient to index the coordinates with another ordered set (in Goppa codes for instance). The support of a codeword is the subset of $\mathcal{I}$ containing its non-zero coordinates.

Puncturing.

Let $\mathcal{C}$ be an $[n, k]$ linear code of support $\mathcal{I}$, let G be a generator matrix of $\mathcal{C}$, and let $\mathcal{J}$ be a subset of $\mathcal{I}$.

- Punctured matrix: we denote by $\mathsf{G}_{.\mathcal{J}}$ the $k \times |\mathcal{J}|$ matrix obtained from G by keeping the columns indexed by $\mathcal{J}$.
- Punctured code: We denote by $\mathcal{C}_{.\mathcal{J}}$ the code obtained by retaining in all codeword of $\mathcal{C}$ the coordinates indexed by $\mathcal{J}$.

Note that $\mathcal{C}_{.\mathcal{J}} = \langle \mathsf{G}_{.\mathcal{J}} \rangle$ (i.e. the punctured matrix spans the corresponding punctured code).

Subcodes.

Any linear subspace of $\mathcal{C}$ is said to be a subcode of $\mathcal{C}$. If $\mathcal{C}$ is a code over $\mathbb{F}$ and $\mathbb{F}_{\mathrm{SUB}}$ is a subfield of $\mathbb{F}$, then the $\mathbb{F}_{\mathrm{SUB}}$-(subfield) subcode of $\mathcal{C}$ is the code consisting of all words of $\mathcal{C}$, which have only entries in $\mathbb{F}_{\mathrm{SUB}}$. A $\mathbb{F}_{\mathrm{SUB}}$-subfield subcode is a $\mathbb{F}_{\mathrm{SUB}}$-linear code. As codes are treated as vector spaces, we will often define them by the matrices related to the code.

6.2 GRS and Goppa codes

An important class of codes are the GRS codes, which are strongly related to the class of Goppa codes used by McEliece to define his cryptosystem. Thus, we briefly introduce them:

Definition 9. *A GRS code over $\mathbb{F}_{q^m}$ of length n with designed minimum Hamming distance $t+1$ is defined by two vectors $\mathbf{a}, \mathbf{z} \in \mathbb{F}_{q^m}^n$, where $a_i \neq a_j$ for $i \neq j$ and all $z_i \neq 0$. The canonical check matrix of the GRS code is of the form*

$$\mathsf{H}^\top = \begin{pmatrix} z_1 a_1^0 & z_1 a_1^1 & \cdots & z_1 a_1^{t-1} \\ z_2 a_2^0 & z_2 a_2^1 & \cdots & z_2 a_2^{t-1} \\ \vdots & & \ddots & \vdots \\ z_n a_n^0 & z_n a_n^1 & \cdots & z_n a_n^{t-1} \end{pmatrix} \in \mathbb{F}_{q^m}^{n \times t}. \tag{10}$$

The code with check matrix

$$\left[\frac{\mathsf{H}^\top}{0\cdots\ 0\ 1}\right]^\top$$

is called an extended GRS code.

The $\mathbb{F}_q$-subfield subcode of a GRS code is called an *alternant code* and has dimension $k \geq n - mt$. If for a GRS code, there exists a polynomial $g \in \mathbb{F}_{q^m}[X]$ of degree t, for which $g(a_i) = 1/z_i$, the polynomial is called *Goppa polynomial* and the $\mathbb{F}_q$-subfield subcode is called *Goppa code* (see e.g. [47] or [17]). An equivalent definition is the following:

Definition 10. *A* binary Goppa code $\mathcal{G}$ *over* $\mathbb{F}_{2^m}$ *is defined by a vector* $\mathbf{a} \in \mathbb{F}_{2^m}^n$*, where* $a_i \neq a_j$ *and the Goppa polynomial* $g(X) = \sum_{i=0}^{t} g_i X^i \in \mathbb{F}_{2^m}[X]$*.* $\mathcal{G}$ *is the set of all* $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{n-1}) \in \mathbb{F}_2^n$ *such that the identity*

$$S_{\mathbf{c}}(X) = -\sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{g(a_i)} \frac{g(X) - g(a_i)}{X - a_i} \mod g(X) = 0 \tag{11}$$

holds in the polynomial ring $\mathbb{F}_{2^m}[X]$ *or equivalently if*

$$S_{\mathbf{c}}(X) \equiv \sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{X - a_i} \equiv 0 \mod g(X). \tag{12}$$

Oftentimes, the vector $\mathbf{a}$ *is called* γ *or* $\mathbf{L}$ *and since* $\mathcal{G}$ *is defined in function of* $\mathbf{L}$ *and the Goppa polynomial we write:* $\mathcal{G} = \Gamma(\mathbf{L}, g)$*.*

If the Goppa polynomial is irreducible, then the Goppa code has minimum distance $2 \cdot t + 1$ and is called an *irreducible Goppa code*.

The coefficients of the syndrome polynomial $S_{\mathbf{c}}(X) = \sum_{i=0}^{t-1} s_i X^i$ of a vector $\mathbf{c}$ in a Goppa code may be computed via equation (13), where H is given in equation (10) with $z_i = 1/g(a_i)$.

$$\begin{pmatrix} s_0 & s_1 & \cdots & s_{t-1} \end{pmatrix} = \mathbf{c}\mathsf{H}^\top \begin{pmatrix} g_t & 0 & \cdots & 0 \\ g_{t-1} & g_t & \ddots & 0 \\ \vdots & & \ddots & \vdots \\ g_1 & g_2 & \cdots & g_t \end{pmatrix} \tag{13}$$

For GRS codes, as well as for Goppa codes, there exist algorithms for correcting errors of Hamming weight up to half of the minimum distance. Such algorithms take $\mathcal{O}(n^2)$ respectively $\mathcal{O}(n \cdot t \cdot m^2)$ binary operations, see e.g. [7, 58]. Here we present Patterson's algorithm for correcting errors in irreducible binary Goppa codes, where we follow the presentation in [17]: Let $\mathbf{m}$ be a codeword, $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathrm{wt}(\mathbf{e}) \leq t$ an error vector, and $\mathbf{c} = \mathbf{m} \oplus \mathbf{e}$. Since $S_{\mathbf{m}}(X) \equiv 0 \mod g(X)$, we have

$$0 \neq S_{\mathbf{c}}(X) \equiv S_{\mathbf{e}}(X) \mod g(X).$$

We introduce the *error locator polynomial* $\sigma_{\mathbf{e}}(X)$ of $\mathbf{e}$ as

$$\sigma_{\mathbf{e}}(X) := \prod_{j \in \mathcal{T}_{\mathbf{e}}} (X - \gamma_j) \in \mathbb{F}_{2^m}[X],$$

where $\mathcal{T}_{\mathbf{e}}$ is the support of $\mathbf{e}$. From (12), it follows that

$$\sigma_{\mathbf{e}}(X) S_{\mathbf{e}}(X) \equiv \sigma'_{\mathbf{e}}(X) \mod g(X). \tag{14}$$

We split $\sigma_{\mathbf{e}}(X)$ in squares and non-squares:

$$\sigma_{\mathbf{e}}(X) = \alpha^2(X) + X\beta^2(X).$$

Since the characteristic of the field is 2, we have $\sigma'_{\mathbf{e}}(X) = \beta^2(X)$. Setting $T(X) = S_{\mathbf{e}}^{-1}(X)$ and multiply equation (14) by $T(X)$ we obtain

$$\beta^2(X)(X + T(X)) \equiv \alpha^2(X) \mod g(X) \tag{15}$$

Each element of $\mathbb{F}_{2^{mt}}$ has a unique square root. Let $\tau(X) \in \mathbb{F}_{2^m}[X]$ be the square root of $T(X) + X$, then

$$\beta(X)\tau(X) \equiv \alpha(X) \mod g(X).$$

The equation above can be solved: We have to determine $\alpha(X)$ and $\beta(X)$ of least degree, i.e. with $\deg(\alpha(X)) \leq \lfloor t/2 \rfloor$ and $\deg(\beta(X)) \leq \lfloor (t-1)/2 \rfloor$. Computing the inverse of $\tau(X)$ modulo $g(X)$ via the extended Euclidean algorithm and stopping it in mid-time gives the (unique) solution [33, 43, 47]. Finally, the zeroes of $\sigma_{\mathbf{e}}(X) = \alpha^2(X) + X\beta^2(X)$ can be determined, which reveals $\mathbf{e}$.

The runtime of the presented error correction algorithm may be estimated as follows. To compute the syndrome $S_{\mathbf{c}}(X)$ employing the check matrix H, we need at most $(n-k)n$ binary operations. To compute $T(X)$, we employ the extended Euclidean algorithm. This takes $\mathcal{O}\left(t^2m^2\right)$ binary operations, as the computations are modulo $g(X)$, a polynomial of degree t and coefficients of size m. Computing the square root of $T(X)+X$ takes $\mathcal{O}\left(t^2m^2\right)$ operation since it is a linear mapping on $\mathbb{F}_{2^m}[X]/g(X)$. The subsequently employed variant of the extended Euclidean algorithm takes $\mathcal{O}\left(t^2m^2\right)$ binary operations, too. These steps are fast in comparison to the last step to find all roots of the error locator polynomial. The latter can be performed in $n(tm^2 + tm)$ binary operations. Since $mt \geq (n-k)$, the error correction algorithm needs

$$\mathcal{O}\left(n \cdot t \cdot m^2\right)$$

binary operations. However, verifying, that an unique error locator polynomial exists requires only

$$\mathcal{O}\left(m^3t^2\right)$$

if the syndrome is already known.

6.3 Rank Distance

Not all codes are used with the Hamming metric. Here, we introduce a metric, which allows to correct "crisscross" errors in memory chip arrays or in magnetic tape recording, see [9, 41]:

Definition 11. *Let* $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbb{F}_{q^m}^n$ *and* $b_1, \cdots, b_m$ *a basis of* $\mathbb{F}_{q^m}$ *over* $\mathbb{F}_q$. *We can write* $x_i = \sum_{j=1}^{m} x_{ij} b_j$ *for each* $i = 1, \cdots, n$ *with* $x_{ij} \in \mathbb{F}_q$. *The rank norm* $\|\cdot\|_q$ *is defined as follows:*

$$\|\mathbf{x}\|_q := \operatorname{rank}\left((x_{ij})_{1 \le i \le n,\ 1 \le j \le m}\right) .$$

There are more isometries preserving rank distance than Hamming distance since all invertible matrices over the base field are isometries for the rank metric. The Syndrome Decoding Problem seems to be much harder in rank metric than in Hamming metric. In [36] Ourivski and Johansson presented two algorithms which solve the general decoding problem in $\mathcal{O}\left((k + \frac{d-1}{2})^3 (\frac{d-1}{2})^3 \times q^{(d-3)(m-(d-1)/2)/2}\right)$, respectively $\mathcal{O}\left((m\frac{d-1}{2})^3 q^{(d-3)(k+1)/2}\right)$ operations over $\mathbb{F}_q$ for $[n, k, d]$ rank distance codes over $\mathbb{F}_{q^m}$.

Even if rank distance codes can not be used to build a PKC (compare §4.3), the introduction of the rank metric into cryptography is interesting and might be useful, as it could, e.g., allow to reduce the key sizes for Stern's identification scheme or strengthen the FSB hash. The interested reader may find more information on the aspects of rank metric in [23, 25, 36, 45].

References

1. Alabbadi, M. and Wicker, S.: A digital signature scheme based on linear error-correcting block codes. In *ASIACRYPT '94*, volume LNCS 917, pages 238–248 (Springer 1995).
2. Assmus, Jr, E.F. and Key, J.D.: Affine and projective planes. *Discrete Mathematics*, 83:161–187 (1990).
3. Augot, D., Finiasz, M., and N.Sendrier: A family of fast syndrome based cryptographic hash functions. In *Proc. of Mycrypt 2005*, volume 3715 of *LNCS*, pages 64–83 (2005).
4. Barg, A.: Complexity issues in coding theory. In V.S. Pless and W.C. Huffman, editors, *Handbook of Coding theory*, volume I, chapter 7, pages 649–754. North-Holland (1998).
5. Berger, T. and Loidreau, P.: Security of the Niederreiter form of the GPT public-key cryptosystem. In *IEEE International Symposium on Information Theory, Lausanne, Suisse*. IEEE (July 2002).
6. Berlekamp, E., McEliece, R., and van Tilborg, H.: On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386 (1978).
7. Berlekamp, E.: *Algebraic coding theory*. McGraw-Hill, New York (1968).

8. Berson, T.: Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In *Proceedings of CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer Verlag (1997).
9. Blaum, M. and McEliece, R.J.: Coding protection for magnetic tapes: A generalization of the Patel - Hong code. *IEEE Transactions on Information Theory*, 31(5):690– (1985).
10. Camion, P. and Patarin, J.: The knapsack hash function proposed at Crypto'89 can be broken. In D.W. Davies, editor, *Advances in Cryptology - EUROCRYPT'91*, number 547 in LNCS, pages 39–53. Springer-Verlag (1991).
11. Canteaut, A. and Chabaud, F.: Improvements of the attacks on cryptosystems based on error-correcting codes. *Rapport interne du Departement Mathematiques et Informatique*, LIENS-95-21 (1995).
12. Canteaut, A. and Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEETIT: IEEE Transactions on Information Theory*, 44 (1998).
13. Canteaut, A. and Sendrier, N.: Cryptanalysis of the original McEliece cryptosystem. In *Advances in Cryptology - ASIACRYPT '98 Proceedings*, pages 187–199. Springer-Verlag (1998).
14. Courtois, N., Finiasz, M., and N.Sendrier: How to achieve a McEliece-based digital signature scheme. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248, pages 157–174. Springer-Verlag (2001).
15. Cover, T.: Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77 (1973).
16. Dallot, L., Tillich, J., Otmani, A.: Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes (2008). CoRR, abs/0804.0409, available at http://arxiv.org/abs/0804.0409 (2008).
17. Engelbert, D., Overbeck, R., and Schmidt, A.: A summary of McEliece-type cryptosystems and their security. *Journal of Mathematical Cryptology*, 1(2):151–199 (2007).
18. Finiasz, M.: *Nouvelles constructions utilisant des codes correcteurs d'erreurs en cryptographie à clef publique.* Thèse de doctorat, École Polytechnique (2004).
19. Fischer, J.B. and Stern, J.: An eficient pseudo-random generator provably as secure as syndrome decoding. In U.M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 245–255. Springer-Verlag (1996).
20. Fossorier, M., Imai, H., and Kobara, K.: Modeling bit flipping decoding based on non orthogonal check sums and application to iterative decoding attack of McEliece cryptosystem. In *Proc. of 2004 International Symposium on Information Theory and its Applications, Parma, Italy (ISITA'04)* (October 2004).
21. Fujisaki, E. and Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In *Proc. of CRYPTO*, volume 547 of *LNCS*, pages 535–554. Springer Verlag (1999).
22. Gabidulin, E.M. and Ourivski, A.V.: Column scrambler for the GPT cryptosystem. *Discrete Applied Mathematics*, 128(1):207–221 (2003).
23. Gabidulin, E.: Theory of codes with maximum rank distance. *Problems of Information Transmission*, 21, No. 1 (1985).
24. Gabidulin, E.: On public-key cryptosystems based on linear codes. In *Proc. of 4th IMA Conference on Cryptography and Coding 1993*, Codes and Ciphers. IMA Press (1995).

25. Gabidulin, E. and Loidreau, P.: Subfield subcodes of maximum-rank distance codes. In *Seventh International Workshop on Algebraic and Combinatorial Coding Theory*, volume 7 of *ACCT*, pages 151–156 (2000).
26. Gabidulin, E., Ourivski, A., Honary, B., and Ammar, B.: Reducible rank codes and their applications to cryptography. *IEEE Transactions on Information Theory*, 49(12):3289–3293 (2003).
27. Gabidulin, E., Paramonov, A., and Tretjakov, O.: Ideals over a non-commutative ring and their applications to cryptography. In *Proc. Eurocrypt '91*, volume 547 of *LNCS*. Springer Verlag (1991).
28. Gaborit, P.: Shorter keys for code based cryptography. In *Proc. of WCC 2005*, pages 81–90 (2005).
29. Gaborit, P. and Girault, M.: Lightweight code-based authentication and signature. In *Proc. of ISIT 2007* (2007).
30. Gaborit, P., Laudaroux, C., and Sendrier, N.: Synd: a very fast code-based cipher stream with a security reduction. In *IEEE Conference, ISIT'07*, pages 186–190. Nice, France (2007).
31. Gibson, K.: Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem. In D.W. Davies, editor, *Advances in Cryptology - Eurocrypt'91*, volume 547 of *LNCS*, pages 517–521. Springer Verlag (1991).
32. Harn, L. and Wang, D.C.: Cryptanalysis and modification of digital signature scheme based on error-correcting codes. *Electronics Letters*, 28(2):157–159 (1992).
33. Heise and Quattrocchi: *Informations- und Codierungstheorie*. Springer Berlin Heidelberg, 3 edition (1995).
34. Jabri, A.K.A.: A statistical decoding algorithm for general linear block codes. In *Cryptography and Coding 2001*, volume 2260 of *LNCS*, pages 1–8. Springer Verlag (2001).
35. Janwa, H. and Moreno, O.: McEliece public key cryptosystems using algebraic-geometric codes. *Designes, Codes and Cryptography*, 8:293–307 (1996).
36. Johansson, T. and Ourivski, A.: New technique for decoding codes in the rank metric and its cryptography applications. *Problems of Information Transmission*, 38, No. 3:237–246 (2002).
37. Kobara, K. and Imai, H.: Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC. In *Practice and Theory in Public Key Cryptography - PKC '01 Proceedings*. Springer Verlag (2001).
38. Kobara, K. and Imai, H.: On the one-wayness against chosen-plaintext attacks of the Loidreau's modified McEliece PKC. *IEEE Transactions on Information Theory*, 49, No. 12:3160–3168 (2003).
39. Lee, P. and Brickell, E.: An observation on the security of McEliece's public key cryptosystem. In *Advances in Cryptology-EUROCRYPT'88*, volume 330 of *LNCS*, pages 275–280. Springer Verlag (1989).
40. Leon, J.: A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359 (1988).
41. Levine, L. and Myers, W.: Semiconductor memory reliability with error detecting and correcting codes. *COMPUTER*, 9(10):43–50 (1976). ISSN 0018-9162.
42. Li, Y., Deng, R., and Wang, X.: the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, Vol. 40, pp. 271-273 (1994).

43. Lidl, R. and Niederreiter, H.: *Introduction to finite fields and their applications.* Cambridge University Press, 2 edition (1986).
44. Loidreau, P.: Strengthening McEliece cryptosystem. In *Advances in Cryptology - ASIACRYPT '00 Proceedings*, pages 585–598. Springer Verlag (2000).
45. Loidreau, P. and Overbeck, R.: Decoding rank errors beyond the error-correction capability. In *Proc. of ACCT-10, Zvenigorod*, pages 168–190 (2006).
46. Loidreau, P. and Sendrier, N.: Weak keys in the McEliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47, No. 3:1207 –1211 (March 2001).
47. MacWilliams, F. and Sloane, N.: *The Theory of Error-Correctiong Codes.* North-Holland Amsterdam, 7 edition (1992).
48. McEliece, R.: A public key cryptosystem based on algebraic coding theory. *DSN progress report*, 42-44:114–116 (1978).
49. Minder, L.: *Cryptography based on error correcting codes.* Phd thesis, EPFL (2007).
50. Minder, L. and Shokrollahi, A.: Cryptanalysis of the Sidelnikov cryptosystem. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, number 4515 in LNCS, pages 347–360. Springer (2007).
51. Montpetit, A.: Note sur la notion d'équivalence entre deux codes linéaires. *Discrete Mathematics*, 65:177–185 (1987).
52. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control and Inform. Theory*, 15:19–34 (1986).
53. Overbeck, R.: Public key cryptography based on coding theory. Ph.D. Thesis, Available at http://elib.tu-darmstadt.de/diss/000823/.
54. Overbeck, R.: A new structural attack for GPT and variants. In *Proc. of Mycrypt 2005*, volume 3715 of *LNCS*, pages 50–63. Springer Verlag (2005).
55. Overbeck, R.: Statistical decoding revisited. In *Proc. of ACISP 2006*, volume 4058 of *LNCS*, pages 283–294. Springer Verlag (2006).
56. Overbeck, R.: Recognizing the structure of permuted reducible codes. In *Proc. of WCC 2007*, pages 269–276 (2007).
57. Overbeck, R.: Structural attacks for public key cryptosystems based on Gabidulin codes. *Journal of Cryptology*, 21(2):280–301 (2008).
58. Patterson, N.: Algebraic decoding of Goppa codes. *IEEE Trans. Info.Theory*, 21:203–207 (1975).
59. Petrank, E. and Roth, R.M.: Is code equivalence easy to decide? *IEEE Trans. on IT*, 43(5):1602–1604 (1997).
60. Pointcheval, D.: Chosen-ciphertext security for any one-way cryptosystem. In *Proc. of PKC*, volume 1751 of *LNCS*, pages 129–146. Springer Verlag (2000).
61. Ramabadran, T.V.: A coding scheme for m-out-of-n codes. *IEEE Transactions on Communications*, 38(8):1156–1163 (1990).
62. Schalkwijk, J.P.M.: An algorithm for source coding. *IEEE Transactions on Information Theory*, 18(3):395–399 (1972).
63. Sendrier, N.: Efficient generation of binary words of given weight. In C. Boyd, editor, *Cryptography and Coding ; proceedings of the 5th IMA conference*, number 1025 in LNCS, pages 184–187. Springer-Verlag (1995).
64. Sendrier, N.: On the concatenated structure of a linear code. *AAECC*, 9(3):221–242 (1998).
65. Sendrier, N.: *Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs.* Mémoire d'habilitation à diriger des recherches, Université Paris 6 (2002).

66. Sendrier, N.: On the security of the McEliece public-key cryptosystem. In M. Blaum, P. Farrell, and H. van Tilborg, editors, *Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday*, pages 141–163. Kluwer (2002).
67. Sendrier, N.: Encoding information into constant weight words. In *IEEE Conference, ISIT'2005*, pages 435–438. Adelaide, Australia (2005).
68. Sendrier, N.: Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46:1193–1203 (Jul 2000).
69. Sendrier, N.: On the dimension of the hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293 (May 1997).
70. Sidelnikov, V.: A public-key cryptosystem based on binary Reed-Muller codes. *Discrete Mathematics and Applications*, 4 No. 3 (1994).
71. Sidelnikov, V. and Shestakov, S.: On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2, No. 4:439–444 (1992).
72. Stern, J.: A method for finding codewords of small weight. *Coding Theory and Applications*, 388:106–133 (1989).
73. Stern, J.: A new identification scheme based on syndrome decoding. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *LNCS*. Springer Verlag (1994).
74. Stern, J.: Can one design a signature scheme based on error-correcting codes. In *ASIACRYPT '94*, volume 917 of *LNCS*, pages 424–426 (1995).
75. van Tilburg, J.: On the McEliece cryptosystem. In S. Goldwasser, editor, *Advances in Cryptology - CRYPTO'88*, number 403 in LNCS, pages 119–131. Springer-Verlag (1990).
76. Véron, P.: Improved identification schemes based on error-correcting codes. *Appl. Algebra Eng. Commun. Comput.*, 8(1):57–69 (1996).
77. Wagner, D.: A generalized birthday problem. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer (2002). ISBN 3-540-44050-X.
78. Wieschebrink, C.: An attack on a modified Niederreiter encryption scheme. In *Public Key Cryptography*, volume 3958 of *LNCS*, pages 14–26 (2006).
79. Xinmei, W.: Digital signature scheme based on error-correcting codes. *Electronics Letters*, 26(13):898–899 (1990).

Lattice-based Cryptography

Daniele Micciancio[1*] and Oded Regev[2†]

[1] CSE Department, University of California, San Diego.
[2] School of Computer Science, Tel-Aviv University.

1 Introduction

In this chapter we describe some of the recent progress in *lattice-based cryptography*. Lattice-based cryptographic constructions hold a great promise for post-quantum cryptography, as they enjoy very strong security proofs based on worst-case hardness, relatively efficient implementations, as well as great simplicity. In addition, lattice-based cryptography is believed to be secure against quantum computers. Our focus here will be mainly on the practical aspects of lattice-based cryptography and less on the methods used to establish their security. For other surveys on the topic of lattice-based cryptography, see, e.g., [36, 52, 60, 71] and the lecture notes [51, 67]. The survey by Nguyen and Stern [60] also describes some applications of lattices in cryptanalysis, an important topic that we do not discuss here. Another useful resource is the book by Micciancio and Goldwasser [49], which also contains a wealth of information on the computational complexity aspects of lattice problems.

So what is a lattice? A lattice is a set of points in n-dimensional space with a periodic structure, such as the one illustrated in Figure 1. More formally, given n-linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$, the lattice generated by them is the set of vectors

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \ : \ x_i \in \mathbb{Z} \right\}.$$

The vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are known as a *basis* of the lattice.

The way lattices can be used in cryptography is by no means obvious, and was discovered in a breakthrough paper by Ajtai [7]. His result has by now

* Supported in part by NSF Grant CCF 0634909.
† Supported by the Binational Science Foundation, by the Israel Science Foundation, by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848, and by a European Research Council (ERC) Starting Grant.

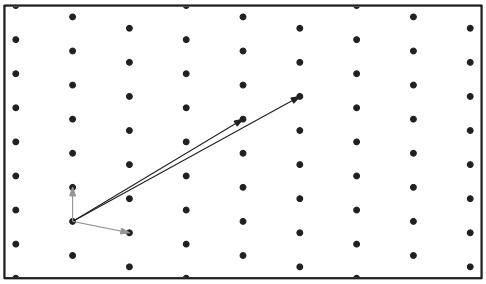

Fig. 1. A two-dimensional lattice and two possible bases.

developed into a whole area of research whose main focus is on expanding the scope of lattice-based cryptography and on creating more practical lattice-based cryptosystems. Before discussing this area of research in more detail, let us first describe the computational problems involving lattices, whose presumed hardness lies at the heart of lattice-based cryptography.

1.1 Lattice problems and algorithms

Lattice-based cryptographic constructions are based on the presumed hardness of lattice problems, the most basic of which is the *shortest vector problem* (SVP). Here, we are given as input a lattice represented by an arbitrary basis, and our goal is to output the shortest nonzero vector in it. In fact, one typically considers the approximation variant of SVP where the goal is to output a lattice vector whose length is at most some approximation factor $\gamma(n)$ times the length of the shortest nonzero vector, where n is the dimension of the lattice. A more precise definition of SVP and several other lattice problems appears in Section 2.

The most well known and widely studied algorithm for lattice problems is the *LLL algorithm*, developed in 1982 by Lenstra, Lenstra, and Lovász [39]. This is a polynomial time algorithm for SVP (and for most other basic lattice problems) that achieves an approximation factor of $2^{O(n)}$ where n is the dimension of the lattice. As bad as this might seem, the LLL algorithm is surprisingly useful, with applications ranging from factoring polynomials over the rational numbers [39], to integer programming [31], as well as many applications in cryptanalysis (such as attacks on knapsack-based cryptosystems and special cases of RSA).

In 1987, Schnorr presented an extension of the LLL algorithm leading to somewhat better approximation factors [73]. The main idea in Schnorr's algorithm is to replace the core of the LLL algorithm, which involves 2×2 blocks, with blocks of larger size. Increasing the block size improves the approximation factor (i.e., leads to shorter vectors) at the price of an increased running time. Schnorr's algorithm (e.g., as implemented in Shoup's NTL package [75])

is often used by experimenters. Several variants of Schnorr's algorithm exist, such as the recent one by Gama and Nguyen [15] which is quite natural and elegant. Unfortunately, all these variants achieve more or less the same exponential approximation guarantee.

If one insists on an *exact* solution to SVP, or even just an approximation to within $\text{poly}(n)$ factors, the best known algorithm has a running time of $2^{O(n)}$ [6]. The space requirement of this algorithm is unfortunately also exponential which makes it essentially impractical (but see [57] for a recent implementation that can handle dimensions up to 50). Other algorithms require only polynomial space, but run in $2^{O(n \log n)}$ time (see [31] and the references in [57]).

The above discussion leads us to the following conjecture.

Conjecture 1. There is no polynomial time algorithm that approximates lattice problems to within polynomial factors.

Less formally, it is conjectured that approximating lattice problems to within polynomial factors is a hard problem (see also [72]). As we shall see later, the security of many lattice-based cryptographic constructions is based on this conjecture. As a further evidence for this conjecture, we note that progress in lattice algorithms has been stubbornly difficult, with no significant improvement in performance since the 1980s. This is in contrast to number theoretic problems such as factoring for which we have some remarkable subexponential time algorithms like the number field sieve [38]. We should note, though, that approximating lattice problems to within factors above $\sqrt{n/\log n}$ is *not* NP-hard unless the polynomial time hierarchy collapses [2, 20, 37]; NP-hardness results for lattice problems are known only for much smaller approximation factors such as $n^{O(1/\log\log n)}$ (see [3, 12, 14, 25, 33, 47, 77] and the survey [34]).

When applied to "real-life" lattices or lattices chosen randomly from some natural distribution, lattice reduction algorithms tend to perform somewhat better than their worst-case performance. This phenomenon is still not fully explained, but has been observed in many experiments. In one such recent investigation [16], Gama and Nguyen performed extensive experiments with several lattice reduction algorithms and several distributions on lattices. One of their conclusions is that known lattice reduction algorithms provide an approximation ratio of roughly δ^n where n is the dimension of the lattice and δ is a constant that depends on the algorithm. The best δ achievable with algorithms running in reasonable time is very close to 1.012. Moreover, it seems that approximation ratios of $(1.01)^n$ are outside the reach of known lattice reduction algorithm. See Section 3 for a further discussion of the Gama-Nguyen experiments.

1.2 Lattice-based cryptography

As mentioned in the beginning of this chapter, lattice-based cryptographic constructions hold a great promise for post-quantum cryptography. Many of

them are quite efficient, and some even compete with the best known alternatives; they are typically quite simple to implement; and of course, they are all believed to be secure against quantum computers (a topic which we will discuss in more detail in the next subsection).

In terms of security, lattice-based cryptographic constructions can be divided into two types. The first includes practical proposals, which are typically very efficient, but often lack a supporting proof of security. The second type admit strong provable security guarantees based on the worst-case hardness of lattice problems, but only a few of them are sufficiently efficient to be used in practice. We will consider both types in this chapter, with more emphasis on the latter type.

In the rest of this subsection, we elaborate on the strong security guarantees given by constructions of the latter type, namely that of worst-case hardness. What this means is that breaking the cryptographic construction (even with some small non-negligible probability) is provably at least as hard as solving several lattice problems (approximately, within polynomial factors) in the *worst case*. In other words, breaking the cryptographic construction implies an efficient algorithm for solving *any* instance of some underlying lattice problem. In most cases, the underlying problem is that of approximating lattice problems such as SVP to within polynomial factors, which as mentioned above, is conjectured to be a hard problem.

Such a strong security guarantee is one of the distinguishing features of lattice-based cryptography. Virtually all other cryptographic constructions are based on average-case hardness. For instance, breaking a cryptosystem based on factoring might imply the ability to factor *some* numbers chosen *according to a certain distribution*, but not the ability to factor *all* numbers.

The importance of the worst-case security guarantee is twofold. First, it assures us that attacks on the cryptographic construction are likely to be effective only for small choices of parameters and not asymptotically. In other words, it assures us that there are no fundamental flaws in the design of our cryptographic construction. In fact, in some cases, the worst-case security guarantee can even guide us in making design decisions. Second, in principle the worst-case security guarantee can help us in choosing concrete parameters for the cryptosystem, although in practice this leads to what seems like overly conservative estimates, and as we shall see later, one often sets the parameters based on the best known attacks.

1.3 Quantum and lattices

As we have seen above, lattice problems are typically quite hard. The best known algorithms either run in exponential time, or provide quite bad approximation ratios. The field of lattice-based cryptography has been developed based on the assumption that lattice problems are hard. But is lattice-based cryptography suitable for a post-quantum world? Are lattice problems hard even for quantum computers?

The short answer to this is "probably yes": *There are currently no known quantum algorithms for solving lattice problems that perform significantly better than the best known classical (i.e., non-quantum) algorithms* (but see [41]). This is despite the fact that lattice problems seem like a natural candidate to attempt to solve using quantum algorithms: because they are believed not to be NP-hard for typical approximation factors, because of their periodic structure, and because the Fourier transform, which is used so successfully in quantum algorithms, is tightly related to the notion of lattice duality.

Attempts to solve lattice problems by quantum algorithms have been made since Shor's discovery of the quantum factoring algorithm in the mid-1990s, but have so far met with little success if any at all. The main difficulty is that the periodicity finding technique, which is used in Shor's factoring algorithm and related quantum algorithms, does not seem to be applicable to lattice problems. It is therefore natural to consider the following conjecture, which justifies the use of lattice-based cryptography for post-quantum cryptography:

Conjecture 2. There is no polynomial time quantum algorithm that approximates lattice problems to within polynomial factors.

The above discussion, however, should not be interpreted as saying that the advent of quantum algorithms had no influence on our understanding of lattice problems. Although actual quantum algorithms for lattice problems are not known, there are a few very intriguing connections between quantum algorithms and lattice problems. The first such connection was demonstrated in [69] where it was shown that a certain extension of the period finding problem to non-Abelian groups can be used to give quantum algorithms for lattice problems. This approach, unfortunately, has so far not led to any interesting quantum algorithms for lattice problems.

A possibly more interesting connection is the use of a quantum hardness assumption in the lattice-based cryptosystem of [70]. A detailed discussion of this cryptosystem and its applications will appear in Subsection 5.4. For now, we briefly discuss the way quantum algorithms are used there. The main observation made there is that quantum algorithms *can* be useful in solving lattice problems, albeit somewhat unnatural ones. Consider the following scenario. We are given an oracle that is able to answer queries of the following type: on input a lattice $\mathcal{L}$ and a point $\mathbf{x}$ that is somewhat close to $\mathcal{L}$, it outputs the closest lattice point to $\mathbf{x}$. If $\mathbf{x}$ is not close enough to $\mathcal{L}$, the output of the oracle is undefined. In some sense, such an oracle seems quite powerful: the best known algorithms for performing such a task require exponential time. Nevertheless, there seems to be absolutely no way to do anything "useful" with this oracle classically! Indeed, it seems that the only way to generate inputs to the oracle is the following: somehow choose a lattice point $\mathbf{y} \in \mathcal{L}$ and let $\mathbf{x} = \mathbf{y} + \mathbf{z}$ for some small perturbation vector $\mathbf{z}$. We can now feed $\mathbf{x}$ to the oracle since it is close to the lattice. But the result we get, $\mathbf{y}$, is totally useless since we already know it!

It turns out that in the quantum setting, such an oracle is quite useful. Indeed, being able to compute $\mathbf{y}$ from $\mathbf{x}$ allows to *uncompute* $\mathbf{y}$. More precisely, it allows to transform the quantum state $|\mathbf{x}, \mathbf{y}\rangle$ to the state $|\mathbf{x}, 0\rangle$ in a reversible (i.e., unitary) way. This ability to erase the content of a memory cell in a reversible way seems useful only in the quantum setting. By using this together with the Fourier transform, it is shown in [70] how to use such an oracle in order to find short lattice vectors in the dual lattice.

1.4 Organization

The rest of this chapter is organized as follows. In Section 2 we provide some preliminaries on lattices. In Section 3 we consider a certain lattice problem that lies at the heart of many lattice-based cryptographic constructions, and discuss the best known algorithms for solving it. This will be used when we suggest concrete parameters for lattice-based constructions. The next three sections discuss three main cryptographic primitives: hash functions (Section 4), public key cryptosystems (Section 5), and digital signature schemes (Section 6). Some recent constructions of other cryptographic primitives are mentioned in Section 7. Finally, in Section 8 we list some of the main open questions in the area.

2 Preliminaries

All logarithms are base 2 unless otherwise indicated. We use column notation for vectors and use $(x_1, \ldots, x_n)$ to denote the column vector with entries $x_1, \ldots, x_n$. We use square brackets to enclose matrices and row vectors.

Lattices:

A *lattice* is defined as the set of all integer combinations

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \colon x_i \in \mathbb{Z} \text{ for } 1 \leq i \leq n \right\}$$

of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^n$ (see Figure 1). The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is called a *basis* for the lattice. A basis can be represented by the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{n \times n}$ having the basis vectors as columns. Using matrix notation, the lattice generated by a matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ can be defined as $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} \colon \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{Bx}$ is the usual matrix-vector multiplication.

It is not difficult to see that if $\mathbf{U}$ is a unimodular matrix (i.e., an integer square matrix with determinant ± 1), the bases $\mathbf{B}$ and $\mathbf{BU}$ generate the same lattice. (In fact, $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$ if and only if there exists a unimodular matrix

$\mathbf{U}$ such that $\mathbf{B}' = \mathbf{BU}$.) In particular, any lattice admits multiple bases, and this fact is at the core of many cryptographic applications.

The *determinant* of a lattice is the absolute value of the determinant of the basis matrix $\det(\mathcal{L}(\mathbf{B})) = |\det(\mathbf{B})|$. The value of the determinant is independent of the choice of the basis, and geometrically corresponds to the inverse of the density of the lattice points in $\mathbb{R}^n$. The *dual* of a lattice $\mathcal{L}$ in $\mathbb{R}^n$, denoted $\mathcal{L}^*$, is the lattice given by the set of all vectors $\mathbf{y} \in \mathbb{R}^n$ satisfying $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}$ for all vectors $\mathbf{x} \in \mathcal{L}$. It can be seen that for any $\mathbf{B} \in \mathbb{R}^{n \times n}$, $\mathcal{L}(\mathbf{B})^* = \mathcal{L}((\mathbf{B}^{-1})^T)$. From this it follows that $\det(\mathcal{L}^*) = 1/\det(\mathcal{L})$.

q-ary lattices:

Of particular importance in lattice-based cryptography are *q-ary lattices*. These are lattices $\mathcal{L}$ satisfying $q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n$ for some (possibly prime) integer q. In other words, the membership of a vector $\mathbf{x}$ in $\mathcal{L}$ is determined by $\mathbf{x} \bmod q$. Such lattices are in one-to-one correspondence with linear codes in $\mathbb{Z}_q^n$. Most lattice-based cryptographic constructions use q-ary lattices as their hard-on-average problem. We remark that any integer lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is a q-ary lattice for some q, e.g., whenever q is an integer multiple of the determinant $\det(\mathcal{L})$. However, we will be mostly concerned with q-ary lattices with q much smaller than $\det(L)$.

Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for some integers q, m, n, we can define two m-dimensional q-ary lattices,

$$\Lambda_q(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^m \ : \ \mathbf{y} = \mathbf{A}^T\mathbf{s} \bmod q \text{ for some } \mathbf{s} \in \mathbb{Z}^n\}$$
$$\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}^m \ : \ \mathbf{Ay} = \mathbf{0} \bmod q\}.$$

The first q-ary lattice is generated by the rows of $\mathbf{A}$; the second contains all vectors that are orthogonal modulo q to the rows of $\mathbf{A}$. In other words, the first q-ary lattice corresponds to the code generated by the rows of $\mathbf{A}$ whereas the second corresponds to the code whose parity check matrix is $\mathbf{A}$. It follows from the definition that these lattices are dual to each other, up to normalization; namely, $\Lambda_q^\perp(\mathbf{A}) = q \cdot \Lambda_q(\mathbf{A})^*$ and $\Lambda_q(\mathbf{A}) = q \cdot \Lambda_q^\perp(\mathbf{A})^*$.

Lattice problems:

The most well known computational problems on lattices are the following.

- Shortest Vector Problem (SVP): Given a lattice basis $\mathbf{B}$, find the shortest nonzero vector in $\mathcal{L}(\mathbf{B})$.
- Closest Vector Problem (CVP): Given a lattice basis $\mathbf{B}$ and a target vector $\mathbf{t}$ (not necessarily in the lattice), find the lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ closest to $\mathbf{t}$.
- Shortest Independent Vectors Problem (SIVP): Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, find n linearly independent lattice vectors $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_n]$ (where $\mathbf{s}_i \in \mathcal{L}(\mathbf{B})$ for all i) minimizing the quantity $\|\mathbf{S}\| = \max_i \|\mathbf{s}_i\|$.

In lattice-based cryptography, one typically considers the approximation variant of these problems, which are denoted by an additional subscript γ indicating the approximation factor. For instance, in SVP_γ the goal is to find a vector whose norm is at most γ times that of the shortest nonzero vector. Finally, let us mention that all problems can be defined with respect to any norm, but the Euclidean norm $\|\mathbf{x}\| = \sqrt{\sum_i x^2}$ is the most common (see [66]).

3 Finding Short Vectors in Random q-ary Lattices

Consider the following problem. We are given a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for some q, n and $m \geq n$ and we are asked to find a short vector in $\Lambda_q^\perp(\mathbf{A})$. What is the shortest vector that we can hope to find in a reasonable amount of time? Notice that this is equivalent to asking for a short solution to a set of n random equations modulo q in m variables. There are two main methods to find such solutions, which we review in the next paragraphs. Before addressing the algorithmic question, let us try to estimate the length of the shortest nonzero vector. For this, assume q is prime. Then with high probability (assuming m is not too close to n), the rows of $\mathbf{A}$ are linearly independent over $\mathbb{Z}_q$. In such a case, the number of elements of $\mathbb{Z}_q^m$ that belong to $\Lambda_q^\perp(\mathbf{A})$ is exactly q^{m-n} from which it follows that $\det(\Lambda_q^\perp(\mathbf{A})) = q^n$. We can now heuristically estimate $\lambda_1(\Lambda_q^\perp(\mathbf{A}))$ as the smallest radius of a ball whose volume is q^n, i.e.,

$$\lambda_1(\Lambda_q^\perp(\mathbf{A})) \approx q^{n/m} \cdot ((m/2)!)^{1/m} / \sqrt{\pi} \approx q^{n/m} \cdot \sqrt{\frac{m}{2\pi e}}$$

where we used the formula for the volume of a ball in m dimensions. For reasonable values of m (that are not too close to n nor too large) this estimate seems to be very good, as indicated by some of our experiments in low dimensions. The above estimate applies if we are interested in vectors that have small Euclidean length. Similar arguments apply to other norms. For example, we can expect the lattice to contain nonzero vectors with coordinates all bounded in absolute value by

$$\lambda_1^\infty(\Lambda_q^\perp(\mathbf{A})) \approx \frac{q^{n/m} - 1}{2}.$$

Lattice reduction methods.

We now get back to our original algorithmic question: what is the shortest vector that we can hope to find in a reasonable amount of time? In order to answer this question, we rely on the extensive experiments made by Gama and Nguyen in [16]. Although their experiments were performed on a different distribution on lattices, their results seem to apply very well also to the case of random q-ary lattices. Indeed, in all our experiments we observed the same behavior reported in their paper, with the exception that a "trivial" vector of

length q can always be found; namely, the length of the vector obtained by running the best known algorithms on a random m-dimensional q-ary lattice $\Lambda_q^\perp(\mathbf{A})$ is close to

$$\min\{q, (\det(\Lambda_q^\perp(\mathbf{A})))^{1/m} \cdot \delta^m\} = \min\{q, q^{n/m}\delta^m\} \quad (1)$$

where the equality holds with high probability. The parameter δ depends on the algorithm used. Faster algorithms (which are unavoidable when the dimension is several hundreds) provide $\delta \approx 1.013$ whereas slower and more precise algorithms provide $\delta \approx 1.012$ or even $\delta \approx 1.011$. Lower values of δ seem to be impossible to obtain with our current understanding of lattice reduction. Gama and Nguyen in fact estimate that a factor of 1.005 is totally out of reach in dimension 500.

We now try to understand the effect that m has on the hardness of the question. A simple yet important observation to make is that the problem cannot become harder by increasing m. Indeed, we can always fix some of the variables (or coordinates) to 0 thereby effectively reducing to a problem with smaller m. In lattice terminology, this says that $\Lambda_q^\perp(\mathbf{A})$ contains as a "sublattice" $\Lambda_q^\perp(\mathbf{A}')$ where $\mathbf{A}'$ is obtained from $\mathbf{A}$ by removing some of its columns. (More precisely, since the two lattices are of different dimensions, we need to append zero coordinates to the latter in order for it to be a true sublattice of the former.)

In Figure 2 we plot $q^{n/m}\delta^m$ as a function of m. It is easy to see that the minimum of the function is $2^{2\sqrt{n \log q \log \delta}}$ and is obtained for $m = \sqrt{n \log q / \log \delta}$. This means that when applying lattice reduction algorithms to $\Lambda_q^\perp(\mathbf{A})$, the shortest vectors are produced when $m = \sqrt{n \log q / \log \delta}$. For smaller m, the lattice is too sparse and does not contain short enough vectors. For larger m, the high dimension prevents lattice reduction algorithms from finding short vectors. In such a case, one is better off removing some of the columns of $\mathbf{A}$ in order to arrive at a lower dimensional problem. We note that this phenomenon has showed up clearly in our experiments.

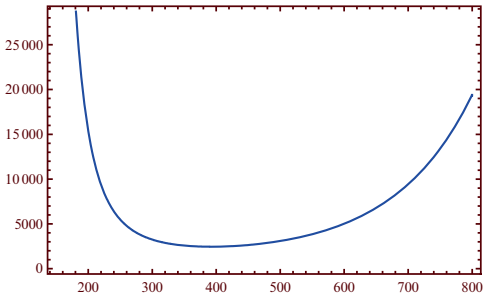

Fig. 2. Estimated length of vector found with $\delta = 1.01$, $q = 4416857$, and $n = 100$ as a function of m.

To summarize, based on the experiments made by Gama and Nguyen, we can conclude that the shortest vector one can find in $\Lambda_q^\perp(\mathbf{A})$ for a random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ using state of the art lattice reduction algorithms is of length at least

$$\min\{q, 2^{2\sqrt{n \log q \log \delta}}\}, \tag{2}$$

where δ is not less than 1.01. Notice that the above expression is independent of m. This indicates that the difficulty of the problem depends mainly on n and q and not so much on m. Interestingly, the parameter m plays a minor role also in Ajtai's worst-case connection, giving further evidence that n and q alone determine the difficulty of the problem.

Combinatorial methods.

It is interesting to consider also combinatorial methods to find short vectors in a q-ary lattice, as for certain choices of parameters these methods perform better than lattice reduction. The best combinatorial methods to find short vectors in q-ary lattices are variants of the algorithms presented [9, 78], e.g., as described in [45] in the context of attacking lattice-based hash functions.

The method works as follows. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, say we want to find a lattice point in $\Lambda_q^\perp(\mathbf{A})$ with coordinates all bounded in absolute value by b. We proceed as follows:

- Divide the columns of $\mathbf{A}$ into 2^k groups (for some k to be determined), each containing $m/2^k$ columns.
- For each group, build a list containing all linear combinations of the columns with coefficients in $\{-b, \ldots, b\}$.
- At this point we have 2^k lists, each containing $L = (2b+1)^{m/2^k}$ vectors in $\mathbb{Z}_q^n$. Combine the lists in pairs. When two lists are combined, take all the sums $\mathbf{x} + \mathbf{y}$ where $\mathbf{x}$ is an element of the first list, $\mathbf{y}$ is an element of the second list, and their sum $\mathbf{x} + \mathbf{y}$ is zero in the first $\log_q L$ coordinates. Since these coordinates can take $q^{\log_q L} = L$ values, we can expect the list resulting from the combination process to have size approximately equal to $L \cdot L/L = L$.
- At this point we have 2^{k-1} lists of size L containing vectors that are zero in their first $\log_q L$ coordinates. Keep combining the lists in pairs, until after k iterations we are left with only one list of size L containing vectors that are 0 in their first $k \cdot \log_q L$ coordinates. The parameter k is chosen in such a way that $n \approx (k+1) \log_q L$, or equivalently,

$$\frac{2^k}{k+1} \approx \frac{m \log(2b+1)}{n \log q}. \tag{3}$$

  For such a value of k, the vectors in the last list are zero in all but their last $n - k \log_q L \approx \log_q L$ coordinates. So, we can expect the list to contain the all zero vector.

The all zero vector found in the last list is given by a combination of the columns of $\mathbf{A}$ with coefficients bounded by b, so we have found the desired short lattice vector. Differently from lattice reduction, we can always expect this attack to succeed when $\mathbf{A}$ is random. The question is: what is the cost of running the attack? It is easy to see that the cost of the attack is dominated by the size of the lists L, which equals $(2b+1)^{m/2^k}$, where k is the largest integer satisfying (3). In certain settings (e.g., the construction of lattice-based hash functions presented in Section 4) lattice-based attacks stop finding short enough vectors well before the combinatorial attack becomes infeasible. So, the combinatorial attack can be used to determine the value of the parameters necessary to achieve a certain level of security.

Another difference between the combinatorial attack and those based on lattice reduction is that the combinatorial attack does take advantage of the large value of m. Larger values of m allow to use larger values for k, yielding shorter lists and more efficient attacks.

4 Hash Functions

A collision resistant hash function is a function $h : D \to R$ mapping a domain D to a much smaller set R, $|R| \ll |D|$ such that it is computationally hard to find collisions, i.e., input pairs $x_1, x_2 \in D$ such that $x_1 \neq x_2$ and still $h(x_1) = h(x_2)$. Technically, hash functions are often defined as keyed function families, where a collection of functions $\{h_k \colon D \to R\}$ is specified, and the security property is that given a randomly chosen k, no attacker can efficiently find a collision in h_k, even though such collisions certainly exist because D is larger than R. Collision resistant hash functions are very useful cryptographic primitives because they allow to compress a long message $x \in D$ to a short digest $h(x) \in R$, and still the digest is (computationally) bound to a unique x because of the collision resistance property.

For efficiency reasons, hash functions currently used in practice are based on ad-hoc design principles, similar to those used in the construction of block ciphers. Such functions, however, have been subject to attacks, raising interest in more theoretical constructions that can be proved secure based on some underlying mathematical problem. Collision resistant hash functions can be built starting from standard number theoretic problems (like the hardness of factoring integers, or the RSA problem), similar to those used in public key cryptography, but such constructions are unsatisfactory for two reasons: they are much slower than block ciphers, and they can be broken by quantum computers.

In this section we present various constructions of collision resistant hash functions based on lattices, starting from Ajtai's original work, and ending with SWIFFT, a highly efficient recent proposal based on a special class of lattices. These have several benefits over competing constructions: they admit supporting proofs of security (based on worst-case complexity assumptions),

they appear to be resistant to quantum attacks, and the most efficient of them approaches efficiency levels comparable to those of traditional block cipher design. Finally, many techniques used in other lattice-based cryptographic constructions have been first developed in the context of collision resistant hashing. So, hash functions offer an excellent starting point to discuss the methods of lattice-based cryptography at large.

4.1 Ajtai's construction and further improvements

The first lattice-based cryptographic construction with worst-case security guarantees was presented in the seminal work of Ajtai [7]. Ajtai presented a family of one-way functions whose security is based on the worst-case hardness of n^c-approximate SVP for some constant $c > 0$. In other words, he showed that being able to invert a function chosen from this family with non-negligible probability implies the ability to solve *any* instance of n^c-approximate SVP.

Followup work concentrated on improving Ajtai's security proof. Goldreich et al. [21] showed that Ajtai's function is collision resistant, a stronger (and much more useful) security property than one-wayness. Most of the subsequent work focused on reducing the value of the constant c [11, 48, 54], thereby improving the security assumption. In the most recent work, the constant is essentially $c = 1$ [54]. We remark that all these constructions are based on the worst-case hardness of a problem not believed to be NP-hard (since $c \geq \frac{1}{2}$).

The main statement in all the above results is that for an appropriate choice of q, n, m, finding short vectors in $\Lambda_q^\perp(\mathbf{A})$ when $\mathbf{A}$ is chosen uniformly at random from $\mathbb{Z}_q^{n \times m}$ is as hard as solving certain lattice problems (such as approximate SIVP and approximate SVP) in the *worst case*. This holds even if the algorithm is successful in finding short vectors only with an inverse polynomially small probability (over the choice of matrix $\mathbf{A}$ and its internal randomness).

Once such a reduction is established, constructing a family of collision resistant hash functions is easy (see Algorithm 4.1). The hash function is parameterized by integers n, m, q, d. A possible choice is $d = 2$, $q = n^2$, and $m > n \log q / \log d$. The choice of n then determines the security of the hash function. The key to the hash function is given by a matrix $\mathbf{A}$ chosen uniformly from $\mathbb{Z}_q^{n \times m}$. The hash function $f_{\mathbf{A}} : \{0, \ldots, d-1\}^m \to \mathbb{Z}_q^n$ is given by $f_{\mathbf{A}}(\mathbf{y}) = \mathbf{A}\mathbf{y} \bmod q$. In terms of bits, the function maps $m \log d$ bits into $n \log q$ bits, hence we should choose $m > n \log q / \log d$ in order to obtain a hash function that compresses the input, or more typically $m \approx 2n \log q / \log d$ to achieve compression by a factor 2.

Notice that a collision $f_{\mathbf{A}}(\mathbf{y}) = f_{\mathbf{A}}(\mathbf{y}')$ for some $\mathbf{y} \neq \mathbf{y}'$ immediately yields a short non-zero vector $\mathbf{y} - \mathbf{y}' \in \Lambda_q^\perp(\mathbf{A})$. Using a worst-case to average-case reduction as above, we obtain that finding collisions for function $f_{\mathbf{A}}$ (even with an inverse polynomially small probability), is as hard as solving approximate SIVP and approximate SVP in the worst case.

Algorithm 4.1 A hash function following Ajtai's construction.

- **Parameters:** Integers $n, m, q, d \geq 1$.
- **Key:** A matrix $\mathbf{A}$ chosen uniformly from $\mathbb{Z}_q^{n \times m}$.
- **Hash function:** $f_{\mathbf{A}} : \{0, \ldots, d-1\}^m \to \mathbb{Z}_q^n$ given by $f_{\mathbf{A}}(\mathbf{y}) = \mathbf{A}\mathbf{y} \bmod q$.

It is worth noting that this hash function is extremely simple to implement as it involves nothing but addition and multiplication modulo q, and q is a $O(\log n)$ bit number which comfortably fits into a single memory word or processor register. So, all arithmetic can be performed very efficiently without the need of the arbitrary precision integers commonly used in number theoretic cryptographic functions. As we shall see later, this is typical of lattice-based cryptography. Further optimizations can be obtained by choosing q to be a power of 2, and $d = 2$ which allows to represent the input as a sequence of m bits as well as to avoid the need for multiplications. Nevertheless, these hash functions are not particularly efficient because the key size grows at least quadratically in n. Consider for example setting $d = 2$, $q = n^2$, and $m = 2n \log q = 4n \log n$. The corresponding function has a key containing $mn = 4n^2 \log n$ elements of $\mathbb{Z}_q$, and its evaluation requires roughly as many arithmetic operations. Collisions are given by vectors in $\Lambda_q^{\perp}(\mathbf{A})$ with entries in $\{1, 0, -1\}$. The combinatorial method described in Section 3 with bound $b = 1$ and parameter $k = 4$, yields an attack with complexity $L = 3^{m/16} \approx 2^{m/10}$. So, in order to get 100 bits of security ($L \approx 2^{100}$), one needs to set $m = 4n \log n \approx 1000$, and $n \geq 46$. This yields a hash function with a key size of $mn \log q \approx 500{,}000$ bits, and computation time of the order of $mn \approx 50{,}000$ arithmetic operations. Although still reasonable for a public key encryption function, this is considered unacceptable in practice for simpler cryptographic primitives like symmetric block ciphers or collision resistant hash functions.

4.2 Efficient hash functions based on cyclic and ideal lattices

The efficiency of lattice-based cryptographic functions can be substantially improved replacing general matrices by matrices with special structure. For example, in Algorithm 4.1, the random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ can be replaced by a block-matrix

$$\mathbf{A} = [\mathbf{A}^{(1)} \mid \ldots \mid \mathbf{A}^{(m/n)}] \tag{4}$$

where each block $\mathbf{A}^{(i)} \in \mathbb{Z}_q^{n \times n}$ is a circulant matrix

$$\mathbf{A}^{(i)} = \begin{bmatrix} a_1^{(i)} & a_n^{(i)} & \cdots & a_3^{(i)} & a_2^{(i)} \\ a_2^{(i)} & a_1^{(i)} & \cdots & a_4^{(i)} & a_3^{(i)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1}^{(i)} & a_{n-2}^{(i)} & \cdots & a_1^{(i)} & a_n^{(i)} \\ a_n^{(i)} & a_{n-1}^{(i)} & \cdots & a_2^{(i)} & a_1^{(i)} \end{bmatrix},$$

i.e., a matrix whose columns are all cyclic rotations of the first column $\mathbf{a}^{(i)} = (a_1^{(i)}, \ldots, a_n^{(i)})$. Using matrix notation, $\mathbf{A}^{(i)} = [\mathbf{a}^{(i)}, \mathbf{T}\mathbf{a}^{(i)}, \ldots, \mathbf{T}^{n-1}\mathbf{a}^{(i)}]$ where

$$\mathbf{T} = \left[\begin{array}{ccc|c} & \mathbf{0}^T & & 1 \\ \hline \ddots & & & \\ & \mathbf{I} & & \mathbf{0} \\ & & \ddots & \end{array}\right], \tag{5}$$

is the permutation matrix that rotates the coordinates of $\mathbf{a}^{(i)}$ cyclically. The circulant structure of the blocks has two immediate consequences:

- It reduces the key storage requirement from nm elements of $\mathbb{Z}_q$ to just m elements, because each block $\mathbf{A}^{(i)}$ is fully specified by its first column $\mathbf{a}^{(i)} = (a_1^{(i)}, \ldots, a_n^{(i)})$.
- It also reduces (at least asymptotically) the running time required to compute the matrix-vector product $\mathbf{A}\mathbf{y} \bmod q$, from $O(mn)$ arithmetic operations (over $\mathbb{Z}_q$), to just $\tilde{O}(m)$ operations, because multiplication by a circulant matrix can be implemented in $\tilde{O}(n)$ time using the Fast Fourier Transform.

Of course, imposing any structure on matrix $\mathbf{A}$, immediately invalidates the proofs of security [7,11,48,54] showing that finding collisions on the average is at least as hard as approximating lattice problems in the worst case. A fundamental question that needs to be addressed whenever a theoretical construction is modified for the sake of efficiency, is if the modification introduces security weaknesses.

The use of circulant matrices in lattice-based cryptography can be traced back to the NTRU cryptosystem [29], which is described in Section 5. However, till recently no theoretical results were known supporting the use of structured matrices in lattice-based cryptography. Several years after Ajtai's worst-case connection for general lattices [7] and the proposal of the NTRU cryptosystem [29], Micciancio [53] discovered that the efficient one-way function obtained by imposing a circulant structure on the blocks of (4) can still be proved to be hard to invert on the average based on the worst-case hardness of approximating SVP, albeit only over a restricted class of lattices which are invariant under cyclic rotation of the coordinates. Interestingly, no better algorithms (than those for general lattices) are known to solve lattice problems for such cyclic lattices. So, it is reasonable to assume that solving lattice problems on these lattices is as hard as the general case.

Micciancio's adaptation [53] of Ajtai's worst-case connection to cyclic lattices is non-trivial. In particular, Micciancio could only prove that the resulting function is one-way (i.e., hard to invert), as opposed to collision resistant. In fact, collisions can be efficiently found: in [42, 61] it was observed that if each block $\mathbf{A}^{(i)}$ is multiplied by a constant vector $c_i \cdot \mathbf{1} = (c_i, \ldots, c_i)$, then the output of $f_{\mathbf{A}}$ is going to be a constant vector $c \cdot \mathbf{1}$ too. Since c can take only

q different values, a collision can be found in time q (or even $O(\sqrt{q})$, probabilistically), which is typically polynomial in n. Similar methods were later used in [45] to find collisions in the compression function of LASH, a practical hash function proposal modeled after the NTRU cryptosystem. The existence of collisions for these functions demonstrates the importance of theoretical security proofs whenever a cryptographic construction is modified.

While one-way functions are not strong enough security primitives to be directly useful in applications, the results of [53] stimulated theoretical interest in the construction of efficient cryptographic functions based on structured lattices, leading to the use of cyclic (and other similarly structured) lattices in the design of many other more useful primitives [42–44, 61], as well as further investigation of lattices with algebraic structure [62]. In the rest of this section, we describe the collision resistant hash functions of [42, 61], and their most recent practical instantiation [45]. Other cryptographic primitives based on structured lattices are described in Sections 5, 6, and 7.

Collision resistance from ideal lattices

The problem of turning the efficient one-way function of [53] into a collision resistant function was independently solved by Peikert and Rosen [61], and Lyubashevsky and Micciancio [42] using different (but related) methods. Here we follow the approach used in the latter work, which also generalizes the construction of [53,61] based on circulant matrices, to a wider range of structured matrices, some of which admit very efficient implementations [45]. The general construction, shown in Algorithm 4.2, is parametrized by integers n, m, q, d and a vector $\mathbf{f} \in \mathbb{Z}^n$, and it can be regarded as a special case of Algorithm 4.1 with structured keys $\mathbf{A}$. In Algorithm 4.2, instead of choosing $\mathbf{A}$ at random from the set of *all* matrices, one sets $\mathbf{A}$ to a block-matrix as in Eq. (4) with structured blocks $\mathbf{A}^{(i)} = \mathbf{F}^*\mathbf{a}^{(i)}$ defined as

$$\mathbf{F}^*\mathbf{a}^{(i)} = [\mathbf{a}^{(i)}, \mathbf{F}\mathbf{a}^{(i)}, \ldots, \mathbf{F}^{n-1}\mathbf{a}^{(i)}] \qquad \text{where} \quad \mathbf{F} = \left[\begin{array}{c|c} \mathbf{0}^T & \\ \hline \ddots & \\ \mathbf{I} & -\mathbf{f} \\ & \ddots \end{array}\right].$$

The circulant matrices discussed earlier are obtained as a special case by setting $\mathbf{f} = (-1, 0, \ldots, 0)$, for which $\mathbf{F} = \mathbf{T}$ is just a cyclic rotation of the coordinates. The complexity assumption underlying the function is that lattice problems are hard to approximate in the worst case over the class of lattices that are invariant under transformation $\mathbf{F}$ (over the integers). When $\mathbf{f} = (-1, 0, \ldots, 0)$, this is exactly the class of cyclic lattices, i.e., lattices that are invariant under cyclic rotation of the coordinates. For general $\mathbf{f}$, the corresponding lattices have been named *ideal* lattices in [42], because they can be equivalently characterized as *ideals* of the ring of modular polynomials

$\mathbb{Z}[x]/\langle f(x)\rangle$ where $f(x) = x^n + f_n x^{n-1} + \cdots + f_1 \in \mathbb{Z}[x]$. As for the class of cyclic lattices, no algorithm is known that solves lattice problems on ideal lattices any better than on general lattices. So, it is reasonable to assume that solving lattice problems on ideal lattices is as hard as the general case.

Algorithm 4.2 Hash function based on ideal lattices.

- **Parameters:** Integers q, n, m, d with $n|m$, and vector $\mathbf{f} \in \mathbb{Z}^n$.
- **Key:** m/n vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{m/n}$ chosen independently and uniformly at random in $\mathbb{Z}_q^n$.
- **Hash function:** $f_{\mathbf{A}} : \{0, \ldots, d-1\}^m \to \mathbb{Z}_q^n$ given by

$$f_{\mathbf{A}}(\mathbf{y}) = [\mathbf{F}^*\mathbf{a}_1 \mid \ldots \mid \mathbf{F}^*\mathbf{a}_{m/n}]\mathbf{y} \bmod q.$$

Even for arbitrary $\mathbf{f}$, the construction described in Algorithm 4.2 still enjoys the efficiency properties of the one-way function of [53]: keys are represented by just m elements of $\mathbb{Z}_q$, and the function can be evaluated with $\tilde{O}(m)$ arithmetic operations using the Fast Fourier Transform (over the complex numbers). As usual, collisions are short vectors in the lattice $\Lambda_q^\perp([\mathbf{F}^*\mathbf{a}_1 \mid \ldots \mid \mathbf{F}^*\mathbf{a}_{m/n}])$. But, are short vectors in these lattices hard to find? We have already seen that in general the answer to this question is no: when $\mathbf{f} = (-1, 0, \ldots, 0)$ short vectors (and collisions in the hash function) can be easily found in time $O(q)$. Interestingly, [42] proves that finding short vectors in $\Lambda_q^\perp([\mathbf{F}^*\mathbf{a}_1 \mid \ldots \mid \mathbf{F}^*\mathbf{a}_{m/n}])$ on the average (even with just inverse polynomial probability) is as hard as solving various lattice problems (such as approximate SVP and SIVP) in the worst case over ideal lattices, provided the vector $\mathbf{f}$ satisfies the following two properties:

- For any two unit vectors $\mathbf{u}, \mathbf{v}$, the vector $[\mathbf{F}^*\mathbf{u}]\mathbf{v}$ has small (say, polynomial in n, typically $O(\sqrt{n})$) norm.
- The polynomial $f(x) = x^n + f_n x^{n-1} + \cdots + f_1 \in \mathbb{Z}[x]$ is irreducible over the integers, i.e., it does not factor into the product of integer polynomials of smaller degree.

Notice that the first property is satisfied by the vector $\mathbf{f} = (-1, 0, \ldots, 0)$ corresponding to circulant matrices, because all the coordinates of $[\mathbf{F}^*\mathbf{u}]\mathbf{v}$ are bounded by 1, and hence $\|[\mathbf{F}^*\mathbf{u}]\mathbf{v}\| \leq \sqrt{n}$. However, the polynomial $x^n - 1$ corresponding to $\mathbf{f} = (-1, 0, \ldots, 0)$ is not irreducible because it factors into $(x-1)(x^{n-1} + x^{n-2} + \cdots + x + 1)$, and this is why collisions can be efficiently found. So, $\mathbf{f} = (-1, 0, \ldots, 0)$ is not a good choice to get collision resistant hash functions, but many other choices are possible. For example, some choices of $\mathbf{f}$ considered in [42] for which both properties are satisfied (and therefore, result in collision resistant hash functions with worst-case security guarantees) are

- $\mathbf{f} = (1, \ldots, 1) \in \mathbb{Z}^n$ where $n+1$ is prime, and
- $\mathbf{f} = (1, 0, \ldots, 0) \in \mathbb{Z}^n$ for n equal to a power of 2.

The latter choice turns out to be very convenient from an implementation point of view, as described in the next subsection. Notice how ideal lattices associated to vector $(1, 0, \ldots, 0)$ are very similar to cyclic lattices: the transformation $\mathbf{F}$ is just a cyclic rotation of the coordinates, with the sign of the coordinate wrapping around changed, and the blocks of $\mathbf{A}$ are just circulant matrices, but with the elements above the diagonal negated. This small change in the structure of matrix $\mathbf{A}$ has dramatic effects on the collision resistance properties of the resulting hash function: If the signs of the elements above the diagonals of the blocks is not changed, then collisions in the hash function can be easily found. Changing the sign results in hash functions for which finding collisions is provably as hard as the worst-case complexity of lattice approximation problems over ideal lattices.

The SWIFFT hash function

The hash function described in the previous section is quite efficient and can be computed asymptotically in $\tilde{O}(m)$ time using the Fast Fourier Transform over the complex numbers. However, in practice, this carries a substantial overhead. In this subsection we describe the SWIFFT family of hash functions proposed in [45]. This is essentially a highly optimized variant of the hash function described in the previous section, and is highly efficient *in practice*, mainly due to the use of the FFT in $\mathbb{Z}_q$.

We now proceed to describe the SWIFFT hash function. As already suggested earlier, the vector $\mathbf{f}$ is set to $(1, 0, \ldots, 0) \in \mathbb{Z}^n$ for n equal to a power of 2, so that the corresponding polynomial $x^n + 1$ is irreducible. The novelty in [45] is a clever choice of the modulus q and a pre/post-processing operation applied to the key and the output of the hash function. More specifically, let q be a prime number such that $2n$ divides $q-1$, and let $\mathbf{W} \in \mathbb{Z}_q^{n \times n}$ be an invertible matrix over $\mathbb{Z}_q$ to be chosen later. The SWIFFT hash function maps a key $\tilde{\mathbf{a}}^{(1)}, \ldots, \tilde{\mathbf{a}}^{(m/n)}$ consisting of m/n vectors chosen uniformly from $\mathbb{Z}_q^n$ and an input $\mathbf{y} \in \{0, \ldots, d-1\}^m$ to $\mathbf{W} \cdot f_{\mathbf{A}}(\mathbf{y}) \bmod q$ where $\mathbf{A} = [\mathbf{F}^* \mathbf{a}^{(1)}, \ldots, \mathbf{F}^* \mathbf{a}^{(m/n)}]$ is as before and $\mathbf{a}^{(i)} = \mathbf{W}^{-1} \tilde{\mathbf{a}}^{(i)} \bmod q$. As we shall see later, SWIFFT can be computed very efficiently (even though at this point its definition looks more complicated than that of $f_{\mathbf{A}}$).

Notice that multiplication by the invertible matrix $\mathbf{W}^{-1}$ maps a uniformly chosen $\tilde{\mathbf{a}} \in \mathbb{Z}_q^n$ to a uniformly chosen $\mathbf{a} \in \mathbb{Z}_q^n$. Moreover, $\mathbf{W} \cdot f_{\mathbf{A}}(\mathbf{y}) = \mathbf{W} \cdot f_{\mathbf{A}}(\mathbf{y}')$ (mod q) if and only if $f_{\mathbf{A}}(\mathbf{y}) = f_{\mathbf{A}}(\mathbf{y}')$ (mod q). Together, these two facts establish that finding collisions in SWIFFT is equivalent to finding collisions in the underlying ideal lattice function $f_{\mathbf{A}}$, and the claimed collision resistance property of SWIFFT is supported by the connection [42] to worst case lattice problems on ideal lattices.

We now explain the efficient implementation of SWIFFT given in Algorithm 4.3. By our choice of q, the multiplicative group $\mathbb{Z}_q^*$ of the integers modulo q has an element ω of order $2n$. Let

Algorithm 4.3 The SWIFFT hash function.

- **Parameters:** Integers n, m, q, d such that n is a power of 2, q is prime, $2n|(q-1)$ and $n|m$.
- **Key:** m/n vectors $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_{m/n}$ chosen independently and uniformly at random in $\mathbb{Z}_q^n$.
- **Input:** m/n vectors $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(m/n)} \in \{0, \ldots, d-1\}^n$.
- **Output:** the vector $\sum_{i=1}^{m/n} \tilde{\mathbf{a}}^{(i)} \odot (\mathbf{W}\mathbf{y}^{(i)}) \in \mathbb{Z}_q^n$, where $\odot$ is the component-wise vector product.

$$\mathbf{W} = [\omega^{(2i-1)(j-1)}]_{i=1,j=1}^{n,n}$$

be the Vandermonde matrix of $\omega, \omega^3, \omega^5, \ldots, \omega^{2n-1}$. Since ω has order $2n$, the elements $\omega, \omega^3, \omega^5, \ldots, \omega^{2n-1}$ are distinct, and hence the matrix $\mathbf{W}$ is invertible over $\mathbb{Z}_q$ as required. Moreover, it is not difficult to see that for any vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^n$, the identity

$$\mathbf{W}([\mathbf{F}^*\mathbf{a}]\mathbf{b}) = (\mathbf{W}\mathbf{a}) \odot (\mathbf{W}\mathbf{b}) \bmod q$$

holds true, where $\odot$ is the component-wise vector product. This implies that Algorithm 4.3 correctly computes

$$\mathbf{W} \cdot f_{\mathbf{A}}(\mathbf{y}) = \sum_{i=1}^{m/n} \mathbf{W}[\mathbf{F}^*\mathbf{a}^{(i)}]\mathbf{y}^{(i)} = \sum_{i=1}^{m/n} \tilde{\mathbf{a}}^{(i)} \odot (\mathbf{W}\mathbf{y}^{(i)}).$$

The most expensive part of the algorithm is the computation of the matrix-vector products $\mathbf{W}\mathbf{y}^{(i)}$. These can be efficiently computed using the FFT over $\mathbb{Z}_q$ as follows. Remember that the FFT algorithm over a field $\mathbb{Z}_q$ with an nth root of unity ζ (where n is a power of 2) allows to evaluate any polynomial $p(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} \in \mathbb{Z}_q[x]$ at *all* nth roots of unity ζ^i (for $i = 0, \ldots, n-1$) with just $O(n \log n)$ arithmetic operations in $\mathbb{Z}_q$. Using matrix notation and $\zeta = \omega^2$, the FFT algorithm computes the product $\mathbf{V}\mathbf{c}$ where $\mathbf{V} = [\omega^{2(i-1)(j-1)}]_{i,j}$ is the Vandermonde matrix of the roots $\omega^0, \omega^2, \ldots, \omega^{2(n-1)}$, and $\mathbf{c} = (c_0, \ldots, c_{n-1})$. Going back to the SWIFFT algorithm, the matrix $\mathbf{W}$ can be factored as the product $\mathbf{W} = \mathbf{V}\mathbf{D}$ of $\mathbf{V}$ by the diagonal matrix $\mathbf{D}$ with entries $d_{j,j} = \omega^{j-1}$. So, the product $\mathbf{W}\mathbf{y}^{(i)} = \mathbf{V}\mathbf{D}\mathbf{y}^{(i)}$ can be efficiently evaluated by first computing $\mathbf{D}\mathbf{y}^{(i)}$ (i.e., multiplying the elements of $\mathbf{y}^{(i)}$ component-wise by the diagonal of $\mathbf{D}$), and then applying the FFT algorithm over $\mathbb{Z}_q$ to $\mathbf{D}\mathbf{y}^{(i)}$ to obtain $\mathbf{W}\mathbf{y}^{(i)}$.

Several other implementation-level optimizations are possible, including the use of look-up tables and SIMD (single instruction multiple data) operations in the FFT computation. An optimized implementation of SWIFFT for the choice of parameters given in Table 1 is given in [45], which achieves throughput comparable to the SHA-2 family of hash functions.

Choice of parameters and security.

The authors of [45] propose the set of parameters shown in Table 1. It is easy

n	m	q	d	ω	key size (bits)	input size (bits)	output size (bits)
64	1024	257	2	42	8192	1024	513

Table 1. Concrete parameters for the SWIFFT hash function achieving 100 bits of security.

to verify that $q = 257$ is a prime, $2n = 128$ divides $q-1 = 256$, $n = 64$ divides $m = 1024$, $\omega = 42$ has order $2n = 128$ in $\mathbb{Z}_q^n$, and the resulting hash function $f_{\mathbf{A}} : \{0,1\}^m \to \mathbb{Z}_q^n$ has compression ratio approximately equal to 2, mapping $m = 1024$ input bits to one of $q^n = (2^8+1)^{64} < 2^{513}$ possible outputs. An issue to be addressed is how to represent the vector in $\mathbb{Z}_q^n$ output by SWIFFT as a sequence of bits. The easiest solution is to represent each element of $\mathbb{Z}_q$ as a sequence of 9 bits, so that the resulting output has $9 \cdot 64 = 576$ bits. It is also easy to reduce the output size closer to 513 bits at very little cost. (See [45] for details.)

We now analyze the security of SWIFFT with respect to combinatorial and lattice-based attacks. The combinatorial method described in Section 3 with bound $b = 1$ and parameter $k = 4$ set to the largest integer satisfying (3), yields an attack with complexity $L = 3^{m/16} \geq 2^{100}$.

Let us check that lattice-based attacks are also not likely to be effective in finding collisions. Collisions in SWIFFT are vectors in the m-dimensional lattice $\Lambda_q^{\perp}([\mathbf{F}^*\mathbf{a}_1 \mid \ldots \mid \mathbf{F}^*\mathbf{a}_{m/n}])$ with coordinates in $\{1, 0, -1\}$. Such vectors have Euclidean length at most $\sqrt{m} = 32$. However, according to estimate (2) for $\delta = 1.01$, state of the art lattice reduction algorithms will not be able to find nontrivial lattice vectors of Euclidean length bounded by

$$2^{2\sqrt{n \log q \log \delta}} \approx 42.$$

So, lattice reduction algorithms are unlikely to find collisions. In order to find lattice vectors with Euclidean length bounded by 32, one would need lattice reduction algorithms achieving $\delta < 1.0085$, which seems out of reach with current techniques, and even such algorithms would find vectors with short Euclidean length, but coordinates not necessarily in $\{1, 0, -1\}$.

5 Public Key Encryption Schemes

Several methods have been proposed to build public key encryption schemes based on the hardness of lattice problems. Some are mostly of theoretical interest, as they are still too inefficient to be used in practice, but admit strong provable security guarantees similar to those discussed in Section 4 for hash functions: breaking the encryption scheme (on the average, when the key is chosen at random) can be shown to be at least as hard as solving several lattice problems (approximately, within polynomial factors) in the

worst case. Other schemes are practical proposals, much more efficient than the theoretical constructions, but often lacking a supporting proof of security.

In this section we describe the main lattice-based public key encryption schemes that have been proposed so far. We start from the GGH cryptosystem, which is perhaps the most intuitive encryption scheme based on lattices. We remark that the GGH cryptosystem has been subject to cryptanalytic attacks [58] even for moderately large values of the security parameter, and should be considered insecure from a practical point of view. Still, many of the elements of GGH and its HNF variant [50], can be found in other lattice-based encryption schemes. So, due to its simplicity, the GGH/HNF cryptosystem still offers a good starting point for the discussion of lattice-based public key encryption. Next, we describe the NTRU cryptosystem, which is the most practical lattice-based encryption scheme known to date. Unfortunately, neither GGH nor NTRU is supported by a proof of security showing that breaking the cryptosystem is at least as hard as solving some underlying lattice problem; they are primarily practical proposals aimed at offering a concrete alternative to RSA or other number theoretic cryptosystems.

The rest of this section is dedicated to theoretical constructions of cryptosystems that can be proved to be as hard to break as solving certain lattice problems in the worst case. We briefly review the Ajtai-Dwork cryptosystem (which was the first of its kind admitting a proof of security based on worst-case hardness assumptions on lattice problems) and followup work, and then give a detailed account of a cryptosystem of Regev based on a certain learning problem (called "learning with errors", LWE) that can be related to worst-case lattice assumptions via a quantum reduction. This last cryptosystem is currently the most efficient construction admitting a known theoretical proof of security. While still not as efficient as NTRU, it is the first theoretical construction approaching performance levels that are reasonable enough to be used in practice. Moreover, due to its algebraic features, the LWE cryptosystem has been recently used as the starting point for the construction of various other cryptographic primitives, as discussed in Section 7.

We remark that all cryptosystems described in this section are aimed at achieving the basic security notion called *semantic security* or *indistinguishability under chosen plaintext attack* [23]. This is a strong security notion, but only against passive adversaries that can intercept and observe (but not alter) ciphertexts being transmitted. Informally, semantic security means that an adversary that observes the ciphertexts being sent, cannot extract any (even partial) information about the underlying plaintexts (not even determining whether two given ciphertexts encrypt the same message) under any message distribution. Encryption schemes with stronger security guarantees (against active adversaries) are discussed in Section 7.

5.1 The GGH/HNF public key cryptosystem

The GGH cryptosystem, proposed by Goldreich, Goldwasser, and Halevi in [19], is essentially a lattice analogue of the McEliece cryptosystem [46] proposed 20 years earlier based on the hardness of decoding linear codes over finite fields. The basic idea is very simple and appealing. At a high level, the GGH cryptosystem works as follows:

- The private key is a "good" lattice basis $\mathbf{B}$. Typically, a good basis is a basis consisting of short, almost orthogonal vectors. Algorithmically, good bases allow to efficiently solve certain instances of the closest vector problem in $\mathcal{L}(\mathbf{B})$, e.g., instances where the target is very close to the lattice.
- The public key $\mathbf{H}$ is a "bad" basis for the same lattice $\mathcal{L}(\mathbf{H}) = \mathcal{L}(\mathbf{B})$. In [50], Micciancio proposed to use, as the public basis, the Hermite Normal Form (HNF) of $\mathbf{B}$. This normal form gives a lower[1] triangular basis for $\mathcal{L}(\mathbf{B})$ which is essentially unique, and can be efficiently computed from *any* basis of $\mathcal{L}(\mathbf{B})$ using an integer variant of the Gaussian elimination algorithm.[2] Notice that any attack on the HNF public key can be easily adapted to work with any other basis $\mathbf{B}'$ of $\mathcal{L}(\mathbf{B})$ by first computing $\mathbf{H}$ from $\mathbf{B}'$. So, in a sense, $\mathbf{H}$ is the worst possible basis for $\mathcal{L}(\mathbf{B})$ (from a cryptanalyst's point of view), and makes a good choice as a public basis.
- The encryption process consists of adding a short noise vector $\mathbf{r}$ (somehow encoding the message to be encrypted) to a properly chosen lattice point $\mathbf{v}$. In [50] it is proposed to select the vector $\mathbf{v}$ such that all the coordinates of $(\mathbf{r} + \mathbf{v})$ are reduced modulo the corresponding element along the diagonal of the HNF public basis $\mathbf{H}$. The vector $(\mathbf{r}+\mathbf{v})$ resulting from such a process is denoted $\mathbf{r} \bmod \mathbf{H}$, and it provably makes cryptanalysis hardest because $\mathbf{r} \bmod \mathbf{H}$ can be efficiently computed from any vector of the form $(\mathbf{r} + \mathbf{v})$ with $\mathbf{v} \in \mathcal{L}(\mathbf{B})$. So, any attack on $\mathbf{r} \bmod \mathbf{H}$ can be easily adapted to work on any vector of the form $\mathbf{r} + \mathbf{v}$ by first computing $(\mathbf{r} + \mathbf{v}) \bmod \mathbf{H} = \mathbf{r} \bmod \mathbf{H}$. Notice that $\mathbf{r} \bmod \mathbf{H}$ can be computed directly from $\mathbf{r}$ and $\mathbf{H}$ (without explicitly computing $\mathbf{v}$) by iteratively subtracting multiples of the columns of $\mathbf{H}$ from $\mathbf{r}$. Column $\mathbf{h}_i$ is used to reduce the ith element of $\mathbf{r}$ modulo $h_{i,i}$.
- The decryption problem corresponds to finding the lattice point $\mathbf{v}$ closest to the target ciphertext $\mathbf{c} = (\mathbf{r} \bmod \mathbf{H}) = \mathbf{v} + \mathbf{r}$, and the associated error vector $\mathbf{r} = \mathbf{c} - \mathbf{v}$.

The correctness of the GGH/HNF cryptosystem rests on the fact that the error vector $\mathbf{r}$ is short enough so that the lattice point $\mathbf{v}$ can be recovered from the ciphertext $\mathbf{v} + \mathbf{r}$ using the private basis $\mathbf{B}$, e.g., by using Babai's rounding procedure [8], which gives

[1] The HNF can be equivalently defined using upper triangular matrices. The choice between the lower or upper triangular formulation is pretty much arbitrary.

[2] Some care is required to prevent the matrix entries from becoming too big during intermediate steps of the computation.

$$\mathbf{v} = \mathbf{B}\lfloor \mathbf{B}^{-1}(\mathbf{v} + \mathbf{r})\rceil.$$

On the other hand, the security relies on the assumption that without knowledge of a special basis (that is, given only the worst possible basis $\mathbf{H}$), solving these instances of the closest vector problem in $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{H})$ is computationally hard. We note that the system described above is *not* semantically secure because the encryption process is deterministic (and thus one can easily distinguish between ciphertexts corresponding to two fixed messages). In practice, one can randomly pad the message in order to resolve this issue (as is often done with the RSA function), although this is not rigorously justified.

Clearly, both the correctness and security depend critically on the choice of the private basis $\mathbf{B}$ and error vector $\mathbf{r}$. Since GGH has been subject to practical attacks, we do not review the specifics of how $\mathbf{B}$ and $\mathbf{r}$ were selected in the GGH cryptosystem, and move on to the description of other cryptosystems.

We remark that no asymptotically good attack to GGH is known: known attacks break the cryptosystem in practice for moderately large values of the security parameter, and can be avoided by making the security parameter even bigger. This, however, makes the cryptosystem impractical. The source of impracticality is similar to that affecting Ajtai's hash function discussed in the previous section, and can be addressed by similar means: general lattice bases require $\Omega(n^2)$ storage, and consequently the encryption/decryption running times also grow quadratically in the security parameter. As we will see shortly, much more efficient cryptosystems can be obtained using lattices with special structure, which admit compact representation.

5.2 The NTRU public key cryptosystem

NTRU is a ring-based cryptosystem proposed by Hoffstein, Pipher and Silverman in [29], which can be equivalently described using lattices with special structure. Below we present NTRU as an instance of the general GGH/HNF framework [19,50] described in the previous subsection. We remark that this is quite different from (but still equivalent to) the original description of NTRU, which, in fact, was proposed concurrently to, and independently from [19].

Using the notation from Section 4, we let $\mathbf{T}$ be the linear transformation in Eq. (5) that rotates the coordinates of the input vector cyclically, and define $\mathbf{T}^*\mathbf{v} = [\mathbf{v}, \mathbf{T}\mathbf{v}, \ldots, \mathbf{T}^{n-1}\mathbf{v}]$ to be the circulant matrix of vector $\mathbf{v} \in \mathbb{Z}^n$. The lattices used by NTRU, named convolutional modular lattices in [29], are lattices in even dimension $2n$ satisfying the following two properties. First, they are closed under the linear transformation that maps the vector $(\mathbf{x}, \mathbf{y})$ (where $\mathbf{x}$ and $\mathbf{y}$ are n-dimensional vectors) to $(\mathbf{T}\mathbf{x}, \mathbf{T}\mathbf{y})$, i.e., the vector obtained by rotating the coordinates of $\mathbf{x}$ and $\mathbf{y}$ cyclically in parallel. Second, they are q-ary lattices, in the sense that they always contain $q\mathbb{Z}^{2n}$ as a sublattice, and hence membership of $(\mathbf{x}, \mathbf{y})$ in the lattice only depends on $(\mathbf{x}, \mathbf{y})$ mod q. The system parameters are a prime dimension n, an integer modulus q, a small integer p, and an integer weight bound d_f. For concreteness, we follow the latest NTRU parameter set recommendations [28], and assume q is a power of 2

(e.g., $q = 2^8$) and $p = 3$. More general parameter choices are possible, some of which are mentioned in [28], and we refer the reader to that publication and the NTRU Cryptosystems web site for details. The NTRU cryptosystem (described by Algorithm 5.1) works as follows:

- **Private Key.** The private key in NTRU is a short vector $(\mathbf{f}, \mathbf{g}) \in \mathbb{Z}^{2n}$. The lattice associated to a private key $(\mathbf{f}, \mathbf{g})$ (and system parameter q) is $\Lambda_q((\mathbf{T}^*\mathbf{f}, \mathbf{T}^*\mathbf{g})^T)$, which can be easily seen to be the smallest convolutional modular lattice containing $(\mathbf{f}, \mathbf{g})$. The secret vectors $\mathbf{f}, \mathbf{g}$ are subject to the following technical restrictions:
  - the matrix $[\mathbf{T}^*\mathbf{f}]$ should be invertible modulo q,
  - $\mathbf{f} \in \mathbf{e}_1 + \{p, 0, -p\}^n$ and $\mathbf{g} \in \{p, 0, -p\}^n$ are randomly chosen polynomials such that $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ have exactly $d_f + 1$ positive entries and d_f negative ones. (The remaining $N - 2d_f - 1$ entries will be zero.)

  The bounds on the number of nonzero entries in $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ are mostly motivated by efficiency reasons. More important are the requirements on the invertibility of $[\mathbf{T}^*\mathbf{f}]$ modulo q, and the restriction of $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ to the set $\{p, 0, -p\}^n$, which are used in the public key computation, encryption and decryption operations. Notice that under these restrictions $[\mathbf{T}^*\mathbf{f}] \equiv \mathbf{I} \pmod{p}$ and $[\mathbf{T}^*\mathbf{g}] \equiv \mathbf{O} \pmod{p}$ (where $\mathbf{O}$ denotes the all zero matrix).
- **Public Key.** Following the general GGH/HNF framework, the NTRU public key corresponds to the HNF basis of the convolutional modular lattice $\Lambda_q((\mathbf{T}^*\mathbf{f}, \mathbf{T}^*\mathbf{g})^T)$ defined by the private key. Due to the structural properties of convolutional modular lattices, and the restrictions on the choice of $\mathbf{f}$, the HNF public basis has an especially nice form

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{T}^*\mathbf{h} & q \cdot \mathbf{I} \end{bmatrix} \qquad \text{where } \mathbf{h} = [\mathbf{T}^*\mathbf{f}]^{-1}\mathbf{g} \pmod{q}, \tag{6}$$

  and can be compactly represented just by the vector $\mathbf{h} \in \mathbb{Z}_q^n$.
- **Encryption.** An input message is encoded as a vector $\mathbf{m} \in \{1, 0, -1\}^n$ with exactly $d_f + 1$ positive entries and d_f negative ones. The vector $\mathbf{m}$ is concatenated with a randomly chosen vector $\mathbf{r} \in \{1, 0, -1\}^n$ also with exactly $d_f + 1$ positive entries and d_f negative ones, to obtain a short error vector $(-\mathbf{r}, \mathbf{m}) \in \{1, 0, -1\}^{2n}$. (The multiplication of $\mathbf{r}$ by -1 is clearly unnecessary, and it is performed here just to keep our notation closer to the original description of NTRU. The restriction on the number of nonzero entries is used to bound the probability of decryption errors.) Reducing the error vector $(-\mathbf{r}, \mathbf{m})$ modulo the public basis $\mathbf{H}$ yields

$$\begin{bmatrix} -\mathbf{r} \\ \mathbf{m} \end{bmatrix} \bmod \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{T}^*\mathbf{h} & q \cdot \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ (\mathbf{m} + [\mathbf{T}^*\mathbf{h}]\mathbf{r}) \bmod q \end{bmatrix}.$$

  Since the first n coordinates of this vector are always 0, they can be omitted, leaving only the n-dimensional vector $\mathbf{c} = \mathbf{m} + [\mathbf{T}^*\mathbf{h}]\mathbf{r} \bmod q$ as the ciphertext.

- **Decryption.** The ciphertext $\mathbf{c}$ is decrypted by multiplying it by the secret matrix $[\mathbf{T}^*\mathbf{f}]$ modulo q, yielding

$$[\mathbf{T}^*\mathbf{f}]\mathbf{c} \bmod q = [\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{f}][\mathbf{T}^*\mathbf{h}]\mathbf{r} \bmod q = [\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r} \bmod q,$$

  where we have used the identity $[\mathbf{T}^*\mathbf{f}][\mathbf{T}^*\mathbf{h}] = [\mathbf{T}^*([\mathbf{T}^*\mathbf{f}]\mathbf{h})]$ valid for any vectors $\mathbf{f}$ and $\mathbf{h}$. The decryption procedure relies on the fact that the coordinates of the vector

$$[\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r} \tag{7}$$

  are all bounded by $q/2$ in absolute value, so the decrypter can recover the exact value of (7) over the integers (i.e., without reduction modulo q.) The bound on the coordinates of (7) holds provided $d_f < (q/2-1)/(4p)-(1/2)$, or, with high probability, even for larger values of d_f. The decryption process is completed by reducing (7) modulo p, to obtain

$$[\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r} \bmod p = \mathbf{I} \cdot \mathbf{m} + \mathbf{O} \cdot \mathbf{r} = \mathbf{m}.$$

Algorithm 5.1 The NTRU public key cryptosystem.

- **Parameters:** Prime n, modulus q, and integer bound d_f. Small integer parameter $p = 3$ is set to a fixed value for simplicity, but other choices are possible.
- **Private key:** Vectors $\mathbf{f} \in \mathbf{e}_1 + \{p, 0, -p\}^n$ and $\mathbf{g} \in \{p, 0, -p\}^n$, such that each of $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ contains exactly $d_f + 1$ positive entries and d_f negative ones, and the matrix $[\mathbf{T}^*\mathbf{f}]$ is invertible modulo q.
- **Public key:** The vector $\mathbf{h} = [\mathbf{T}^*\mathbf{f}]^{-1}\mathbf{g} \bmod q \in \mathbb{Z}_q^n$.
- **Encryption:** The message is encoded as a vector $\mathbf{m} \in \{1, 0, -1\}^n$, and uses as randomness a vector $\mathbf{r} \in \{1, 0, -1\}^n$, each containing exactly $d_f + 1$ positive entries and d_f negative ones. The encryption function outputs $\mathbf{c} = \mathbf{m} + [\mathbf{T}^*\mathbf{h}]\mathbf{r} \bmod q$.
- **Decryption:** On input ciphertext $\mathbf{c} \in \mathbb{Z}_q^n$, output $(([\mathbf{T}^*\mathbf{f}]\mathbf{c}) \bmod q) \bmod p$, where reduction modulo q and p produces vectors with coordinates in $[-q/2, +q/2]$ and $[-p/2, p/2]$ respectively.

This completes the description of the NTRU cryptosystem, at least for the main set of parameters proposed in [28]. Like GGH, no proof of security supporting NTRU is known, and confidence in the security of the scheme is gained primarily from the best currently known attacks. The strongest attack to NTRU known to date was discovered by Howgrave-Graham [30], who combined previous lattice-based attacks of Coppersmith and Shamir [13], with a combinatorial attack due to Odlyzko (reported in [28–30]). Based on Howgrave-Graham's hybrid attack, NTRU Cryptosystems issued a collection of recommended parameter sets [28], some of which are reported in Table 2.

Estimated Security (bits)	n	q	d_f	key size (bits)
80	257	2^{10}	77	2570
80	449	2^8	24	3592
256	797	2^{10}	84	7970
256	14303	2^8	26	114424

Table 2. Some recommended parameter sets for NTRU public key cryptosystem. Security is expressed in "bits", where k-bits of security roughly means that the best known attack to NTRU requires at least an effort comparable to about 2^k NTRU encryption operations. The parameter d_f is chosen in such a way to ensure the probability of decryption errors (by honest users) is at most 2^{-k}. See [28] for details, and a wider range of parameter choices.

5.3 The Ajtai-Dwork cryptosystem and followup work

Following Ajtai's discovery of lattice-based hash functions, Ajtai and Dwork [5] constructed a *public-key cryptosystem* whose security is based on the worst-case hardness of a lattice problem. Several improvements were given in subsequent works [22, 68], mostly in terms of the security proof and simplifications to the cryptosystem. In particular, the cryptosystem in [68] is quite simple as it only involves modular operations on integers, though much longer ones than those typically used in lattice-based cryptography.

Unlike the case of hash functions, the security of these cryptosystems is based on the worst-case hardness of a special case of SVP known as unique-SVP. Here, we are given a lattice whose shortest nonzero vector is shorter by some factor γ than all other nonparallel lattice vectors, and our goal is to find a shortest nonzero lattice vector. The hardness of this problem is not understood as well as that of SVP, and it is a very interesting open question whether one can base public-key cryptosystems on the (worst-case) hardness of SVP.

The aforementioned lattice-based cryptosystems are unfortunately quite inefficient. It turns out that when we base the security on lattices of dimension n, the size of the public key is $\tilde{O}(n^4)$ and each encrypted bit gets blown up to $\tilde{O}(n^2)$ bits. So if, for instance, we choose n to be several hundreds, the public key size is on the order of several gigabytes, which clearly makes the cryptosystem impractical.

Ajtai [4] also presented a more efficient cryptosystem whose public key scales like $\tilde{O}(n^2)$ and in which each encrypted bit gets blown up to $\tilde{O}(n)$ bits. The size of the public key can be further reduced to $\tilde{O}(n)$ if one can set up a pre-agreed trusted random string of length $\tilde{O}(n^2)$. Unfortunately, the security of this cryptosystem is not known to be as strong as that of other lattice-based cryptosystems: it is based on a problem by Dirichlet, which is not directly related to any standard lattice problem. Moreover, this system has no worst-case hardness as the ones previously mentioned. Nevertheless, the system does have the flavor of a lattice-based cryptosystem.

5.4 The LWE-based cryptosystem

In this section we describe what is perhaps the most efficient lattice-based cryptosystem to date supported by a theoretical proof of security. The first version of the cryptosystem together with a security proof were presented by Regev [70]. Some improvements in efficiency were suggested by Kawachi et al. [32]. Then, some very significant improvements in efficiency were given by Peikert et al. [64]. The cryptosystem we describe here is identical to the one in [64] except for one additional optimization that we introduce (namely, the parameter r). Another new optimization based on the use of the Hermite Normal Form [50] is described separately at the end of the subsection. When based on the hardness of lattice problems in dimension n, the cryptosystem has a public key of size $\tilde{O}(n^2)$, requires $\tilde{O}(n)$ bit operations per encrypted bit, and expands each encrypted bit to $O(1)$ bits. This is considerably better than those proposals following the Ajtai-Dwork construction, but is still not ideal, especially in terms of the public key size. We will discuss these issues in more detail later, as well as the possibility of reducing the public key size by using restricted classes of lattices such as cyclic lattices.

The cryptosystem was shown to be secure (under chosen plaintext attacks) based on the conjectured hardness of the *learning with errors* problem (LWE), which we define next. This problem is parameterized by integers n, m, q and a probability distribution χ on $\mathbb{Z}_q$, typically taken to be a "rounded" normal distribution. The input is a pair $(\mathbf{A}, \mathbf{v})$ where $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is chosen uniformly, and $\mathbf{v}$ is either chosen uniformly from $\mathbb{Z}_q^m$ or chosen to be $\mathbf{As}+\mathbf{e}$ for a uniformly chosen $\mathbf{s} \in \mathbb{Z}_q^n$ and a vector $\mathbf{e} \in \mathbb{Z}_q^m$ chosen according to χ^m. The goal is to distinguish with some non-negligible probability between these two cases. This problem can be equivalently described as a bounded distance decoding problem in q-ary lattices: given a uniform $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{v} \in \mathbb{Z}_q^m$ we need to distinguish between the case that $\mathbf{v}$ is chosen uniformly from $\mathbb{Z}_q^m$ and the case in which $\mathbf{v}$ is chosen by perturbing each coordinate of a random point in $\Lambda_q(\mathbf{A}^T)$ using χ.

The LWE problem is believed to be very hard (for reasonable choices of parameters), with the best known algorithms running in exponential time in n (see [70]). Several other facts lend credence to the conjectured hardness of LWE. First, the LWE problem can be seen as an extension of a well-known problem in learning theory, known as the *learning parity with noise* problem, which in itself is believed to be very hard. Second, LWE is closely related to decoding problems in coding theory which are also believed to be very hard. Finally, the LWE was shown to have a worst-case connection, as will be discussed below. In Section 7 we will present several other cryptographic constructions based on the LWE problem.

The worst-case connection:

A reduction from worst-case lattice problems such as approximate-SVP and approximate-SIVP to LWE was established in [70], giving a strong indication

that the LWE problem is hard. This reduction, however, is a *quantum* reduction, i.e., the algorithm performing the reduction is a quantum algorithm. What this means is that hardness of LWE (and hence the security of the cryptosystem) is established based on the worst-case *quantum* hardness of approximate-SVP. In other words, breaking the cryptosystem (or finding an efficient algorithm for LWE) implies an efficient quantum algorithm for approximating SVP, which, as discussed in Subsection 1.3, would be very surprising. This security guarantee is incomparable to the one by Ajtai and Dwork: On one hand, it is stronger as it is based on the general SVP and not the special case of unique-SVP. On the other hand, it is weaker as it only implies a *quantum* algorithm for lattice problems.

The reduction is described in detail in the following theorem, whose proof forms the main bulk of [70]. For a real $\alpha > 0$ we let $\bar{\Psi}_\alpha$ denote the distribution on $\mathbb{Z}_q$ obtained by sampling a normal variable with mean 0 and standard deviation $\alpha q/\sqrt{2\pi}$, rounding the result to the nearest integer and reducing it modulo q.

Theorem 1 ([70]). *Assume we have access to an oracle that solves the* LWE *problem with parameters* $n, m, q, \bar{\Psi}_\alpha$ *where* $\alpha q > \sqrt{n}$, $q \leq \text{poly}(n)$ *is prime, and* $m \leq \text{poly}(n)$. *Then there exists a quantum algorithm running in time* $\text{poly}(n)$ *for solving the (worst-case) lattice problems* $\mathsf{SIVP}_{\tilde{O}(n/\alpha)}$ *and (the decision variant of)* $\mathsf{SVP}_{\tilde{O}(n/\alpha)}$ *in* any *lattice of dimension* n.

Notice that m plays almost no role in this reduction and can be taken to be as large as one wishes (it is not difficult to see that the problem can only become easier for larger m). It is possible that this reduction to LWE will one day be "dequantized" (i.e., made non-quantum), leading to a stronger security guarantee for LWE-based cryptosystems. Finally, let us emphasize that quantum arguments show up only in the reduction to LWE — the LWE problem itself, as well as all cryptosystems based on it are entirely classical.

The cryptosystem:

The cryptosystem is given in Algorithm 5.2, and is partly illustrated in Figure 3. It is parameterized by integers n, m, ℓ, t, r, q, and a real $\alpha > 0$. The parameter n is in some sense the main security parameter, and it corresponds to the dimension of the lattices that show up in the worst-case connection. We will later discuss how to choose all other parameters in order to guarantee security and efficiency. The message space is $\mathbb{Z}_t^\ell$. We let f be the function that maps the message space $\mathbb{Z}_t^\ell$ to $\mathbb{Z}_q^\ell$ by multiplying each coordinate by q/t and rounding to the nearest integer. We also define an "inverse" mapping f^{-1} which takes an element of $\mathbb{Z}_q^\ell$ and outputs the element of $\mathbb{Z}_t^\ell$ obtained by dividing each coordinate by q/t and rounding to the nearest integer.

Algorithm 5.2 The LWE-based public key cryptosystem.

- **Parameters:** Integers n, m, ℓ, t, r, q, and a real $\alpha > 0$.
- **Private key:** Choose $\mathbf{S} \in \mathbb{Z}_q^{n \times \ell}$ uniformly at random. The private key is $\mathbf{S}$.
- **Public key:** Choose $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ uniformly at random and $\mathbf{E} \in \mathbb{Z}_q^{m \times \ell}$ by choosing each entry according to $\bar{\Psi}_\alpha$. The public key is $(\mathbf{A}, \mathbf{P} = \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times \ell}$.
- **Encryption:** Given an element of the message space $\mathbf{v} \in \mathbb{Z}_t^\ell$, and a public key $(\mathbf{A}, \mathbf{P})$, choose a vector $\mathbf{a} \in \{-r, -r+1, \ldots, r\}^m$ uniformly at random, and output the ciphertext $(\mathbf{u} = \mathbf{A}^T\mathbf{a}, \mathbf{c} = \mathbf{P}^T\mathbf{a} + f(\mathbf{v})) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^\ell$.
- **Decryption:** Given a ciphertext $(\mathbf{u}, \mathbf{c}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^\ell$ and a private key $\mathbf{S} \in \mathbb{Z}_q^{n \times \ell}$, output $f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{u})$.

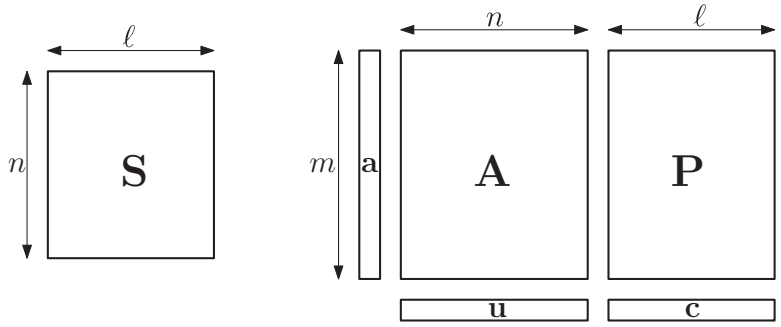

Fig. 3. Ingredients in the LWE-based cryptosystem.

Choosing the parameters

The choice of parameters is meant to guarantee efficiency, a low probability of decryption errors, and security. We now discuss these issues in detail.

Efficiency:

The cryptosystem is clearly very easy to implement, as it involves nothing but additions and multiplications modulo q. Some improvement in running time can be obtained by setting t to be a power of two (which simplifies the task of converting an input message into an element of the message space), and by postponing the modular reduction operations (assuming, of course, that registers are large enough so that no overflow occurs). Moreover, high levels of parallelization are easy to obtain.

In the following we list some properties of the cryptosystem, all of which are easy to observe. All sizes are in bits, logarithms are base 2, and the $\tilde{O}(\cdot)$ notation hides logarithmic factors.

- Private key size: $n\ell \log q$
- Public key size: $m(n + \ell) \log q$
- Message size: $\ell \log t$
- Ciphertext size: $(n + \ell) \log q$

- Encryption blowup factor: $(1 + \frac{n}{\ell}) \log q / \log t$
- Operations for encryption per bit: $\tilde{O}(m(1 + \frac{n}{\ell}))$
- Operations for decryption per bit: $\tilde{O}(n)$

Decryption errors:

The cryptosystem has some positive probability of decryption errors. This probability can be made very small with an appropriate setting of parameters. Moreover, if an error correcting code is used to encode the messages before encryption, this error probability can be reduced to undetectable levels.

We now estimate the probability of a decryption error in one letter, i.e., an element of $\mathbb{Z}_t$ (recall that each message consists of ℓ letters). Assume we choose a private key $\mathbf{S}$, public key $(\mathbf{A}, \mathbf{P})$, encrypt some message $\mathbf{v}$ and then decrypt it. The result is given by

$$\begin{aligned} f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{u}) &= f^{-1}(\mathbf{P}^T\mathbf{a} + f(\mathbf{v}) - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ &= f^{-1}((\mathbf{AS} + \mathbf{E})^T\mathbf{a} + f(\mathbf{v}) - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ &= f^{-1}(\mathbf{E}^T\mathbf{a} + f(\mathbf{v})). \end{aligned}$$

Hence, in order for a letter decryption error to occur, say in the first letter, the first coordinate of $\mathbf{E}^T\mathbf{a}$ must be greater than $q/(2t)$ in absolute value. Fixing the vector $\mathbf{a}$ and ignoring the rounding, the distribution of the first coordinate of $\mathbf{E}^T\mathbf{a}$ is a normal distribution with mean 0 and standard deviation $\alpha q\|\mathbf{a}\|/\sqrt{2\pi}$ since the sum of independent normal variables is still a normal variable with the variance begin the sum of variances. Now the norm of $\mathbf{a}$ can be seen to be with very high probability close to

$$\|\mathbf{a}\| \approx \sqrt{r(r+1)m/3}.$$

To see this, recall that each coordinate of $\mathbf{a}$ is distributed uniformly on $\{-r, \ldots, r\}$. Hence, the expectation squared of each coordinate is

$$\frac{1}{2r+1} \sum_{k=-r}^{r} k^2 = \frac{r(r+1)}{3}$$

from which it follows that $\|\mathbf{a}\|^2$ is tightly concentrated around $r(r+1)m/3$.

The error probability per letter can now be estimated by the probability that a normal variable with mean 0 and standard deviation $\alpha q\sqrt{r(r+1)m/(6\pi)}$ is greater in absolute value than $q/(2t)$, or equivalently,

$$\text{error probability per letter} \;\approx\; 2\left(1 - \Phi\left(\frac{1}{2t\alpha} \cdot \sqrt{\frac{6\pi}{r(r+1)m}}\right)\right) \qquad (8)$$

where Φ here is the cumulative distribution function of the standard normal distribution. For most reasonable choices of parameters, this estimate is in fact very close to the true error probability.

Security:

The proof of security, as given in [70] and [64], consists of two main parts. In the first part, one shows that distinguishing between public keys $(\mathbf{A}, \mathbf{P})$ as generated by the cryptosystem and pairs $(\mathbf{A}, \mathbf{P})$ chosen uniformly at random from $\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{m\times \ell}$ implies a solution to the LWE problem with parameters $n, m, q, \bar{\Psi}_\alpha$. Hence if we set n, m, q, α to values for which we believe LWE is hard, we obtain that the public keys generated by the cryptosystem are indistinguishable from pairs chosen uniformly at random. The second part consists of showing that if one tries to encrypt with a public key $(\mathbf{A}, \mathbf{P})$ chosen at random, then with very high probability, the result carries essentially no statistical information about the encrypted message (this is what [64] refer to as "messy keys"). Together, these two parts establish the security of the cryptosystem (under chosen plaintext attacks). The argument is roughly the following: due to the second part, being able to break the system, even with some small non-negligible probability, implies the ability to distinguish valid public keys from uniform pairs, but this task is hard due to the first part.

In order to guarantee security, our choice of parameters has to be such that the two properties above are satisfied. Let us start with the second one. Our goal is to guarantee that when $(\mathbf{A}, \mathbf{P})$ is chosen uniformly, the encryptions carry no information about the message. For this, it would suffice to guarantee that $(\mathbf{A}^T\mathbf{a}, \mathbf{P}^T\mathbf{a}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^\ell$ is essentially uniformly distributed (since in this case the shift by $f(\mathbf{v})$ is essentially unnoticeable). By following an argument similar to the one in [64, 70], one can show that a sufficient condition for this is that the number of possibilities for $\mathbf{a}$ is much larger than the number of elements in our range, i.e.,

$$(2r+1)^m \gg q^{n+\ell}. \tag{9}$$

More precisely, the statistical distance from the uniform distribution is upper bounded by the square root of the ratio between the two quantities, and hence the latter should be negligible, say 2^{-100}.

We now turn to the first property. Our goal is to choose n, m, q, α so that the LWE problem is hard. One guiding principle we can use is the worst-case connection, as described in Theorem 1. This suggest that the choice of m is inconsequential, that q should be prime, that αq should be bigger than $\sqrt{n}$, and that α should be as big as possible (as it leads to harder worst-case problems). Unfortunately, the worst-case connection does not seem to provide hints on actual security for any concrete choice of parameters. For this, one has to take into account experiments on the hardness of LWE, as we discuss next.

In order to estimate the hardness of LWE for a concrete set of parameters, recall that the LWE can be seen as a certain bounded distance decoding problem on q-ary lattices. Namely, we are given a point $\mathbf{v}$ that is either close to $\Lambda_q(\mathbf{A}^T)$ (with the perturbation in each coordinate chosen according to $\bar{\Psi}_\alpha$) or

uniform. One natural approach to try to distinguish between these two cases is to find a short vector $\mathbf{w}$ in the dual lattice $\Lambda_q(\mathbf{A}^T)^*$ and check the inner product $\langle \mathbf{v}, \mathbf{w} \rangle$: if $\mathbf{v}$ is close to the lattice, this inner product will tend to be close to an integer. This method is effective as long as the perturbation in the direction of $\mathbf{w}$ is not much bigger than $1/\|\mathbf{w}\|$. Since our perturbation is (essentially) Gaussian, its standard deviation in any direction (and in particular in the direction of $\mathbf{w}$) is $\alpha q/\sqrt{2\pi}$. Therefore, in order to guarantee security, we need to ensure that

$$\alpha q/\sqrt{2\pi} \gg 1/\|\mathbf{w}\|.$$

A factor of 1.5 between the two sides of the inequality is sufficient to guarantee that the observed distribution of $\langle \mathbf{v}, \mathbf{w} \rangle \bmod 1$ is within negligible statistical distance of uniform.

Using the results of Section 3, we can predict that the shortest vector found by the best known lattice reduction algorithms when applied to the lattice $\Lambda_q(\mathbf{A}^T)^* = \frac{1}{q}\Lambda_q^{\perp}(\mathbf{A}^T)$ is of length

$$\|\mathbf{w}\| \approx \frac{1}{q} \cdot \min\{q, 2^{2\sqrt{n \log q \log \delta}}\}$$

and that in order to arrive at such a vector (assuming the minimum is achieved by the second term) one needs to apply lattice reduction to lattices of dimension

$$\sqrt{n \log q / \log \delta}. \tag{10}$$

We therefore obtain the requirement

$$\alpha \geq 1.5\sqrt{2\pi} \max\left\{\frac{1}{q}, 2^{-2\sqrt{n \log q \log \delta}}\right\}. \tag{11}$$

The parameter m again seems to play only a minor role in the practical security of the system.

Choice of parameters:

By taking the above discussion into account, we can now finally give some concrete choices of parameters that seem to guarantee both security and efficiency. To recall, the system has seven parameters, n, ℓ, q, r, t, m and α. In order to guarantee security, we need to satisfy Eqs. (9) and (11). To obtain the former, we set

$$m = ((n + \ell) \log q + 200)/\log(2r + 1).$$

Next, following Eq. (11), we set

$$\alpha = 4 \cdot \max\left\{\frac{1}{q}, 2^{-2\sqrt{n \log q \log(1.01)}}\right\}.$$

Our choice of $\delta = 1.01$ seems reasonable for the lattice dimensions with which we are dealing here; one can also consider more conservative choices like $\delta = 1.005$.

We are thus left with five parameters, n, ℓ, q, r, and t. We will choose them in an attempt to optimize the following measures.

- Public key size: $m(n+\ell)\log q$
- Encryption blowup factor: $(1+\frac{n}{\ell})\log q/\log t$
- Error probability per letter:

$$2\left(1-\Phi\left(\frac{1}{2t\alpha}\cdot\sqrt{\frac{6\pi}{r(r+1)m}}\right)\right)$$

- Lattice dimension involved in best known attack:

$$\sqrt{n\log q/\log(1.01)}$$

As a next step, notice that ℓ should not be much smaller than n as this makes the encryption blowup factor very large. For concreteness we choose $\ell = n$, which gives a fair balance between the encryption blowup factor and the public key size. Denoting $N = n\log q$, we are thus left with the following measures.

- Public key size: $2N(2N+200)/\log(2r+1)$
- Encryption blowup factor: $2\log q/\log t$
- Error probability per letter:

$$2\left(1-\Phi\left(\frac{1}{8t}\min\{q, 2^{2\sqrt{N\log(1.01)}}\}\cdot\sqrt{\frac{6\pi}{r(r+1)(2N+200)/\log(2r+1)}}\right)\right)$$

- Lattice dimension involved in best known attack: $\sqrt{N/\log(1.01)}$

Finally, once we fix $N = n\log q$, we should choose q as small as possible and r and t as large as possible while still keeping the error probability within the desired range.

Some examples are given in Table 3. In all examples we took $\ell = n$, and tried to minimize either the public key size or the encryption blowup factor while keeping the error probability below 1%. To recall, this error probability can be made negligible by using an error correcting code. The public key size can be decreased by up to a factor of 2 by choosing a smaller ℓ (at the expense of higher encryption blowup).

Further optimizations:

If all users of the system have access to a trusted source of random bits, they can use it to agree on a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$. This allows us to include only $\mathbf{P}$ in the public key, thereby reducing its size to $m\ell\log q$, which is

n	136	166	192	214	233	233
ℓ	136	166	192	214	233	233
m	2008	1319	1500	1333	1042	4536
q	2003	4093	8191	16381	32749	32749
r	1	4	5	12	59	1
t	2	2	4	4	2	40
α	0.0065	0.0024	0.0009959	0.00045	0.000217	0.000217
PKS	6×10^6	5.25×10^6	7.5×10^6	8×10^6	7.3×10^6	31.7×10^6
EBF	21.9	24	13	14	30	5.6
EPL	0.9%	0.56%	1%	0.8%	0.9%	0.9%
LDA	322	372	417	457	493	493

Table 3. Some possible choices of parameters using $\delta = 1.01$. PKS is the public key size, EBF is the encryption blowup factor, EPL is the error probability per letter, and LDA is the lattice dimension involved in best known attack.

$\tilde{O}(n)$ if ℓ is chosen to be constant and $m = \tilde{O}(n)$. This observation, originally due to Ajtai [4], crucially relies on the source of random bits being trusted, since otherwise it might contain a trapdoor (see [18]). Moreover, as already observed, choosing small ℓ results in large ciphertext blowup factors. If ℓ is set to $O(n)$ in order to achieve constant encryption blowup, then the public key will have size at least $\tilde{O}(n^2)$ even if a common random matrix is used for $\mathbf{A}$.

Another possible optimization results from the HNF technique of [50] already discussed in the context of the GGH cryptosystem. The improvement it gives is quantitatively modest: it allows to shrink the public key size and encryption times by a factor of $(1 - n/m)$. Still, the improvement comes at absolutely no cost, so it seems well worth adopting in any implementation of the system. Recall that the public key consists of a public lattice $\Lambda_q(\mathbf{A}^T)$ represented by a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, and a collection $\mathbf{AS} + \mathbf{E} \bmod q$ of perturbed lattice vectors $\mathbf{As}_i \in \Lambda_q(\mathbf{A}^T)$. As in the HNF modification of the GGH cryptosystem, cryptanalysis only gets harder if we describe the public lattice by its lower triangular HNF basis, and the perturbed lattice vectors are replaced by the result of reducing the error vectors (i.e., the columns of $\mathbf{E}$) by such a basis.

In more detail, let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ be chosen uniformly as before. For simplicity, assume $\mathbf{A}$ has full rank (which happens with probability exponentially close to 1), and that its first n rows are linearly independent over $\mathbb{Z}_q$ (which can be obtained by permuting its rows). Under these conditions, the q-ary lattice $\Lambda_q(\mathbf{A}^T)$ has a very simple HNF basis of the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{A}' & q\mathbf{I} \end{bmatrix}$$

where $\mathbf{A}' \in \mathbb{Z}_q^{(m-n) \times n}$. Let $\mathbf{E}$ be an error matrix chosen as before, and write it as $\mathbf{E} = (\mathbf{E}'', \mathbf{E}')$ where $\mathbf{E}'' \in \mathbb{Z}_q^{n \times \ell}$ and $\mathbf{E}' \in \mathbb{Z}_q^{(m-n) \times \ell}$. Reducing the columns of $\mathbf{E}$ modulo the HNF public basis $\mathbf{H}$ yields vectors $(\mathbf{O}, \mathbf{P}')$ where

$\mathbf{P}' = \mathbf{E}' - \mathbf{A}'\mathbf{E}'' \in \mathbb{Z}_q^{(m-n)\times\ell}$. The public key consists of $(\mathbf{I}, \mathbf{A}') \in \mathbb{Z}_q^{m\times n}$ and $(\mathbf{O}, \mathbf{P}') \in \mathbb{Z}_q^{m\times\ell}$. Since $\mathbf{I}$ and $\mathbf{O}$ are fixed matrices, only $\mathbf{A}'$ and $\mathbf{P}'$ need to be stored as part of the public key, reducing the public key bit-size to $(m-n)(n+\ell)\log q$. Encryption proceeds as before, i.e., the ciphertext is given by

$$(\mathbf{u}, \mathbf{c}) = (\mathbf{a}'' + (\mathbf{A}')^T\mathbf{a}', (\mathbf{P}')^T\mathbf{a}' + f(\mathbf{v}))$$

where $\mathbf{a} = (\mathbf{a}'', \mathbf{a}')$. Notice that the secret matrix $\mathbf{S}$ used by the original LWE cryptosystem has disappeared. The matrix $\mathbf{E}'' \in \mathbb{Z}_q^{n\times\ell}$ is used instead for decryption. Given ciphertext $(\mathbf{u}, \mathbf{c})$, the decrypter outputs $f^{-1}(\mathbf{c} + (\mathbf{E}'')^T\mathbf{u})$. Notice that the vector $\mathbf{c} + (\mathbf{E}'')^T\mathbf{u}$ still equals $(\mathbf{E}^T\mathbf{a}) + f(\mathbf{v})$, so decryption will succeed with exactly the same probability as the original LWE cryptosystem. The security of the system can be established by a reduction from the security of the original cryptosystem. To conclude, this modification allows us to shrink the public key size and encryption time by a factor of $(1 - n/m)$ at no cost.

6 Digital Signature Schemes

Digital signature schemes are among the most important cryptographic primitives. From a theoretical point of view, signature schemes can be constructed from one-way functions in a black-box way without any further assumptions [56]. Therefore, by using the one-way functions described in Section 4 we can obtain signature schemes based on the worst-case hardness of lattice problems. These black-box constructions, however, incur a large overhead and are impractical. In this section we survey some proposals for signature schemes that are directly based on lattice problems, and are typically much more efficient.

The earliest proposal for a lattice-based signature scheme was given by Goldreich et al. [19], and is based on ideas similar to those in their cryptosystem described in Subsection 5.1. In 2003, the company NTRU Cryptosystems proposed an efficient signature scheme called NTRUSign [26]. This signature scheme can be seen as an optimized instantiation of the GGH scheme, based on the NTRU lattices. Unfortunately, both schemes (in their basic version) can be broken in a strong asymptotic sense. We remark that neither scheme came with a security proof, which explains the serious security flaws which we will describe later.

The first construction of efficient signature schemes with a supporting proof of security (in the random oracle model) was suggested by Micciancio and Vadhan [55], who gave statistical zero knowledge proof systems for various lattice problems, and observed that such proof systems can be converted in a relatively efficient way first into secure identification schemes, and then (via the Fiat-Shamir heuristic) into a signature scheme in the random oracle model. More efficient schemes were recently proposed by Lyubashevsky and Micciancio [43], and by Gentry, Peikert and Vaikuntanathan [18]. Interestingly, the latter scheme can be seen as a theoretically justified variant of the

GGH and NTRUSign signature schemes, with worst-case security guarantees based on general lattices in the random oracle model. The scheme of Lyubashevsky and Micciancio [43] has worst-case security guarantees based on ideal lattices similar to those considered in the construction of hash functions (see Section 4), and it is the most (asymptotically) efficient construction known to date, yielding signature generation and verification algorithms that run in almost linear time. Moreover, the security of [43] does not rely on the random oracle model.

In the rest of this section we describe the GGH and NTRUSign signature schemes, and the security flaw in their design, the theoretically justified variant of their scheme proposed by Gentry et al., and finally the signature scheme of Lyubashevsky and Micciancio, which is currently the most efficient (lattice-based) signature scheme with a supporting proof of security, at least in an asymptotic sense.

Lattice-based digital signature schemes have not yet reached the same level of maturity as the collision resistant hash functions and public key encryption schemes presented in the previous sections. So, in this section we present the schemes only informally, and refer the reader to the original papers (and any relevant literature appearing after the time of this writing) for details.

6.1 The GGH and NTRUSign signature schemes

We now briefly describe the GGH signature scheme; for a description of NTRUSign, see [26]. The private and public keys are chosen as in the GGH encryption scheme. That is, the private key is a lattice basis $\mathbf{B}$ consisting of short and fairly orthogonal vectors. The public key $\mathbf{H}$ is a "bad" basis for the same lattice $\mathcal{L}(\mathbf{B})$, i.e., a basis consisting of fairly long and far from orthogonal vectors. As before, it is best to choose $\mathbf{H}$ to be the Hermite normal form of $\mathbf{B}$.

To sign a given message, we first map it to a point $\mathbf{m} \in \mathbb{R}^n$ using some hash function. We assume that the hash function behaves like a random oracle, so that $\mathbf{m}$ is distributed uniformly (in some large volume of space). Next, we round $\mathbf{m}$ to a nearby lattice point $\mathbf{s} \in \mathcal{L}(\mathbf{B})$ by using the secret basis. This is typically done using Babai's round-off procedure [8], which gives

$$\mathbf{s} = \mathbf{B}\lfloor \mathbf{B}^{-1}\mathbf{m} \rceil.$$

Notice that by definition, this implies that

$$\mathbf{s} - \mathbf{m} \in \mathcal{P}_{1/2}(\mathbf{B}) = \{\mathbf{B}\mathbf{x} : \mathbf{x} \in [-1/2, 1/2)^n\}.$$

In order to verify a given message-signature pair $(\mathbf{m}, \mathbf{s})$, one checks that $\mathbf{s} \in \mathcal{L}(\mathbf{H}) = \mathcal{L}(\mathbf{B})$ (which can be done efficiently using the public key $\mathbf{H}$) and that the distance $\|\mathbf{s} - \mathbf{m}\|$ is small (which should be the case since this difference is contained in $\mathcal{P}_{1/2}(\mathbf{B})$).

Attacks:

Some early indications that the GGH and NTRUSign signature schemes might be insecure were given by Gentry and Szydlo [17,76] who observed that each signature leaks some information on the secret key. This information leakage does not necessarily prove that such schemes are insecure, since it might be computationally difficult to use this information. However, as was demonstrated by Nguyen and Regev a few years later [59], this information leakage *does* lead to an attack on the scheme. More precisely, they have shown that given enough message-signature pairs, it is possible to recover the private key. Moreover, their attack is quite efficient, and was implemented and applied in [59] to most reasonable choices of parameters in GGH and NTRUSign, thereby establishing that these signature schemes are not secure in practice (but see below for the use of "perturbations" in NTRUSign).

The idea behind the information leakage and the attack is in fact quite simple. The basic observation is that the difference $\mathbf{m} - \mathbf{s}$ obtained from a message-signature pair $(\mathbf{m}, \mathbf{s})$ is distributed essentially uniformly in $\mathcal{P}_{1/2}(\mathbf{B})$. Hence, given enough such pairs, we end up with the following algorithmic problem, called the hidden parallelepiped problem (see Fig. 4): given many random points uniformly distributed over an unknown n-dimensional parallelepiped, recover the parallelepiped or an approximation thereof. An efficient solution to this problem implies the attack mentioned above.

In the two-dimensional case shown in Fig. 4, one immediately *sees* the parallelepiped enveloping the points, and it is not difficult to come up with an algorithm that implements this. But what about the high-dimensional case? High dimensional problems are often very hard. Here, however, the problem turns out to be easy. The algorithm used in [59] applies a gradient decent method to solve a multivariate optimization problem based on the fourth-moment of the one-dimensional projections. See [59] for further details (as well as for an interesting historical account of the hidden parallelepiped problem).

Countermeasures:

The most efficient countermeasures known against the above attack are perturbation techniques [26, 27]. These modify the signature generation process in such a way that the hidden parallelepiped is replaced by a considerably more complicated body, and this seems to prevent attacks of the type described above. The main drawback of perturbations is that they slow down signature generation and increase the size of the secret key. Nevertheless, the NTRUSign signature scheme with perturbation is still relatively efficient. Finally, notice that even with perturbations, NTRUSign does not have any security proof.

6.2 Schemes based on preimage sampleable trapdoor functions

In a recent paper, Gentry, Peikert, and Vaikuntanathan [18] defined an abstraction called "preimage sampleable trapdoor functions", and showed how

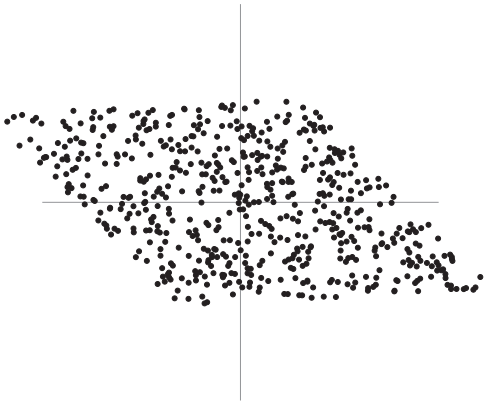

Fig. 4. The hidden parallelepiped problem in two dimensions.

to instantiate it based on the worst-case hardness of lattice problems. They then showed that this abstraction is quite powerful: it can be used instead of trapdoor permutations in several known constructions of signature schemes in the random oracle model. This leads to relatively efficient signature schemes that are provably secure (in the random oracle model) based on the worst-case hardness of lattice problems.

One particularly interesting feature of their construction is that it can be seen as a provably secure variant of the (insecure) GGH scheme. Compared to the GGH scheme, their construction differs in two main aspects. First, it is based on lattices chosen from a distribution that enjoys a worst-case connection (the lattices in GGH and NTRU are believed to be hard, but not known to have a worst-case connection). A second and crucial difference is that their signing algorithm is designed so that it does not reveal any information about the secret basis. This is achieved by replacing Babai's round-off procedure with a "Gaussian sampling procedure", originally due to Klein [35], whose distinctive feature is that its output distribution, for the range of parameters considered in [18], is essentially independent of the secret basis used. The effect of this on the attack outlined above is that instead of observing points chosen uniformly from the parallelepiped generated by the secret basis, the attack observes points chosen from a spherically symmetric Gaussian distribution, and therefore learns nothing about the secret basis. The Gaussian sampling procedure is quite useful, and has already led to the development of several other lattice-based constructions, as will be mentioned in Section 7.

As most schemes based on general lattices, the signatures of [18] have quadratic complexity both in terms of key size and signing and verification times. It should be remarked that although most of the techniques from [18] apply to any lattice, it is not clear how to obtain substantially more efficient instantiations of their signatures using structured lattices (e.g., NTRU lattices, or the cyclic/ideal lattices used in the construction of hash functions).

For example, even when instantiated with NTRU lattices, the running time of the signing algorithm seems to remain quadratic in the security parameter because of the expensive sampling procedure.

6.3 Schemes based on collision resistant hash functions

Finally, in [43], Lyubashevsky and Micciancio gave a signature scheme which is seemingly optimal on all fronts, at least asymptotically: it admits a proof of security based on worst-case complexity assumptions, the proof of security holds in the standard computational model (no need for random oracles), and the scheme is asymptotically efficient, with key size and signing/verification times all almost linear in the dimension of the underlying lattice. The lattice assumption underlying this scheme is that no algorithm can approximate SVP to within polynomial factors in all ideal lattices, i.e., lattices that are closed under some linear transformation $\mathbf{F}$ of the kind considered in Section 4.

The scheme makes use of a new hash-based one-time signature scheme, i.e., a signature scheme that allows to securely sign a single message. Such schemes can be transformed into full-fledged signature schemes using standard tree constructions (dating back to [24, 56]), with only a logarithmic loss in efficiency. The one-time signature scheme, in turn, is based on a collision resistant hash function based on ideal lattices, of the kind discussed in Section 4. The hash function h can be selected during the key generation process, or be a fixed global parameter. The assumption is that finding collisions in h is computationally hard. The input to h can be interpreted as a sequence of vectors $\mathbf{y}_1, \ldots, \mathbf{y}_{m/n} \in \mathbb{Z}_q^n$ with small coordinates. The secret key to the hash function is a pair of randomly chosen inputs $\mathbf{x}_1, \ldots, \mathbf{x}_{m/n} \in \mathbb{Z}_q^n$ and $\mathbf{y}_1, \ldots, \mathbf{y}_{m/n} \in \mathbb{Z}_q^n$, each chosen according to an appropriate distribution that generates short vectors with high probability.[3] The public key is given by the images of these two inputs under the hash function $X = h(\mathbf{x}_1, \ldots, \mathbf{x}_{m/n}), Y = h(\mathbf{y}_1, \ldots, \mathbf{y}_{m/n})$. Messages to be signed are represented by short vectors $\mathbf{m} \in \mathbb{Z}_q^n$. The signature of a message $\mathbf{m}$ is simply computed as

$$\sigma = (\sigma_1, \ldots, \sigma_{m/n}) = ([\mathbf{F}^*\mathbf{m}]\mathbf{x}_1 + \mathbf{y}_1, \ldots, [\mathbf{F}^*\mathbf{m}]\mathbf{x}_{m/n} + \mathbf{y}_{m/n}) \bmod q.$$

The signature is verified by checking that σ is a sequence of short vectors that hashes to $[\mathbf{F}^*\mathbf{m}]X + Y \bmod q$.

The security of the scheme relies on the fact that even after seeing a signature, the exact value of the secret key is still information theoretically concealed from the adversary. Therefore, if the adversary manages to come up with a forged signature, it is likely to be different from the one that the legitimate signer can compute using the secret key. Since the forged signature and legitimate signature hash to the same value, they provide a collision in the hash function.

[3] For technical reasons, the input vectors cannot be chosen simply uniformly at random from a set of short vectors without invalidating the proof.

7 Other Cryptographic Primitives

In this section we briefly survey lattice-based constructions of other cryptographic primitives. Previous constructions of these primitives were based on (sometimes non-standard) number theoretic assumptions. Since all these constructions are very recent, we will not provide too many details.

CCA-secure cryptosystems:

All the cryptosystems mentioned in Section 5 are secure only under chosen plaintext attacks (CPA), and not under chosen ciphertext attacks (CCA). Indeed, it is not difficult to see that given access to the decryption oracle, one can recover the private key. For certain applications, security against CCA attacks is necessary.

CCA-secure cryptosystems are typically constructed based on specific number theoretic assumptions (or in the random oracle model) and no general constructions in the standard model were known till very recently. In a recent breakthrough, Peikert and Waters [65] showed for the first time how to construct CCA-secure cryptosystems based on a general primitive which they call *lossy trapdoor functions*. They also showed how to construct this primitive based either on traditional number theoretic assumptions or on the LWE problem. The latter result is particularly important as it gives for the first time a CCA-secure cryptosystem based on the worst-case (quantum) hardness of lattice problems.

IBE:

Gentry et al. [18] have recently constructed identity based encryption (IBE) schemes based on LWE. Generally speaking, IBE schemes are difficult to construct and only a few other proposals are known; the fact that IBE schemes can be based on the LWE problem (and hence on the worst-case quantum hardness of lattice problems) is therefore quite remarkable.

OT protocols:

In another recent work, Peikert, Vaikuntanathan, and Waters [64] provide a construction of an oblivious transfer (OT) protocol that is both universally composable and relatively efficient. Their construction can be based on a variety of cryptographic assumptions, and in particular on the LWE problem (and hence on the worst-case quantum hardness of lattice problems). Such protocols are often used in secure multiparty computation.

Zero-Knowledge proofs and ID schemes:

Various zero-knowledge proof systems and identification schemes were recently discovered. *Interactive* statistical zero-knowledge proof systems for various lattice problems (including approximate SVP) were already given by Micciancio

and Vadhan in [55]. In [63], Peikert and Vaikuntanathan gave *non-interactive* statistical zero-knowledge proof systems for approximate SIVP and other lattice problems. Zero-knowledge proof systems are potentially useful building blocks both in the context of key registration in a public-key infrastructure (PKI), and in the construction of identification (ID) protocols. Finally, more efficient identification protocols (than those obtainable from zero-knowledge) were recently discovered by Lyubashevsky [44]. Remarkably, the proof systems of [44] are not zero-knowledge, and still they achieve secure identification under active attacks using an interesting aborting technique.

8 Open Questions

- **Cryptanalysis:** The experiments of [16] are very useful to gain some insight into the concrete hardness of lattice problems for specific values of the lattice dimension, as needed by lattice-based cryptography. But more work is still needed to increase our confidence and understanding, and in order to support widespread use of lattice-based cryptography. An interesting recent effort in this direction is the "Lattice Challenge" web page created by Lindner and Rückert [10, 40], containing a collection of randomly chosen lattices in increasing dimension for which finding short vectors is apparently hard.
- **Improved cryptosystems:** The LWE-based cryptosystem described in Section 5.4 is reasonably efficient and has a security proof based on a worst-case connection. Still, one might hope to considerably improve the efficiency, and in particular the public key size, by using structured lattices such as cyclic lattices. Another desirable improvement is to obtain a classical (i.e., non-quantum) worst-case connection. Finally, obtaining practical CCA-secure cryptosystems in the standard model is another important open question.
- **Comparison with number theoretic cryptography:** Can one factor integers or compute discrete logarithms using an oracle that solves, say, $\sqrt{n}$-approximate SVP? Such a result would prove that the security of lattice-based cryptosystems is superior to that of traditional number-theoretic-based cryptosystems (see [1, 74] for related work).

Acknowledgements

We thank Phong Nguyen and Markus Rückert for helpful discussions on the practical security of lattice-based cryptography. We also thank Richard Lindner, Vadim Lyubashevsky, and Chris Peikert for comments on an earlier version.

References

1. Adleman, L.M.: Factoring and lattice reduction (1995). Unpublished manuscript.
2. Aharonov, D. and Regev, O.: Lattice problems in NP intersect coNP. *Journal of the ACM*, 52(5):749–765 (2005). Preliminary version in FOCS 2004.
3. Ajtai, M.: The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract) 10-19. In *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, pages 10–19. ACM (1998).
4. Ajtai, M.: Representing hard lattices with $O(n \log n)$ bits. In *Proc. 37th Annual ACM Symp. on Theory of Computing (STOC)* (2005).
5. Ajtai, M. and Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. 29th Annual ACM Symp. on Theory of Computing (STOC)*, pages 284–293 (1997).
6. Ajtai, M., Kumar, R., and Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 601–610 (2001).
7. Ajtai, M.: Generating hard instances of lattice problems. In *Complexity of computations and proofs*, volume 13 of *Quad. Mat.*, pages 1–32. Dept. Math., Seconda Univ. Napoli, Caserta (2004). Preliminary version in STOC 1996.
8. Babai, L.: On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13 (1986).
9. Blum, A., Kalai, A., and Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519 (2003). Preliminary version in STOC'00.
10. Buchmann, J., Lindner, R., and Rückert, M.: Creating a lattice challenge (2008). Manuscript.
11. Cai, J.Y. and Nerurkar, A.: An improved worst-case to average-case connection for lattice problems. In *Proc. 38th IEEE Symp. on Found. of Comp. Science*, pages 468–477 (1997).
12. Cai, J.Y. and Nerurkar, A.: Approximating the SVP to within a factor $(1 + 1/\dim^{\epsilon})$ is NP-hard under randomized reductions. *J. Comput. System Sci.*, 59(2):221–239 (1999). ISSN 0022-0000.
13. Coppersmith, D. and Shamir, A.: Lattice attacks on NTRU. In *Proc. of Eurocrypt '97*, volume 1233 of *LNCS*. IACR, Springer (1997).
14. Dinur, I., Kindler, G., Raz, R., and Safra, S.: Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243 (2003).
15. Gama, N. and Nguyen, P.Q.: Finding short lattice vectors within Mordell's inequality. In *Proc. 40th ACM Symp. on Theory of Computing (STOC)*, pages 207–216 (2008).
16. Gama, N. and Nguyen, P.Q.: Predicting lattice reduction. In *Advances in Cryptology – Proc. Eurocrypt '08*, Lecture Notes in Computer Science. Springer (2008).
17. Gentry, C. and Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In *Proc. of Eurocrypt '02*, volume 2332 of *LNCS*. Springer-Verlag (2002).
18. Gentry, C., Peikert, C., and Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In *Proc. 40th ACM Symp. on Theory of Computing (STOC)*, pages 197–206 (2008).

19. Goldreich, O., Goldwasser, S., and Halevi, S.: Public-key cryptosystems from lattice reduction problems. In *Advances in cryptology*, volume 1294 of *Lecture Notes in Comput. Sci.*, pages 112–131. Springer (1997).
20. Goldreich, O. and Goldwasser, S.: On the limits of nonapproximability of lattice problems. *Journal of Computer and System Sciences*, 60(3):540–563 (2000). Preliminary version in STOC 1998.
21. Goldreich, O., Goldwasser, S., and Halevi, S.: Collision-free hashing from lattice problems. Technical Report TR96-056, Electronic Colloquium on Computational Complexity (ECCC) (1996).
22. Goldreich, O., Goldwasser, S., and Halevi, S.: Eliminating decryption errors in the Ajtai-Dwork cryptosystem. In *Advances in cryptology*, volume 1294 of *Lecture Notes in Comput. Sci.*, pages 105–111. Springer (1997).
23. Goldwasser, S. and Micali, S.: Probabilistic encryption. *Journal of Computer and System Sience*, 28(2):270–299 (1984). Preliminary version in Proc. of STOC 1982.
24. Goldwasser, S., Micali, S., and Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. on Computing*, 17(2):281–308 (1987).
25. Haviv, I. and Regev, O.: Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proc. 39th ACM Symp. on Theory of Computing (STOC)*, pages 469–477 (2007).
26. Hoffstein, J., Graham, N.A.H., Pipher, J., Silverman, J.H., and Whyte, W.: NTRUSIGN: Digital signatures using the NTRU lattice. In *Proc. of CT-RSA*, volume 2612 of *Lecture Notes in Comput. Sci.*, pages 122–140. Springer-Verlag (2003).
27. Hoffstein, J., Graham, N.A.H., Pipher, J., Silverman, J.H., and Whyte, W.: Performances improvements and a baseline parameter generation algorithm for NTRUsign. In *Proc. of Workshop on Mathematical Problems and Techniques in Cryptology*, pages 99–126. CRM (2005).
28. Hoffstein, J., Howgrave-Graham, N., Pipher, J., and Silverman, J.H.: Hybrid lattice reduction and meet in the middle resistant parameter selection for NTRUEncrypt. Submission/contribution to ieee p1363.1, NTRU Cryptosystems, Inc., URL http://grouper.ieee.org/groups/1363/lattPK/submissions.html#2007-02 (2007).
29. Hoffstein, J., Pipher, J., and Silverman, J.H.: NTRU: a ring based public key cryptosystem. In *Proceedings of ANTS-III*, volume 1423 of *LNCS*, pages 267–288. Springer (1998).
30. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Advances in cryptology (CRYPTO)*, pages 150–169 (2007).
31. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In *Proc. 15th ACM Symp. on Theory of Computing (STOC)*, pages 193–206. ACM (1983).
32. Kawachi, A., Tanaka, K., and Xagawa, K.: Multi-bit cryptosystems based on lattice problems. In *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Comput. Sci.*, pages 315–329. Springer, Berlin (2007).
33. Khot, S.: Hardness of approximating the shortest vector problem in lattices. In *Proc. 45th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 126–135 (2004).

34. Khot, S.: Inapproximability results for computational problems on lattices (2007). Survey paper prepared for the LLL+25 conference. To appear.
35. Klein, P.: Finding the closest lattice vector when it's unusually close. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–941 (2000).
36. Kumar, R. and Sivakumar, D.: Complexity of SVP – a reader's digest. *SIGACT News*, 32(3):40–52 (2001). doi:http://doi.acm.org/10.1145/582475.582484.
37. Lagarias, J.C., Lenstra, Jr., H.W., and Schnorr, C.P.: Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10(4):333–348 (1990).
38. Lenstra, A.K. and Lenstra, Jr., H.W., editors: *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin (1993). ISBN 3-540-57013-6.
39. Lenstra, A.K., Lenstra, Jr., H.W., and Lovász, L.: Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534 (1982).
40. Lindner, R. and Rückert, M.: The lattice challence (2008). Available at `http://www.latticechallenge.org/`.
41. Ludwig, C.: A faster lattice reduction method using quantum search. In *ISAAC*, pages 199–208 (2003).
42. Lyubashevsky, V. and Micciancio, D.: Generalized compact knapsacks are collision resistant. In *33rd International Colloquium on Automata, Languages and Programming (ICALP)* (2006).
43. Lyubashevsky, V. and Micciancio, D.: Asymptotically efficient lattice-based digital signatures. In *Fifth Theory of Cryptography Conference (TCC)*, volume 4948 of *Lecture Notes in Computer Science*. Springer (2008).
44. Lyubashevsky, V.: Lattice-based identification schemes secure under active attacks. In *PKC'08*, number 4939 in LNCS, pages 162–179 (2008).
45. Lyubashevsky, V., Micciancio, D., Peikert, C., and Rosen, A.: SWIFFT: a modest proposal for FFT hashing. In *FSE 2008* (2008).
46. McEliece, R.: A public-key cryptosystem based on algebraic number theory. Technical report, Jet Propulsion Laboratory (1978). DSN Progress Report 42-44.
47. Micciancio, D.: The shortest vector problem is NP-hard to approximate to within some constant. *SIAM J. on Computing*, 30(6):2008–2035 (2001). Preliminary version in FOCS 1998.
48. Micciancio, D.: Improved cryptographic hash functions with worst-case/average-case connection. In *Proc. 34th ACM Symp. on Theory of Computing (STOC)*, pages 609–618 (2002).
49. Micciancio, D. and Goldwasser, S.: *Complexity of Lattice Problems: A Cryptographic Perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts (2002).
50. Micciancio, D.: Improving lattice based cryptosystems using the hermite normal form. In J. Silverman, editor, *Cryptography and Lattices Conference — CaLC 2001*, volume 2146 of *Lecture Notes in Computer Science*, pages 126–145. Springer-Verlag, Providence, Rhode Island (2001).
51. Micciancio, D.: Lattices in cryptography and cryptanalysis (2002). Lecture notes of a course given in UC San Diego.
52. Micciancio, D.: Cryptographic functions from worst-case complexity assumptions (2007). Survey paper prepared for the LLL+25 conference. To appear.

53. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. *Computational Complexity*, 16(4):365–411 (2007). Preliminary versions in FOCS 2002 and ECCC TR04-095.
54. Micciancio, D. and Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In *Proc. 45th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 372–381 (2004).
55. Micciancio, D. and Vadhan, S.: Statistical zero-knowledge proofs with efficient provers: lattice problems and more. In *Advances in cryptology (CRYPTO)*, volume 2729 of *Lecture Notes in Computer Science*, pages 282–298. Springer-Verlag (2003).
56. Naor, M. and Yung, M.: Universal one-way hash functions and their cryptographic applications. In *Proc. 21st ACM Symp. on Theory of Computing (STOC)*, pages 33–43 (1989).
57. Nguyen, P.Q. and Vidick, T.: Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology* (2008). To appear.
58. Nguyen, P. and Stern, J.: Cryptanalysis of the Ajtai-Dwork cryptosystem. In *Advances in cryptology (CRYPTO)*, volume 1462 of *Lecture Notes in Comput. Sci.*, pages 223–242. Springer (1998).
59. Nguyen, P.Q. and Regev, O.: Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In *The 25th International Cryptology Conference (Eurocrypt)*, pages 271–288 (2006).
60. Nguyen, P.Q. and Stern, J.: The two faces of lattices in cryptology. In J.H. Silverman, editor, *Cryptography and Lattices, International Conference (CaLC 2001)*, number 2146 in Lecture Notes in Computer Science, pages 146–180 (2001).
61. Peikert, C. and Rosen, A.: Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *3rd Theory of Cryptography Conference (TCC)*, pages 145–166 (2006).
62. Peikert, C. and Rosen, A.: Lattices that admit logarithmic worst-case to average-case connection factors. In *Proc. 39th ACM Symp. on Theory of Computing (STOC)*, pages 478–487 (2007).
63. Peikert, C. and Vaikuntanathan, V.: Noninteractive statistical zero-knowledge proofs for lattice problems. In *Advances in Cryptology (CRYPTO)*, LNCS. Springer (2008).
64. Peikert, C., Vaikuntanathan, V., and Waters, B.: A framework for efficient and composable oblivious transfer. In *Advances in Cryptology (CRYPTO)*, LNCS. Springer (2008).
65. Peikert, C. and Waters, B.: Lossy trapdoor functions and their applications. In *Proc. 40th ACM Symp. on Theory of Computing (STOC)*, pages 187–196 (2008).
66. Peikert, C.J.: Limits on the hardness of lattice problems in ℓ_p norms. *Computational Complexity* (2008). To appear. Preliminary version in Proc. of CCC 2007.
67. Regev, O.: Lattices in computer science (2004). Lecture notes of a course given in Tel Aviv University.
68. Regev, O.: New lattice-based cryptographic constructions. *Journal of the ACM*, 51(6):899–942 (2004). Preliminary version in STOC'03.
69. Regev, O.: Quantum computation and lattice problems. *SIAM J. on Computing*, 33(3):738–760 (2004). Preliminary version in FOCS'02.

70. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In *Proc. 37th ACM Symp. on Theory of Computing (STOC)*, pages 84–93 (2005).
71. Regev, O.: Lattice-based cryptography. In *Advances in cryptology (CRYPTO)*, pages 131–141 (2006).
72. Regev, O.: On the complexity of lattice problems with polynomial approximation factors (2007). Survey paper prepared for the LLL+25 conference. To appear.
73. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2-3):201–224 (1987).
74. Schnorr, C.P.: Factoring integers and computing discrete logarithms via Diophantine approximation. In J.Y. Cai, editor, *Advances in computational complexity*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 171–182. AMS (1993). Preliminary version in Eurocrypt '91.
75. Shoup, V.: NTL: A library for doing number theory. Available at `http://www.shoup.net/ntl/`.
76. Szydlo, M.: Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In *Proc. of Eurocrypt '03*, volume 2656 of *LNCS*. Springer-Verlag (2003).
77. van Emde Boas, P.: Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, Department of Mathematics, Netherlands (1981). Technical Report 8104.
78. Wagner, D.: A generalized birthday problem. In *Advances in cryptology (CRYPTO)*, volume 2442 of *LNCS*, pages 288–303. Springer (2002).

Multivariate Public Key Cryptography

Jintai Ding[1] and Bo-Yin Yang[2]

[1] University of Cincinnati and Technische Universität Darmstadt.
[2] Academia Sinica and Taiwan InfoSecurity Center, Taipei, Taiwan.

Summary. A multivariate public key cryptosystem (MPKCs for short) have a set of (usually) quadratic polynomials over a finite field as its public map. Its main security assumption is backed by the NP-hardness of the problem to solve nonlinear equations over a finite field. This family is considered as one of the major families of PKCs that could resist potentially even the powerful quantum computers of the future. There has been fast and intensive development in Multivariate Public Key Cryptography in the last two decades. Some constructions are not as secure as was claimed initially, but others are still viable. The paper gives an overview of multivariate public key cryptography and discusses the current status of the research in this area.

Keywords: Gröbner basis, multivariate public key cryptosystem, linear algebra, differential attack

1 Introduction

As envisioned by Diffie and Hellman, a public key cryptosystem (hereafter PKC for short) depends on the existence of class of "trapdoor one-way functions". This class and the mathematical structure behind it will determine all the essential characteristics of the PKC. So for example behind elliptic cryptography is the elliptic curve group, and behind NTRU stands the structure of an integral lattice.

Multivariate (Public-Key) Cryptography is the study of PKCs where the trapdoor one-way function takes the form of a multivariate quadratic polynomial map over a finite field. Namely the public key is in general given by a set of quadratic polynomials:

$$\mathcal{P} = (p_1(w_1, \ldots, w_n), \ldots, p_m(w_1, \ldots, w_n)),$$

where each p_i is a (usu. quadratic) nonlinear polynomial in $\mathbf{w} = (w_1, \ldots, w_n)$:

$$z_k = p_k(\mathbf{w}) := \sum_i P_{ik} w_i + \sum_i Q_{ik} w_i^2 + \sum_{i>j} R_{ijk} w_i w_j \tag{1}$$

with all coefficients and variables in $\mathbb{K} = \mathbb{F}_q$, the field with q elements. The evaluation of these polynomials at any given value corresponds to either the encryption procedure or the verification procedure. Such PKCs are called multivariate public key cryptosystems (hereafter MPKCs). Inverting a multivariate quadratic map is equivalent to solving a set of quadratic equations over a finite field, or the following problem:

Problem $\mathcal{MQ}$: Solve the system $p_1(\mathbf{x}) = p_2(\mathbf{x}) = \cdots = p_m(\mathbf{x}) = 0$, where each p_i is a quadratic in $\mathbf{x} = (x_1, \ldots, x_n)$. All coefficients and variables are in $\mathbb{K} = \mathbb{F}_q$, the field with q elements.

$\mathcal{MQ}$ is in general an NP-hard problem. Such problems are believed to be hard unless the class P is equal to NP. Of course, a random set of quadratic equations would not have a trapdoor and hence not be usable in an MPKC. The corresponding mathematical structure to a system of polynomial equations, not necessarily generic, is the ideal generated by those polynomials. So, philosophically speaking, multivariate cryptography relate to mathematics that handles polynomial ideals, namely algebraic geometry.

In contrast, the security of RSA-type cryptosystems relies on the complexity of integer factorization and is based on results in number theory developed in the 17^{th} and 18^{th} centuries. Elliptic curve cryptosystems employ the use of mathematics from the 19^{th} century. This quote is actually from Whitfield Diffie at the RSA Europe conference in Paris in 2002. At least Algebraic Geometry, the mathematics that MPKCs use, is developed in the 20^{th} century.

Since we are no longer dealing with "random" or "generic" systems, but systems where specific trapdoors exist, the security of MPKCs is then not guaranteed by the NP-hardness of $\mathcal{MQ}$, and effective attacks may exist for any chosen trapdoor. The history of MPKCs therefore evolves as we understand more and more about how to design secure multivariate trapdoors.

Sec. 2 is a sketch of how MPKCs work in general. Sec. 3 gives examples of current MPKCs. Sec. 4 describes the known trapdoor constructions in somewhat more detail. Sec. 5 describes the most important mode of attacks. The last section will be a short discussion about future development.

2 The Basics of Multivariate PKCs

After Diffie-Hellman [28], cryptographers proposed many trapdoor functions. Most of these were forgotten and RSA became dominant. The earliest published proposals of MPKCs scheme by Shigeo Tsujii and Hideki Imai, seemed to have arisen around this time. They are independently known to have worked on this topic in the early 1980s. Certainly lectures are given on this topic no later than 1983. However, for several years, their work were not published in anything other than Japanese, and remained largely unknown outside Japan.

As far as we know, the first article written in English describing a PKC with more than one independent variable may be the one from Ong *et al*

[78], and the first use of more than one equation is by Fell and Diffie [52]. The earliest attempt bearing some resemblance to today's MPKCs (with 4 variables) seems to be [71]. In 1988, the first MPKC in the modern form appears [70]. It seems as if basic construction described below (cf. Sec. 2.1) has not changed for 20 years.

2.1 The Standard (Bipolar) Construction and Notations

Even if we restrict ourselves to cryptosystems for which the public key is a set of polynomials $\mathcal{P} = (p_1, \ldots, p_m)$ in variables $\mathbf{w} = (w_1, \ldots, w_n)$ where all variables and coefficients are in $\mathbb{K} = \mathbb{F}_q$, the way to hide the trapdoor is not unique.

However, extant MPKCs almost always hide the private map $\mathcal{Q}$ via composition with two affine maps S, T. So, $\mathcal{P} = T \circ \mathcal{Q} \circ S : \mathbb{K}^n \to \mathbb{K}^m$, or

$$\mathcal{P} : \mathbf{w} = (w_1, \ldots, w_n) \overset{S}{\mapsto} \mathbf{x} = \mathrm{M}_S\mathbf{w} + \mathbf{c}_S \overset{\mathcal{Q}}{\mapsto} \mathbf{y} \overset{T}{\mapsto} \mathbf{z} = \mathrm{M}_T\mathbf{y} + \mathbf{c}_T = (z_1, \ldots, z_m) \tag{2}$$

In any given scheme, the *central map* $\mathcal{Q}$ belongs to a certain class of quadratic maps whose inverse can be computed relatively easily. The maps S, T are affine (sometimes linear) and full-rank. The x_j are called the central variables. The polynomials giving y_i in $\mathbf{x}$ are called the central polynomials; when necessary to distinguish between the variable and the value, we will write $y_i = q_i(\mathbf{x})$. The key of a MPKC is the design of the central map.

The public key consists of the polynomials in $\mathcal{P}$. In practice, this is always the collection of the coefficients of the p_i's, compiled in some order conducive to easy computation. Since we are doing public-key cryptography, $\mathcal{P}(0)$ is always taken to be zero, hence public polynomials do not have constant terms.

The secret key consists of the informations in S, T, and $\mathcal{Q}$. That is, we collect $(\mathrm{M}_S^{-1}, \mathbf{c}_S)$, $(\mathrm{M}_T^{-1}, \mathbf{c}_T)$ and whatever parameters there exist in $\mathcal{Q}$. In theory one of $\mathbf{c}_S$ and $\mathbf{c}_T$ is extraneous but we keep it anyway.

To verify a signature or to encrypt a block, one simply computes $\mathbf{z} = \mathcal{P}(\mathbf{w})$. To sign or to decrypt a block, one computes $\mathbf{y} = T^{-1}(\mathbf{z})$, $\mathbf{x} = \mathcal{Q}^{-1}(\mathbf{y})$ and $\mathbf{w} = S^{-1}(\mathbf{x})$ in turn. Notice that these may be only one of the many pre-images, not necessarily an inverse function in the strict sense of the word.

We summarize the notations used in Table 1 and will henceforth use it consistently to make our exposition easier to understand. And we summarize operating details below so that the reader will have some basic sense of about how these schemes can be applied practically.

Cipher block or Message digest Size: m elements of $\mathbb{F}_q$
Plaintext block or Signature Size: n elements of $\mathbb{F}_q$
Public Key Size: $mn(n+3)/2$ $\mathbb{F}_q$-elements, often stored in log-form
Secret Key Size: Usually $\left(n^2 + m^2 + [\#\text{ parameters in } \mathcal{Q}]\right)$ $\mathbb{F}_q$-elements, often stored in log-form

α	the power in a C^* construction
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	constant vectors
$\mathbf{c}_S, \mathbf{c}_T$	constant parts of linear maps S, T
$C^* = (c_1^*, c_2^*, \ldots, c_n^*)$	the Matsumoto-Imai map $C^*_{q,n,\alpha}: \mathbf{x} \mapsto \mathbf{y} = \mathbf{x}^{q^\alpha+1}$ in $\mathbb{F}_{q^n}$
DF	(symmetric) differential of the function/map F
D, D_{reg}, and D_{XL}	degree in system-solving degree, operating degree of $\mathbf{F_4}$/$\mathbf{F_5}$ and XL
$\mathbb{F}_q$	finite (Galois) field of q elements, any representation of
g	sometimes, a generator of $\mathbb{K} = \mathbb{F}_q$
H_i	symmetric matrices for quadratic part of p_i (or z_i) in w_i
h, i, j, k, l	index variables, k often $:= [\mathbb{L} : \mathbb{K}]$, dimension of $\mathbb{L}$ over $\mathbb{K}$
$\mathcal{K}$	denoting a kernel
$\ker_{\mathbf{v}} f$	kernel of the symmetric matrix denoting quadratic part of f as function of $\mathbf{v}$.
$\mathbb{K}$	the base field, usually $= \mathbb{F}_q$
$\mathbb{L}$	$\mathbb{F}_{q^k}$, a field that is larger than $\mathbb{K}$
M_i	symmetric matrices for the quadratic part of y_i in x_j
M_S, M_T	matrices of linear maps S, T.
m	number of equations
$\mathfrak{m}$	a multiplication, as a unit of time
n	number of variables
$O()$, $o()$, $\Omega()$	standard big-O, small-o, Omega notations
o	number of oil variables
P_{ik}	Matsumoto-Imai notation for coefficient of w_i in z_k
$\mathcal{P} = (p_1, \ldots, p_m)$	public map
Q_{ik}	Matsumoto-Imai notation for coefficient of w_i^2 in z_k
$\mathcal{Q} = (q_1, \ldots, q_m)$	central map
q	the size of the base field
R_{ijk}	Matsumoto-Imai notation for coefficient of $w_i w_j$ in z_k
R	$\lvert\mathcal{R}\rvert$, the number of relations (equations) in XL or $\mathbf{F_4}$
$\mathcal{R}^{(D)}$ or $\mathcal{R}$	Set of equations in XL or $\mathbf{F_4}$
r	usu. the minimum rank or # of removed (minus) equations
S	the initial linear map, $S(\mathbf{w}) = \mathbf{x} = \mathrm{M}_S\mathbf{w} + \mathbf{c}_S$
T	the final linear map, $T(\mathbf{y}) = \mathbf{z} = \mathrm{M}_T\mathbf{y} + \mathbf{c}_T$, or #terms in XL ($\lvert\mathcal{T}\rvert$ below)
$\mathcal{T}^{(D)}$ or $\mathcal{T}$	set of terms (monomials) in XL or $\mathbf{F_4}$
u	often the high rank parameter or # of Rainbow stages
v	number of vinegar variables
$v_1 < v_2 < \cdots < v_{u+1} = n$	structure of Rainbow (v_1, $o_1, \ldots, o_u$), $o_i := v_{i+1} - v_i$
$\mathbf{w} = (w_1, \ldots, w_n)$	signature or plaintext block
X_i, Y_j	elements in intermediate fields
$\mathbf{x} = (x_1, \ldots, x_n)$	central variables, input to central map $\mathcal{Q}$
$\mathbf{y} = (y_1, \ldots, y_m)$	output of central map $\mathcal{Q}$, central polynomials
$\mathbf{z} = (z_1, \ldots, z_m)$	digest or ciphertext block

Table 1. Notations and Terminology

Secret Map Time Complexity: $(n^2 + m^2)$ $\mathbb{F}_q$-multiplications, plus whatever time it is needed to invert $\mathcal{Q}$
Public Map Time Complexity: About $mn^2/2$ $\mathbb{F}_q$-multiplications
Key Generation Time Complexity: n^2 times the invocation cost of $\mathcal{P}$; between $O(n^4)$ and $O(n^5)$

We immediately see the major disadvantage with MPKCs: Their keys are very large compared to traditional systems like RSA or ECC. For example, the public key size of RSA-2048 is not much more than 2048 bits, but a current version of the Rainbow signature scheme has $n = 42$, $m = 24$, $q = 256$, i.e., the size of the public key is 22680 bytes, above the 16kB of flash memory that some small smartcards have. Private keys are smaller, but still formidable for small embedded devices which has memory constraints. However operating on units hundreds of bits long (for Elliptic Curve groups and especially RSA) is prohibitively expensive for embedded devices without a co-processor. So MPKCs have some compensating advantages and still has potential on those devices.

2.2 Other Constructions

It should be noted that MPKCs are also sometimes called trapdoor $\mathcal{MQ}$ schemes for a reason, all the construction currently used do quadratic public keys for speed reasons – with higher order terms, the explosion in number of coefficients offset any possible gain in efficiency. Furthermore, in the bipolar form, higher-order terms may in fact hurt the security.

Here we cover two alternatives in which multivariate polynomials can be used for PKCs. These are called the Implicit Form and Isomorphisms of Polynomials.

Implicit Form MPKCs

The public key is a system of l equations

$$\mathcal{P}(\mathbf{w}, \mathbf{z}) = \mathcal{P}(w_1, \ldots, w_n, z_1, \ldots, z_m) = (p_1(\mathbf{w}, \mathbf{z}), \ldots, p_l(\mathbf{w}, \mathbf{z})) = (0, \ldots, 0), \tag{3}$$

where each p_i is a polynomial in $\mathbf{w} = (w_1, \ldots, w_n)$ and $\mathbf{z} = (z_1, \ldots, z_m)$. This $\mathcal{P}$ is built from the secret $\mathcal{Q}$

$$\mathcal{Q}(\mathbf{x}, \mathbf{y}) = q(x_1, \ldots, x_n, y_1, \ldots, y_m) = (q_1(\mathbf{x}, \mathbf{y}), \ldots, q_l(\mathbf{x}, \mathbf{y})) = (0, \ldots, 0),$$

where $q_i(\mathbf{x}, \mathbf{y})$ is polynomial in $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_m)$ such that

- For any given specific element $\mathbf{x}'$, we can easily solve the equation

$$\mathcal{Q}(\mathbf{x}', \mathbf{y}) = (0, \ldots, 0); \tag{4}$$

- for any given specific element $\mathbf{y}'$, we can easily solve the equation

$$\mathcal{Q}(\mathbf{x},\, \mathbf{y}') = (0, \ldots, 0), \tag{5}$$

- (usu.) Eq. 4 is linear and Eq. 5 is nonlinear but specialized to be solvable.

Now, we can build

$$\mathcal{P} = L \circ h(S(\mathbf{w}), T^{-1}(\mathbf{z})) = (0, \ldots, 0),$$

where S, T are invertible affine maps and L is linear. To verify a signature $\mathbf{w}$ with the digest $\mathbf{z}$, one checks that $\mathcal{P}(\mathbf{w}, \mathbf{z}) = 0$. If we want to use $\mathcal{P}$ to encrypt the plaintext $\mathbf{w}$, we would solve $\mathcal{P}(\mathbf{w}, \mathbf{z}) = (0, \ldots, 0)$, and find the ciphertext $\mathbf{z}$. To invert (i.e., to decrypt or more likely to sign) $\mathbf{z}$, one first calculates $\mathbf{y}' = T^{-1}(\mathbf{z})$, then plugs $\mathbf{y}'$ into the equation (5) and solve for $\mathbf{x}$. The result plaintext or signature is given by $\mathbf{w} = S^{-1}(\mathbf{x})$.

To recap, in an *implicit-form MPKC*, the public key consists of the l polynomial components of $\mathcal{P}$ and the field structure of k. The secret key mainly consists of L, S and T. Depending on the case the equation $\mathcal{Q}(X, Y) = (0, \ldots, 0)$ is either known or has parameters that is a part of the secret key. Again the basic idea is that S, T, L serve the purpose to "hide" the equation $\mathcal{Q}(\mathbf{x}, \mathbf{y}) = 0$, which otherwise could be easily solved for any $\mathbf{y}$. Mixed schemes are relatively rare, one example being Patarin's Dragon [82].

Isomorphism of Polynomials

The IP problem originated by trying to attack MPKCs by finding the secret keys. Let $\bar{F}_1, \bar{F}_2$ with

$$\bar{F}_i(x_1, \ldots, x_n) = (\bar{f}_{i1}, \ldots, \bar{f}_{im}), \tag{6}$$

be two polynomial maps from K^n to K^m. The IP problem is to look for two invertible affine linear transformations S on K^n and T over K^m (if they exist) such that

$$\bar{F}_1(x_1, \ldots, x_n) = T \circ \bar{F}_2 \circ S(x_1, \ldots, x_n). \tag{7}$$

It is clear that this problem is closely related to the attack of finding private keys for a MPKC, for example the Matsumoto-Imai cryptosystems, and was first proposed by Patarin [83], where the verification process is performed through showing the equivalence (or isomorphism) of two different maps. A simplified version is called the isomorphism of polynomials with one secret (IP1s) problem, where we only need to find the map S (if it exists), while the map T is known to be the identity map. More later in this direction are [51, 57, 68, 86, 87].

3 Examples of Multivariate PKCs

In this section, we bring to you three current MPKCs; each with special properties, advantages and disadvantages. We don't try to discuss their security in this section — that will be left until the next section.

Scheme	result	SecrKey	PublKey	KeyGen	SecrMap	PublMap
RSA-1024	1024b	128 B	320 B	2.7 sec	84 ms	2.0 ms
ECDSA-$\mathbb{F}_{2^{163}}$	320b	48 B	24 B	1.6 ms	1.9 ms	5.1 ms
PMI+$(136, 6, 18, 8)$	144b	5.5 kB	165 kB	1.1 sec	1.23 ms	0.18 ms
Rainbow $(2^8, 18, 12, 12)$	336b	24.8 kB	22.5 kB	0.3 sec	0.43 ms	0.40 ms
Rainbow $(2^4, 24, 20, 20)$	256b	91.5 kB	83 kB	1.6 sec	0.93 ms	0.74 ms
QUARTZ	128b	71.0 kB	3.9 kB	3.1 sec	11 sec	0.24 ms

Table 2. Current Multivariate PKCs Compared on a Pentium III 500

3.1 The Rainbow $(2^8, 18, 12, 12)$ Signature Scheme

We characterize a Rainbow [39] type PKC with u stages:

- The segment structure is given by a sequence $0 < v_1 < v_2 < \cdots < v_{u+1} = n$.
- For $l = 1, \ldots, u+1$, set $S_l := \{1, 2, \ldots, v_l\}$ so that $|S_l| = v_l$ and $S_0 \subset S_1 \subset \cdots \subset S_{u+1} = S$. Denote by $o_l := v_{l+1} - v_l$ and $O_l := S_{l+1} \setminus S_l$ for $l = 1 \cdots u$.
- The central map $\mathcal{Q}$ has component polynomials $y_{v_1+1} = q_{v_1+1}(\mathbf{x})$, $y_{v_1+2} = q_{v_1+2}(\mathbf{x}), \ldots, y_n = q_n(\mathbf{x})$ — *notice unusual indexing* — of the following form

$$y_k = q_k(\mathbf{x}) = \sum_{i=1}^{v_l} \sum_{j=i}^{n} \alpha_{ij}^{(k)} x_i x_j + \sum_{i<v_{l+1}} \beta_i^{(k)} x_i, \text{ if } k \in O_l := \{v_l + 1 \cdots v_{l+1}\}.$$

  In every q_k, where $k \in O_l$, there is no cross-term $x_i x_j$ where both i and j are in O_l at all. So given all the y_i with $v_l < i \leq v_{l+1}$, and all the x_j with $j \leq v_l$, we can compute $x_{v_l+1}, \ldots, x_{v_{l+1}}$.
- To expedite computations, some coefficients $(\alpha_{ij}^{(k)})$ may be fixed (e.g., set to zero), chosen at random (and included in the private key), or be interrelated in a predetermined manner.
- To invert $\mathcal{Q}$, determine (usu. at random) $x_1, \ldots x_{v_1}$, i.e., all x_k, $k \in S_1$. From the components of $\mathbf{y}$ that corresponds to the polynomials $p'_{v_1+1}, \ldots p'_{v_2}$, we obtain a set of o_1 equations in the variables x_k, $(k \in O_1)$. We may repeat the process to find all remaining variables.

For historical reasons, a Rainbow type signature scheme is said to be a TTS [107] scheme if the coefficients of $\mathcal{Q}$ are sparse. *We suggest a reference Rainbow design with the following concrete parameters: $q = 256$, $n = 42$, $m = 24$, structure sequence $(18, 12, 12)$ with no omitted central terms, expected security 2^{80} multiplications in $\mathbb{F}_{2^8}$. The size of the public key is* 22680 *bytes, the private key is* 17748 *bytes.* It's called Rainbow $(2^8, 18, 12, 12)$ for obvious reasons. A smaller version with $\mathbb{F}_{2^4}$ as the base field is also given in the table.

3.2 PMI+(136, 6, 18, 8), a Perturbed Matsumoto-Imai Plus

We may always represent the field $\mathbb{F}_{q^n}$ as an n-dimensional vector space over $\mathbb{F}_q$ via $\mathbb{F}_{q^n} \cong \mathbb{F}_q[X]/P(X)$, where P is any irreducible polynomial of degree n. Once we select P, we will then hereafter identify $\mathbb{F}_{q^n}$ with $(\mathbb{F}_q)^n$. The map induced by $\mathbf{x} \in \mathbb{F}_{q^n} \mapsto \mathbf{x}^q$ is then a linear transformation. We thus know that a map $g : \mathbf{x} \mapsto \mathbf{y} = \mathbf{x}^{q^\alpha+1}$ is homogeneously quadratic in $\mathbf{x}$. Furthermore, if and only if $\gcd(q^\alpha + 1, q^n - 1) = 1$, then this map is invertible. In fact we can find an h such that $g^{-1}(\mathbf{y}) = (\mathbf{y})^h$. This g will be termed $C^*_{q,n,\alpha}$, where the parameters may be omitted if context permits. We also write its components as $C^* = (c_1^*, c_2^*, \ldots, c_n^*)$. That is the central map of C^* or Matsumoto-Imai itself.

For Perturbed Matsumoto-Imai Plus we both perturb and add polynomials. That is, we set $q = 2$ (to make guessing easier later) and choose $\mathbf{v} = (v_1, \ldots, v_r)$, a collection of r linear forms in $\mathbf{x}$, and $\mathbf{f} = (f_1, \ldots, f_n)$, a random n-tuple of quadratic functions in $\mathbf{v}$. Further take $\mathbf{g} = (g_1, \ldots, g_a)$ be an a-tuple of random quadratic functions of $\mathbf{x}$. We define $\mathcal{Q} := (C^* + \mathbf{f}(\mathbf{v})) \| \mathbf{g}$. That is, $\mathcal{Q}$ is a map from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^{n+a}}$ whose components are given by

$$q_i(\mathbf{x}) := \begin{cases} c_i^*(\mathbf{x}) + f_i(\mathbf{v}(\mathbf{x})), & i = 1 \cdots n; \\ g_{i-n}(\mathbf{x}), & i = n+1 \cdots n+a. \end{cases}$$

How do invert $\mathcal{Q}$? That is, if $\mathbf{y} = \mathcal{Q}(\mathbf{x})$, how would we then find $\mathbf{x}$? First, we toss out the last a components, and randomly guess at the perturbation term $\mathbf{v}(\mathbf{x})$. That is, let h is the exponent that can be used to invert C^*. If $\mathbf{y}'$ is the first n components of $\mathbf{y}$, for all possible $\mathbf{b} \in \mathbb{F}_{2^r}$ we compute $\mathbf{x} = (\mathbf{y}' - \mathbf{b})^h$ and check to see whether $\mathbf{v}(\mathbf{x}) = \mathbf{b}$. Since inverting C^* is relatively slow, we can say that the perturbation made it 2^r times slower to decrypt than the corresponding C^*. *The last a components can also ensure the correctness of the ciphertext.*

It remains to give the system some concrete parameters. *At the moment, our choices are as in [32]:* $(n, r, a, \alpha) = (136, 6, 18, 8)$. *The public key size is* $n(n+1)(n+a)/2$ *bits or* 167688 *bytes; the secret key is* $(n+a)^2 + n^2 + nr(r+3)/2 + an(n+1)/2$ *bits or 26324 bytes. Design security is* 2^{83}.

3.3 The Quartz or HFEv-(2, 129, 103, 3, 4) Signature Scheme

An immediate extension of the C^* concept is Hidden Field Equations, introduced by Patarin [83]. In place of the C^* polynomial, we would substitute this :

$$\mathcal{Q} : \mathbf{x} \in (\mathbb{F}_q)^n \longmapsto \mathbf{y} = \sum_{0 \le i,j < n} a_{ij} \mathbf{x}^{q^i + q^j} + \sum_{0 \le i < n} b_i \mathbf{x}^{q^i} + c,$$

Where the coefficients are chosen more or less at random. It is also quadratic in the components of $\mathbf{x}$. Computing $\mathcal{P}^{-1}(\mathbf{y})$ by the Berlekamp Algorithm has time complexity $O(nd^2 \log d + d^3)$ where d is the maximum degree (= 129

in Quartz). Quartz uses the vinegar modification suggested by Kipnis and Patarin [64], with an auxiliary variable $\bar{\mathbf{x}}$ which occupies a subspace of small rank in $\mathbb{F}_q^n$ as follows:

$$\mathcal{Q}(\mathbf{x}, \bar{\mathbf{x}}) := \sum_{i,j} a_{ij} \mathbf{x}^{q^i+q^j} + \sum_{i,j} b_{ij} \mathbf{x}^{q^i} \bar{\mathbf{x}}^{q^j} + \sum_{i,j} \alpha_{ij} \bar{\mathbf{x}}^{q^i+q^j} + \sum_i b_i \mathbf{x}^{q^i} + \sum_i \beta_i' \bar{\mathbf{x}}^{q^i} + c. \tag{8}$$

The public key of the Quartz signature scheme uses $q = 2$, $n = 103$, dimension 4 for the $\bar{\mathbf{x}}$ subspace, and furthermore uses the *minus variant* by *removing three polynomials* from the public key. So there are 107 variables and 100 equations. The actual verification procedure in Quartz is even more complex [21], involving using the public map *four times* to avoid birthday attacks, since the design goal is a short signature (here 128 bits) and not speed. Despite this detail, the ability to solve such system still enables one to forge a signature.

The secret key of Quartz is 3kB, the public key size is $(100 \times 107 \times 108/2)$ *bits = 71kB. Design security is* 2^{80}.

3.4 Some Computational Aspects of MPKCs

Many computations of MPKCs will be conducted in $\mathbb{K} = \mathbb{F}_q$. Often q is a small power of 2 so that each element in $\mathbb{K}$ can be stored in a byte and addition represented by bitwise exclusive-or. To multiply, normally one choose a generator g in $\mathbb{K}$ such that all non-zero x can be written as $x = g^i$ (this i is also denoted $\log_g x$). We build logarithm and exponential tables and evaluate multiplications between non-zero x and y as $g^{(\log_g x + \log_g y)}$. Doing each multiplication from scratch via this method takes three table lookups and two conditional jumps and is comparatively time-consuming. This is why time-complexities are often counted in $\mathbb{K}$-multiplications. *To save time, elements of* $\mathbb{F}_{2^7}$ *or* $\mathbb{F}_{2^8}$ *that will only be used for multiplication are always stored as logarithms, for example coefficients.*

For today's highly pipelined CPUs, accessing memory is particularly expensive, and buffer memory for the most often used data (known as *L1 cache memory*) is limited to between 32kB and 256kB. Therefore complete multiplication tables of size q^2 are almost never used (except maybe when $q = 16$).

Things change when working with small microcontrollers. For example, the table exponentials is usually $2q$ $\mathbb{K}$-elements long. But for 8-bit microcontrollers, one can't have indices larger than a byte and hence evaluate $(\log_g x + \log_g y) \bmod 255$ using a single extra `ADC` (add with carry) instruction instead.

SIMD (single instruction, multiple data) is an important factor. It is very important to pack data so that one can make use of the 64- and 128-bit-wide `XOR` instructions, especially if $q = 2$ or 4 (it's called "bit-slicing", cf. [7]).

Operations in an extension $\mathbb{L} = \mathbb{F}_q^n$ *as vectors over* $\mathbb{K} = \mathbb{F}_q$ *is frequent (e.g., in big-field MPKCs). A product in* $\mathbb{L}$ *is like multiplying two degree* $< n$

polynomials over $\mathbb{F}_q$. Using schoolbook multiplication and then reducing the terms with degree $\geq n$ takes at most $2n^2$ multiplications. A more advanced method like Karatsuba takes less time. A division is a little slower than a multiplication.

It is also not a trivial issue to build keys for an MPKC. The classical way to compute the keys is interpolation [70]. In general, one select M_S, $\mathbf{c}_S$, M_T plus whatever parameters in $\mathcal{Q}$, if any. We can set

$$\mathbf{c}_T := \mathrm{M}_T \mathcal{Q}(\mathbf{c}_S),$$

which makes all the constant terms zero. Now we can evaluate $\mathcal{P}(\mathbf{w}) = T \circ \mathcal{Q} \circ S(\mathbf{w})$ for any $\mathbf{w}$. Write the Matsumoto-Imai form public key (Eq. 1) as:

$$z_k = \sum_i w_i \left[P_{ik} + Q_{ik} w_i + \sum_{j<i} R_{ijk} w_j \right]. \tag{9}$$

In $\mathbb{F}_2$, $x^2 = x$ for any x, so there is no Q_{ik} term. One also note that to evaluate the public key one need to do one $\mathbb{F}_q$ multiplication per element of the public key. Let $\mathbf{b}_i \in \mathbb{F}_q^n$ be the unit vector in the i-th axis, and for $q > 2$, we choose any $a \neq 0, 1$ and get

$$\begin{aligned} Q_{ik} &:= \left(a(a-1)\right)^{-1} \left(p_k(a\mathbf{b}_i) - a p_k(\mathbf{b}_i)\right) \\ P_{ik} &:= p_k(\mathbf{b}_i) - Q_{ik} \\ R_{ijk} &:= p_k(\mathbf{b}_i + \mathbf{b}_j) - Q_{ik} - Q_{jk} - P_{ik} - P_{jk} \end{aligned} \tag{10}$$

For $\mathbb{F}_2$, it becomes

$$\begin{aligned} P_{ik} &:= p_k(\mathbf{b}_i) \\ R_{ijk} &:= p_k(\mathbf{b}_i + \mathbf{b}_j) - P_{ik} - P_{jk} \end{aligned}$$

So key generation means invoke n^2 times the combination $T \circ \mathcal{Q} \circ S$. We can see that both S and T takes about n^2 time. If we write $\mathcal{Q}$ coefficientwise, it would take $n^3/2$ multiplications. So we see that worst case key generations takes about $n^5/2$ multiplications in $\mathbb{K} = \mathbb{F}_q$. In certain situations, it is closer to $O(n^4)$.

Let's demonstrate this for a C^* based scheme where the rate-determining mechanism is the evaluation of $C^* : \mathbf{x} \in \mathbb{L} = \mathbb{F}_{q^n} \mapsto \mathbf{y} = \mathbf{x}^{q^\alpha+1}$. There is a linear map L in $(\mathbb{F}_q)^n$ that maps $\mathbf{x} \mapsto \mathbf{x}^{q^\alpha}$. This we precompute. Evaluating $L\mathbf{x}$ takes n^2 multiplications in $\mathbb{F}_q$. Then the product in $\mathbb{L}$ is $2n^2$ multiplications max.

Other big-field variants based on ℓIC and HFE have a similar property. For single-field MPKCs where key generations takes close to $O(n^4)$, see Sec. 4.4.

4 Basic Constructions and Variations

MPKCs are built in many ways. We aim to give you the major types of constructions, maybe accent some important associated algebraic characteristics

(like this), and common variations thereof. A summary of variants is given in Table 3.

4.1 Historical Constructions

The first attempt to construct a multivariate signature [78, 79] is based on a quadratic equation

$$y = x_1^2 + \alpha x_2^2 \pmod{n}, \tag{11}$$

where $n = pq$ is an RSA modulus, the product of two large primes. To sign a message y, we need to find one of the many (about n) solutions (x_1, x_2) to Eq. 11, which is easy if we know the factorization of n. The public key is essentially the integer n and Eq. 11. Since the security relies on the factorization of n, this system is really a derivative of RSA, though it indeed initiated the idea of multivariate cryptosystems. This system was broken by Pollard and Schnorr in [89], where they found a probabilistic algorithm to solve Eq. 11 for any y without knowing the factors of n. Assuming the generalized Riemann hypothesis, a solution can be found with a time complexity of $O((\log n)^2 \log\log|k|)$ in $O(\log n)$-bit integer operations.

The idea of Diffie and Fell [52] was to build a cryptosystems using the composition of invertible linear maps and simple tame maps of the form $T(x_1, x_2) = (x_1 + g(x_2), x_2)$, where g is a polynomial.

Tame maps are easily invertible and hard to unscramble when composed with each other, however [52] used only two variables and equations; not surprisingly, the authors concluded that it appeared very difficult to build such a cryptosystem with practical value that is both secure and has a public key of practical size.

An attempt to build a true multivariate (with four variables) public key cryptosystem were also made by Matsumoto, Imai, Harashima and Miyagawa [71], where the public keys are given by quadratic polynomials. However it was soon defeated [77]. People soon realized that more than 4 variables are needed.

4.2 Triangular Constructions

Of course, the tame maps used in [52] are a special case of the "triangular" or *de Jonquières* maps from algebraic geometry, which are more generally defined by:

$$J(x_1, \ldots, x_n) = (x_1 + g_1(x_2, \ldots, x_n), \ldots, x_{n-1} + g_{n-1}(x_n), x_n), \tag{12}$$

where the g_i are arbitrary polynomial functions. We note that J can be easily inverted assuming that the g_i are known. The invertible affine linear maps over k^n together with the de Jonquières maps belong to the family of so-called tame transformations from algebraic geometry, including all transformations that arise as a composition of elements of these two types of transformations. Tame

transformations are elements of the group of automorphisms of the polynomial ring $k[x_1, \ldots, x_n]$. Elements in this automorphism group that are not tame are call wild. Given a polynomial map, it is in general very difficult to decide whether or not the map is tame, or even if there is indeed any wild map [75], a question closely related to the Jacobian conjecture. This problem was possibly solved in 2003 when [93] claims to prove that the Nagata map is wild.

The first attempt in the English literature with a clear triangular form is the Birational Permutations construction by Shamir [92]. However, triangular constructions were earlier pursued unsuccessfully in Japan under the name " sequential solution type systems" [61, 95, 97]. Their construction is actually even more general in the sense that they use rational functions instead of just polynomial. These works are not so well-known, partially because they were in Japanese.

Triangular maps are lightning fast to evaluate and to invert. However, they do have another definitive characteristic, an algebraic one, that must be accounted for. On the small end of a triangular system, so to speak, a variable is mapped to some simple function of itself. On the bigger end, one variable appears in a single equation only. The other equations involve successively more variables.

In other words, let us write the quadratic portion of the central polynomials $y_i = q_i(\mathbf{x})$ as bilinear forms, or take the symmetric matrix denoting the symmetric differential of the central polynomials as in

$$q_i(\mathbf{x} + \mathbf{b}) - q_i(\mathbf{x}) - q_i(\mathbf{b}) + q_i(0) := \mathbf{b}^T M_i \mathbf{x}, \tag{13}$$

then rank M_i increases monotonically as i increases. In fact, if $q = 2^k$, the equation dealing with x_1 always has rank zero. Furthermore, $\ker M_1 \subset \ker M_2 \cdots$. This is the *chain of kernels* as pointed out by Coppersmith *et al* [18].

This *rank* and chain relation is invariant under invertible map S. That is, consider y_i as a function of $\mathbf{w}$, the corresponding differential is $\mathbf{b}^T \left(\mathrm{M}_S^T M_i \mathrm{M}_S\right) \mathbf{x}$. For the most part, M_S is full-rank, hence $\operatorname{rank}\left(\mathrm{M}_S^T M_i \mathrm{M}_S\right) = \operatorname{rank} M_i$.

This leads to what is known as *rank attacks* based on linear algebra [18,58]. Therefore triangular/tame constructions can't be used alone. Some ways to design around this problem are *lock polynomials* (Sec. 4.6), *solvable segments* (Sec. 4.4) and *plus-minus* (Sec. 4.5).

4.3 Big-Field Families: Matsumoto-Imai (C^*) and HFE

Triangular (and Oil-and-Vinegar, and variants thereof) systems are sometimes called "single-field" or "small-field" approaches to MPKC design, in contrast to the approach taken by Matsumoto and Imai in 1988 [70]. In such "big-field" variants, the central map is really a map in a larger field $\mathbb{L}$, a degree n extension of a finite field $\mathbb{K}$. To be quite precise, we have a map $\overline{\mathcal{Q}} : \mathbb{L} \to \mathbb{L}$ that we can invert, and pick a $\mathbb{K}$-linear bijection $\phi : \mathbb{L} \to \mathbb{K}^n$. Then we have the following multivariate polynomial map, which is presumably quadratic (for efficiency):

$$\mathcal{Q} = \phi \circ \overline{\mathcal{Q}} \circ \phi^{-1}. \tag{14}$$

then, one "hide" this map $\mathcal{Q}$ by composing from both sides by two invertible affine linear maps S and T in $\mathbb{K}^n$, as in Eq. 2.

Now we briefly recap how C^* is defined earlier (cf. Sec. 3.2). Matsumoto and Imai suggest that we pick a $\mathbb{K}$ of characteristic 2 and this map $\overline{\mathcal{Q}}$

$$\overline{\mathcal{Q}} : \mathbf{x} \longmapsto \mathbf{y} = \mathbf{x}^{1+q^\alpha}, \tag{15}$$

where $\mathbf{x}$ is an element in $\mathbb{L}$, and such that $\gcd(1 + q^\alpha, q^n - 1) = 1$. The last condition ensures that the map $\overline{\mathcal{Q}}$ has an inverse, which is given by

$$\overline{\mathcal{Q}}^{-1}(\mathbf{x}) = \mathbf{x}^h, \tag{16}$$

where $h(1 + q^\alpha) = 1 \bmod (q^n - 1)$. This ensures that we can decrypt any secret message easily by this inverse. *For the rest of this chapter, we will simply identify a vector space $\mathbb{K}^k$ with larger field $\mathbb{L}$, and $\mathcal{Q}$ with $\overline{\mathcal{Q}}$, totally omitting the isomorphism ϕ from formulas. When necessary to distinguish the inner product in a vector space over $\mathbb{K}$ and the larger field $\mathbb{L}$, the former will be denoted by a dot $(\cdot)$ and the latter an asterisk $(*)$. One more important thing is that the map $\mathcal{Q}$ is always quadratic due to the linearity of the Frobenius map $\mathbf{x} \to \mathbf{x}^{q^\alpha}$.*

A significant algebraic implication of C^* and Eq. 15 is $\mathbf{y}^{q^\alpha - 1} = \mathbf{x}^{q^{2\alpha} - 1}$ or

$$\mathbf{x}\mathbf{y}^{q^\alpha} = \mathbf{x}^{q^{2\alpha}}\mathbf{y}. \tag{17}$$

This enabled Jacques Patarin [81] to cryptanalyze the original C^* with his bilinear relations (see Sec. 5.1). Though the original idea of C^* failed, it has inspired many new designs, mostly from Patarin and his collaborators (cf. Secs. 4.5 and 4.8).

The most significant of the C^* derivatives is likely HFE (Hidden Field Equations). As mentioned in Sec. 3.3, instead of using for $\mathcal{Q}$ the monomial used by C^*, we would substitute the *extended Dembowski-Ostrom polynomial map*:

$$\mathcal{Q} : \mathbf{x} \in \mathbb{L} = \mathbb{F}_q^n \longmapsto \mathbf{y} = \sum_{0 \le i \le j < r} a_{ij}\mathbf{x}^{q^i + q^j} + \sum_{0 \le i < r} b_i \mathbf{x}^{q^i} + c, \tag{18}$$

This map is in general *not* one-to-one; some kind of checksum is required to identify the inverse from one of a number of possible candidates. Inverting $\mathcal{Q}$ is equivalent to solving a univariate equation of high degree in $\mathbb{L}$. It is fairly well-studied and straightforward to implement but *not very fast*, using some version of the Berlekamp (or say Cantor-Zassenhaus) algorithm [8,14,56]. Typically, the cost of this solution is $O(nd^2 \log d + d^3)$, where d is the maximum degree of $\mathcal{Q}$.

One might conclude, then, that we should have as low d as possible, or since usually $d = 2q^r$ or $q^r + 1$, as low r as possible. It turns out not to be like

this. Just as Eq. 15 intrinsically meant that the C^* map has a rank of 2 and leads to Eq. 17 and all the known cryptanalysis of C^* related systems, Eq. 18 fundamentally is responsible for all the algebraic properties of HFE.

A key fact is that the intrinsic rank of the map is bounded by r, and usually achieves that value for randomly chosen parameters. This rank is very closely related to the complexity of current attacks [23,50]. For example, the HFE Challenge 1 solved by Faugère and Joux [50] has an intrinsic rank of 4.

HFE with a high d is unbroken, although it can be really slow to decrypt/invert. Quartz probably set a record for the slowest cryptographical algorithm when submitted to NESSIE — on a Pentium III 500MHz, it took half a minute to do a signature [since improved to 10s with better programming].

Finally, C^* and HFE each can be modified by techniques mentioned elsewhere (Plus-Minus, vinegar variables, and internal perturbation). Also related are the ℓIC system (Sec. 4.7) and probabilistic big-field based MPKCs [59]. One can safely say that all in all, C^* really spawned a lot of useful research.

4.4 Unbalanced Oil and Vinegar and Derivatives

The Oil and Vinegar and later derived unbalance Oil and Vinegar schemes [64, 80] are suitable for signatures. This construction is inspired by the idea of linearization equations (cf. Sec. 4.3). Suppose $v < n$ is an integer and $m = o = n - v$. The variables $x_1, \ldots, x_v$ are termed *vinegar* variables and $x_{v+1}, \ldots, x_n$ *oil* variables.

Take a map $\mathcal{Q} : \mathbb{K}^n \to \mathbb{K}^m$ with form $\mathbf{y} = \mathcal{Q}(\mathbf{x}) = (q_1(\mathbf{x}), \ldots, q_o(\mathbf{x}))$, where

$$q_l(\mathbf{x}) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{ij}^{(l)} x_i x_j,\ l = 1 \cdots o$$

and all coefficients are randomly chosen from the base field $\mathbb{K}$. Here we notice that there are no quadratic terms of oil variables, which means the oil variables and vinegar variables are not fully mixed (like oil and vinegar in a salad dressing) and this explains the name of this scheme.

The public map $\mathcal{P}$ is constructed as $\mathcal{P} = \mathcal{Q} \circ S$, where S is an invertible linear map. Here the change of basis is a process to "mix" fully oil and vinegar, so one can not see what is oil and what is vinegar. Note that with the pure OV and UOV constructions, we need not use a T, and it is in fact usually omitted.

The original Oil and Vinegar signature scheme has $m = o = v = n/2$. When $o < v$, it becomes the unbalance Oil and Vinegar signature scheme. The public key are $\mathcal{P} = (p_1, \ldots, p_o)$, the polynomial components of $\mathcal{P}$. The secret key consists of the linear map S and the map $\mathcal{Q}$.

Given a message $\mathbf{y} = (y_1, \ldots, y_o)$, in order to sign it, one needs to try to find a vector $\mathbf{w} = (w_1, \ldots, w_n)$ such that $\mathcal{P}(\mathbf{w}) = \mathbf{y}$. With the secret key it can be done easily. First, one guesses values for each vinegar variable $x_1, \ldots, x_v$,

and obtains a set of o linear equations with the o oil variables $x_{v+1}, \ldots, x_n$. With high probability it has a solution. If it does not have a solution, one guesses at the vinegar variables again until one finds a pre-image of a given element in $\mathbb{K}^o$. Then one applies S^{-1}. To check if $\mathbf{w}$ is indeed a legitimate signature for $\mathbf{y}$, one only needs to get the public map $\mathcal{P}$ and check if indeed $\mathcal{P}(\mathbf{w}) = \mathbf{y}$.

What algebraic property is most significant in an unbalanced Oil-and-Vinegar system? No doubt the lack of pure oil cross-terms. Equivalently, if we have an UOV structure, then the quadratic part of each component q_i in the central map from $\mathbf{x}$ to $\mathbf{y}$, when expressed as a symmetric matrix (cf. Eq. 13), looks like

$$M_i := \left[\begin{array}{ccc|ccc} \alpha_{11}^{(i)} & \cdots & \alpha_{1v}^{(i)} & \alpha_{1,v+1,}^{(i)} & \cdots & \alpha_{1n}^{(i)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{v1}^{(i)} & \cdots & \alpha_{vv}^{(i)} & \alpha_{v,v+1,}^{(i)} & \cdots & \alpha_{vn}^{(i)} \\ \hline \alpha_{v+1,1,}^{(i)} & \cdots & \alpha_{v+1,v,}^{(i)} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1}^{(i)} & \cdots & \alpha_{nv}^{(i)} & 0 & \cdots & 0 \end{array}\right] \quad \left(\text{or for short, } \left[\begin{array}{c|c} * & * \\ \hline * & 0 \end{array}\right]\right). \quad (19)$$

We should mention the fact that there are many *equivalent keys* [103]. Computing the essential part of secret keys is part of the attack of Sec. 5.5.

UOV as a Booster Stage

At some point one would be bound to ask: Suppose we have an MPKC with m equations in n variables, which is a size too small for our needs. How could we reasonably make it $m + v$ equations in $n + v$ variables? Or even $m + v'$ equations in $n + v$ variables, where $v' > v$? How can we build these "booster stages"? *The answer today is: What you can do seems limited to:*

- *Do not add extra variables, that is "Plus" (Sec. 4.5), with limited use.*
- *Solve linear equations for extra variables. That is a UOV stage, like rainbow.*
- *Solve higher-degree equations. The cost is prohibitive.*
- *Use brute-force guessing. Proposed [62, 63] and promptly broken [102].*

By stacking several layers of Unbalanced Oil-Vinegar together for an easily invertible central map, we arrive at the Rainbow-type constructions [39]. To recap (Sec. 4.4), for a u-stage Rainbow $0 < v_1 < v_2 < \cdots < v_{u+1} = n$ and

$$y_k = q_k(\mathbf{x}) = \sum_{i=1}^{v_l} \sum_{j=i}^{n} \alpha_{ij}^{(k)} x_i x_j + \sum_{i<v_{l+1}} \beta_i^{(k)} x_i, \quad \text{if } v_l < k \leq v_{l+1}. \quad (20)$$

Starting from a random choice of initial vinegar variables $x_1, \ldots, x_{v_1}$, one solve for more x_i's in sets of equations until we have all the x_i's. Note that

the components of $\mathbf{y}$ in a Rainbow-type construction is typically written to have indices $v_1 + 1, \ldots, n$. In the pure Rainbow scheme, S and T and the coefficients α and β are totally randomly chosen. The essential structure of the Rainbow instance is determined by $0 < v_1 < v_2 < \cdots < v_{u+1} = n$ or more often the "Rainbow Structure Sequence" $(v_1, o_1, o_2, \ldots, o_u)$, where $o_i := v_{i+1} - v_i$.

What is the main algebraic property of an UOV stage? First and foremost is that it is of course, a special case of UOV; however, the form of

$$o_u \text{ equations of form } \left[\begin{array}{c|c} * & * \\ \hline * & 0 \end{array}\right] \text{ following } m - o_u \text{ equations of form } \left[\begin{array}{c|c} * & 0 \\ \hline 0 & 0 \end{array}\right]$$

leads to a different attack of which the reader will be appraised later in Sec. 5.5.

Aside from attacks peculiar to UOV and Rainbow, the Rainbow-type constructions also share enough characteristics of triangular schemes, that there is the need to account for rank-based attacks (Sec. 5.4), such as the two improved attacks in [9,44]. At the moment, none of these attacks are considered essential.

Sparsity and Speed: TTS

We want the central map and its inverse be fast. However, if a booster stage can only solve linear systems for $x_{v_i+1}, \ldots, x_{v_{i+1}}$ with coefficients determined by $x_1, \ldots, x_{v_i}$, i.e., be like UOV (with $o_i = v_{i+1} - v_i$) in essence, then our hands are tied. What can we do to speed this up?

1. Setting up the system to be solved takes $o_i v_i v_{i+1}$ $\mathbb{K}$-multiplications. If we make the central map sparse, one can make this a small multiple of o_i^2.
2. Solving an $o_i \times o_i$ system in $\mathbb{K}$ takes $\sim o_i^3/3$ $\mathbb{K}$-multiplications via Gaussian elimination. For small o_i, this does not get much faster. It might be faster as an inversion in an extension field of $\mathbb{K}$. A side effect is also to make a segment sparse (with any reasonable representation of $\mathbb{F}_{q^{o_i}} \equiv \mathbb{K}^{o_i}$).

TTS (Tame Transformation Signatures) are categorically Rainbow schemes with a sparse central map, even though the term TTS came first [107].

TTS instances differ widely. The earlier ones known by that name, such as [16] are close to Triangular-minus. Later they became [107,108] much more like Rainbow with few terms. The TRMS [100] of Wang *et al* are of course also a TTS instance, although they use the larger field structure as above. Having sparse terms helps a lot: We have less to store in the private key, the private map becomes a lot quicker to execute, and even key generation is faster, since when the central map is $\mathbb{K}^n \to \mathbb{K}^m$ with sparse terms, then we can do [107]:

$$P_{ik} = \sum_{h=0}^{m-1} \left[(\mathrm{M}_T)_{kh} \left((\mathrm{M}_S)_{hi} + \sum_{p\, x_\alpha x_\beta \text{ in } q_h} p\, ((\mathrm{M}_S)_{\alpha i}(\mathbf{c}_S)_\beta + (\mathbf{c}_S)_\alpha (\mathrm{M}_S)_{\beta i}) \right) \right]$$

$$Q_{ik} = \sum_{h=0}^{m-1} \left[(\mathrm{M}_T)_{kh} \left(\sum_{p\, x_\alpha x_\beta \text{ in } q_h} p\, (\mathrm{M}_S)_{\alpha i}(\mathrm{M}_S)_{\beta i} \right) \right]$$

$$R_{ijk} = \sum_{h=0}^{m-1} \left[(\mathrm{M}_T)_{kh} \left(\sum_{p\, x_\alpha x_\beta \text{ in } q_h} p\, ((\mathrm{M}_S)_{\alpha i}(\mathrm{M}_S)_{\beta j} + (\mathrm{M}_S)_{\alpha j}(\mathrm{M}_S)_{\beta i}) \right) \right]$$

What are the drawbacks of TTS? Since TTS (TRMS) can also be viewed as Rainbow type of signature schemes, they have all the vulnerabilities of Rainbow structures. Due to their sparsity, there also exist certain extra possibilities of linear algebra and related vulnerabilities, principally UOV-type vulnerabilities such as [41].

4.5 Plus-Minus Variations

Minus and *Plus* are simple but useful ideas, earliest mentioned by Matsumoto, Patarin and Shamir (probably found independently [85, 92]).

Minus for Big-Field Schemes: SFLASH *et al*

Initially [85], several (r) equations are removed from the public keys in big-field multivariates. When inverting the public map, the legitimate users take random values for the missing variables. Minus is very suitable for signature schemes without even a performance loss, because a document need not have a unique signature.

However, for encryption this is a significant slowdown, since the missing coordinates must be guessed. To clarify a little, in theory the public map of an encryption method should injective or nearly so. If we have to guess r variables in $\mathbb{F}_q$, we effectively have q^{n+r} results, only q^n of which should represent valid ciphertexts, hence the expected number of guesses taken per decryption is q^n. Hence, decryption is slowed by that same factor of q^n.

Minus or removing some public equations makes a C^*-based system much harder to solve. SFLASH [1,22,84], a C^{*-} instance with $(q, n, r) = (2^7, 37, 11)$, was accepted as an European security standard for low-cost smart cards by the New European Schemes for Signatures, Integrity and Encryption [76].

However, in 2007, a method was discovered to defeat the SFLASH family of cryptosystems [46,47]. The key of the attack is to look at the symmetry and the invariants of the differential of the public map $\mathcal{P}$ (Sec. 5.3). If C^*-based signature schemes, it will probably need the new variant called *Projection* (Sec. 4.8).

Plus-Minus for Single-Field Schemes

In the case of *Minus* as applied to triangular constructions, one need to remove instead central equations — here, the lowest-ranked ones. Actually, removing central equations in C^* works too.

Just as *Minus* can remove the equations with smallest ranks from view and remove the problem at one end of the triangle, *Plus* is the the obverse: add random central equations to the original $\mathcal{Q}$; this masks from view the high-end of the triangle. For encryption methods, this again does not affect performance much [except for a slightly larger key]; for digital signatures there is a slowdown as the extra variables again needs to be guessed. Regardless, *Plus-Minus* variations defend against attacks that are predicated on the rank of equations.

As one might well guess, *Plus-Minus* alone does not make triangular constructions safe. Indeed, [58] discuss this in detail and concludes exactly the opposite: Triangle-Plus-Minus constructions can be broken by very straightforward attacks using simple linear algebra. Some more elaborate possibilities [9, 44, 107] are discussed in the following sections.

4.6 TTM and Related Schemes: "Lock" or Repeated Triangular

PKC's based on just triangular constructions were not pursued again until a much more complex defense against rank attacks was proposed, with the tame transformation method (TTM) of Tsong-Tsieng Moh [72].

One can see that de Jonquières maps can be upper triangular as well as lower triangular. In fact, you can arrange the indices any which way you want. Moh [72] suggested a construction where the central map $\mathcal{Q}$ is given by

$$\mathcal{Q} = J_u \circ J_l \circ I(x_1, \ldots, x_n). \tag{21}$$

Here J_u is a $\mathbb{K}^m$ upper triangular de Jonquières map and J_l is a $\mathbb{K}^m$ lower triangular de Jonquières map and the linear map I is the embedding of k^n into k^m: $I(x_1, \ldots, x_n) = (x_1, \ldots, x_n, 0, 0, \ldots, 0)$. The main achievement of such a construction is that any non-trivial linear combinations of the components of $\mathcal{Q}$ is quadratic. Moh's real trick is actually in using map I. One can see that

$$\begin{aligned} J_l \circ I(x_1, \ldots, x_n) = (x_1, x_2 + g_1(x_1), \ldots, x_n + g_{n-1}(x_1, \ldots, x_{n-1}), \\ g_n(x_1, \ldots, x_n), \ldots, g_{m-1}(x_1, \ldots, x_n)), \end{aligned}$$

which gives us the freedom to choose any g_i, $i = n, \ldots, m-1$. When decrypting, one evaluates the de Jonquières maps backwards.

The multiplitude of central polynomials of low rank present in published TTM instances [15, 72, 74] is the main source of known attacks. [74, Appendix II] gives you an idea of the polynomials of a TTM instance can look like.

A few examples of such constructions were given and a family of challenges with monetary award was set up by the US Data Security, Inc. (`www.usdsi.`

com). Shortly afterwards Courtois and Goubin [58] used the MinRank method (cf. Sec. 5.4) to attack this system. MinRank is to look for non-zero matries with minimum rank in a space of matrices; it is NP-hard in general but can be easy for special cases. Despite the inventor's claim that TTM systems are very secure from all standard attacks, Goubin-Courtois did decrypt a www.usdsi.com TTM challenge. To maintain fairness in reporting, the author claimed this to be non-conformant to his conditions of contest. He posted a new implementation of his scheme [15] soon thereafter. As mentioned above, other TTM instances had been published [73, 74] since, more complex but mostly resembling the earlier ones.

The idea of sequentially solvable equations (or stages) can also be used in conjunction with other ideas. Some of the more notable attempts are from L.-C. Wang, who had written about a series of schemes called "Tractable Rational Map Cryptosystems" (TRMC) versions 1–4. Names not withstanding, the versions of TRMC are actually quite distinct. We believe that TRMC v1 is essentially no different from early TTM [15] except for some "gratuitous incompatibility" in the bijection $\mathbf{x} \mapsto \mathbf{x}^2$. The central map of TRMCv2 [98] has a small random overdetermined block on one end (something like 7 variables and 11 equations) and the rest of the variables are determined in the triangular (tame) style. Versions 3 and 4 [99,101] use a similar trick as 3IC (cf. Sec. 4.7).

Although the TTM construction is original and very intriguing, so far existing constructions of the TTM cryptosystem and related schemes do not work for public-key encryption. In fact, most of the schemes proposed are not presented in any systematic way, and no explanation is given why and how they work. We can tell you a little about why some of these fail, however, in Secs. 5.1 and 5.4.

More sophistication is needed and we suspect that to create a successful TTM-like scheme may require deep insight from algebraic geometry.

4.7 Intermediate Fields: MFE and ℓIC

In C^* and HFE, we use a big field $\mathbb{L} = \mathbb{K}^n$, or at least the number of components in the big field is close to the number of variables. In Rainbow/TTS or similar schemes, each component is as small as the base field. It stands to reason that we can use something in between, as seen below in MFE (Medium Field Encryption) and ℓIC (ℓ-Invertible Cycles) we describe below. Both these schemes also happen to share a characteristic: the use a standard Cremona transform in algebraic geometry, where $\mathbb{L}^* := \mathbb{L}\backslash\{0\}$ for some field $\mathbb{L}$:

$$(X_1,\, X_2,\, X_3) \in (\mathbb{L}^*)^3 \longmapsto (Y_1,\, Y_2,\, Y_3) := (X_1X_2,\, X_1X_3,\, X_2X_3) \in (\mathbb{L}^*)^3 \tag{22}$$

This is a bijection for any field $\mathbb{L}$, and inverts via $X_1 := \sqrt{Y_1Y_2/Y_3}$, etc.

Medium Field Encryption

Let $\mathbb{L} = \mathbb{K}^k$ and define $\mathcal{Q} : \mathbb{L}^{12} \to \mathbb{L}^{15}$ as follows:

$$\begin{cases} Y_1 = X_1 + X_5X_8 + X_6X_7 + Q_1; \\ Y_2 = X_2 + X_9X_{12} + X_{10}X_{11} + Q_2; \\ Y_3 = X_3 + X_1X_4 + X_2X_3 + Q_3; \\ Y_4 = X_1X_5 + X_2X_7; & Y_5 = X_1X_6 + X_2X_8; \\ Y_6 = X_3X_5 + X_4X_7; & Y_7 = X_3X_6 + X_4X_8; \\ Y_8 = X_1X_9 + X_2X_{11}; & Y_9 = X_1X_{10} + X_2X_{12}; \\ Y_{10} = X_3X_9 + X_4X_{11}; & Y_{11} = X_3X_{10} + X_4X_{12}; \\ Y_{12} = X_5X_7 + X_2X_{11}; & Y_{13} = X_5X_{10} + X_7X_{12}; \\ Y_{14} = X_6X_9 + X_8X_{11}; & Y_{15} = X_6X_{10} + X_8X_{12}. \end{cases} \tag{23}$$

Here each X_i and Y_i is in $\mathbb{L} = \mathbb{K}^k$. Usu. $\mathbb{K} = \mathbb{F}_{256}$. Split $X_1, X_2, X_3, Q_1, Q_2, Q_3$ into components in $\mathbb{K}^k$, such that $q'_1 = 0$, $q'_2 = (x_1)^2$ and for $i = 3 \cdots 3k$, q'_i is a more or less a random quadratic in variables $(x_1, \ldots, x_{i-1})$.

$$X_1 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}, X_2 = \begin{bmatrix} x_{k+1} \\ x_{k+2} \\ \vdots \\ x_{2k} \end{bmatrix}, X_3 = \begin{bmatrix} x_{2k+1} \\ x_{2k+2} \\ \vdots \\ x_{3k} \end{bmatrix};$$

$$Q_1 = \begin{bmatrix} q'_1 \\ q'_2 \\ \vdots \\ q'_k \end{bmatrix}, Q_2 = \begin{bmatrix} q'_{k+1} \\ q'_{k+2} \\ \vdots \\ q'_{2k} \end{bmatrix}, Q_3 = \begin{bmatrix} q'_{2k+1} \\ q'_{2k+2} \\ \vdots \\ q'_{3k} \end{bmatrix}.$$

Decrypting MFE: Arrange $X_{1,\ldots,12}$ and $Y_{4,\ldots,15}$, into $\mathbb{L}^{2\times 2}$ matrices:

$$A_1 = \begin{bmatrix} X_1 & X_2 \\ X_3 & X_4 \end{bmatrix}, \quad A_2 = \begin{bmatrix} X_5 & X_6 \\ X_7 & X_8 \end{bmatrix}, \quad A_3 = \begin{bmatrix} X_9 & X_{10} \\ X_{11} & X_{12} \end{bmatrix};$$

$$A_1A_2 = \begin{bmatrix} Y_4 & Y_5 \\ Y_6 & Y_7 \end{bmatrix}, A_1A_3 = \begin{bmatrix} Y_8 & Y_9 \\ Y_{10} & Y_{11} \end{bmatrix}, A_2^T A_3 = \begin{bmatrix} Y_{12} & Y_{13} \\ Y_{14} & Y_{15} \end{bmatrix}. \tag{24}$$

The first step to inverting $\mathcal{Q}$ comes from Eq. 24 via simple linear algebra:

$$Y_4Y_7 - Y_5Y_6 = \det(A_1A_2) = \det A_1 \det A_2;$$

and similarly,

$$Y_8Y_{11} - Y_9Y_{10} = \det A_1 \det A_3;\ Y_{12}Y_{15} - Y_{13}Y_{14} = \det A_2 \det A_3.$$

Thus, knowing $Y_4, \ldots, Y_{15}$, we can find $\det A_1$, $\det A_2$, and $\det A_3$, provided that none of them is zero (we will need to take square roots in $\mathbb{L}$). Furthermore,

$$Y_1 = X_1 + \det A_2 + Q_1,\ Y_2 = X_2 + \det A_1 + Q_2,\ Y_3 = X_3 + \det A_3 + Q_3.$$

Therefore, having found $\det A_1, \det A_2, \det A_3$, we reduce the components of Y_1, Y_2, Y_3 to a triangular form in the x_i:

$$X_1 + Q_1 = Y_1 + \sqrt{(Y_4Y_7 + Y_5Y_6)(Y_8Y_{11} + Y_9Y_{10})(Y_{12}Y_{15} + Y_{13}Y_{14})^{-1}}$$
$$X_2 + Q_2 = Y_2 + \sqrt{(Y_4Y_7 + Y_5Y_6)(Y_8Y_{11} + Y_9Y_{10})^{-1}(Y_{12}Y_{15} + Y_{13}Y_{14})}$$
$$X_3 + Q_3 = Y_3 + \sqrt{(Y_4Y_7 + Y_5Y_6)^{-1}(Y_8Y_{11} + Y_9Y_{10})(Y_{12}Y_{15} + Y_{13}Y_{14})}$$

then we apply a second triangular step to compute X_1, X_2, and X_3 component by component. If $X_1 \neq 0$, from $\det A_1$ we can also find X_4 and complete the inversion. [101] has details on how to handle $X_1 = 0$. Of course, cases where one of the $\det A_i$ is 0 result in a decryption failure.

The main algebraic property of MFE is the central round of three matrix products. Today, everyone knows to defend against linearization relations, and MFE did in fact achieve this when they put $A_2^T A_3$ instead of $A_2 A_3$ in the center. But it does not avoid all the problems, as you can see in Sec. 5.1.

The ℓ-invertible cycle

The ℓ-invertible cycle also uses an intermediate field $\mathbb{L} = \mathbb{K}^k$ and extends C^* by using the following central map from $(\mathbb{L}^*)^\ell$ to itself:

$$\begin{aligned}\mathcal{Q} : (X_1, \ldots, X_\ell) &\mapsto (Y_1, \ldots, Y_\ell) \\ &:= (X_1X_2,\, X_2X_3, \ldots,\, X_{\ell-1}X_\ell,\, \underline{X_\ell X_1^{q^\alpha}}).\end{aligned} \tag{25}$$

For "standard 3IC", $\ell = 3$, $\alpha = 0$. Invertion in $(\mathbb{L}^*)^3$ is then easy.

$$\mathcal{Q}^{-1} : (Y_1,\, Y_2,\, Y_3) \in (\mathbb{L}^*)^3 \mapsto (\sqrt{Y_1Y_3/Y_2},\, \sqrt{Y_1Y_2/Y_3},\, \sqrt{Y_2Y_3/Y_1},). \tag{26}$$

This is $10\times$ faster computationally than the inverse of C^*. Aside from that, analysis of the properties of the 3IC map can be found in [42] — the 3IC and C^* maps has so much in common that the former can almost be viewed as a turbocharged version of the latter especially when looking at signature schemes.

For encryption schemes, "2IC" or $\ell = 2$, $q = 2$, $\alpha = 1$ is suggested.

$$\mathcal{Q} : (X_1, X_2) \mapsto (X_1X_2, X_1X_2^2), \quad \mathcal{Q}^{-1} : (Y_1, Y_2) \mapsto (Y_1/Y_2^2, Y_2/Y_1). \tag{27}$$

Again, these has so much in common with C^* that we need the same variations. In other words, we need to do $3\text{IC}^-\text{p}$ (with minus and projection) and $2\text{IC}^+\text{i}$ (with internal perturbation and plus), paralleling $C^{*-}\text{p}$ and $C^{*+}\text{i}$ (a.k.a. PMI+).

4.8 More on Variations and a Summary

Internally Perturbed

Matsumoto-Imai can produce this variation [29]: Take $\mathbf{v} = (v_1, \ldots, v_r)$ to be an r-tuple of random affine forms in the variables $\mathbf{x}$. Let $\mathbf{f} = (f_1, \ldots, f_n)$ be a random r-tuple of quadratic functions in $\mathbf{v}$. Let our new $\mathcal{Q}$ be defined by

Var.	Meaning	Slows
Plus +	extra polynomials in the central map	Slows Signatures
Minus -	remove central or public polynomials	Slows Encryption
Perturb i	internal perturbation	Slows All
Project p	Fix a central variable to be 0	Slows Signatures
Vinegar v	extra variables that can be set arbitrarily	Slows Encryption
Sparse s	make single-field central map sparse	*General Speedup*

Table 3. A Summary of Major Modifications in MPKCs, cf. [104]

$$\mathbf{x} \mapsto \mathbf{y} = (\mathbf{x})^{q^{\alpha}+1} + \mathbf{f}(\mathbf{v}(\mathbf{x}))$$

where the power operation assumes the vector space to represent a field. *The number of Patarin relations decrease quickly down to 0* as r increases. For every $\mathbf{y}$, we may find $\mathcal{Q}^{-1}(\mathbf{y})$ by guessing at $\mathbf{v}(\mathbf{x}) = \mathbf{b}$, finding a candidate $\mathbf{x} = (\mathbf{y} + \mathbf{b})^h$ and checking the initial assumption that $\mathbf{v}(\mathbf{x}) = \mathbf{b}$. Since we repeat the high going-to-the-h-th-power procedure q^r times, we are almost forced to let $q = 2$ and make r as low as possible.

We observe that there are extraneous solutions just as in HFE. Therefore, we must manufacture some redundancy in the form of a hash segment or checksum. PMI (or MIAi as classified by [104]) looked very promising, especially since there are no unbroken $\mathcal{MQ}$-encryption-schemes with any speed at that time. However, this was broken [54] via a surprising *differential cryptanalysis* (cf. Sec. 5.3). Thus, internal perturbation is usually coupled with the *plus* variation (Sec. 3.2).

Vinegar and Projection

The idea of *Vinegar* variables had been introduced earlier with UOV, and used as a defense in Quartz. The idea is to use an auxilliary variable that occupies only a small subspace of the input space (cf. Sec. 3.3). It was pointed out [38] that *Internal Perturbation* is almost exactly equal to *both Vinegar* variables *and Projection*, or fixing the input to an affine subspace. We basically set one, two or more variables of the public key to be zero to create the new public key. However, in the case of signature schemes, each projected dimension will slow down the signing process by a factor of q.

We need to tell the reader why is *Projection* useful for us. Since (Sec. 5.3) a structural attack is always by looking for an invariant or a symmetry, we should break both. Restricting to a subspace of the original $\mathbf{w}$-space breaks a symmetry. Something like the *Minus* variant destroys an invariant. Hence the use of projection by itself prevents some attacks, such as [46, 47, 55]. The differential attack against C^* (and ℓIC) derivatives uses the structure of the big field $\mathbb{L}$. Hence *projection* is expected to prevent such an attack [45].

5 Standard Attacks

Solving an MPKC directly as an MQ problem instance is usually futile; the cryptanalyst usually try to attack it as an extended IP problem, or to exploit the algebraic structures to find extra relations to make the solution easier. We hope to present enough on every approach but avoid too much detail.

5.1 Linearization Equations

A Linearization Equation is a relation between the components $\mathbf{w}$ and $\mathbf{z}$ that always holds for a given set of public keys, such that when substituted with the actual values of $\mathbf{z}$ we get an affine (linear) relation between the w_i's. Each one effectively eliminates one variable from the system.

The prime example is the direct attack against C^* found by Jacques Patarin. As mentioned in Sec. 4.3, a principal algebraic property of the C^* central map (cf. Eq. 15) is Eq. 17. Given Eq. 17. and that we know

1. $L : \mathbf{x} \mapsto \mathbf{x}^{q^{2\alpha}}$ and $L' : \mathbf{y} \mapsto \mathbf{y}^{q^{\alpha}}$ are linear maps in $\mathbb{K}^n$, and
2. $\mathbf{x} * \mathbf{y}$ in $\mathbb{L} = \mathbb{K}^n$ is bilinear, i.e., there are n matrices $\bar{\mathrm{M}}_1, \ldots, \bar{\mathrm{M}}_n$ satisfying

$$\mathbf{x} *_{\mathbb{L}} \mathbf{y} = \left(\mathbf{x}^T \cdot \bar{\mathrm{M}}_1 \cdot \mathbf{y},\, \mathbf{x}^T \cdot \bar{\mathrm{M}}_2 \cdot \mathbf{y}, \ldots, \mathbf{x}^T \cdot \bar{\mathrm{M}}_n \cdot \mathbf{y}\right).$$

We find the following bilinear relations

$$\mathbf{x}^T \cdot \mathrm{M}'_i \cdot \mathbf{y} := \mathbf{x}^T \cdot \left(L^T \bar{\mathrm{M}}_i - \bar{\mathrm{M}}_i L'\right) \cdot \mathbf{y} = 0,\ \forall i = 1 \cdots n. \tag{28}$$

After we substitute $\mathbf{w} = \mathrm{M}_S^{-1}(\mathbf{x} - \mathbf{c}_S)$ and $\mathbf{z} = \mathrm{M}_T \mathbf{y} + \mathbf{c}_T$ we get (as found by Patarin [81]) for this family of cryptosystem, due to the properties of the map $\mathcal{Q}$, the cipher satisfies n equations of the following form:

$$\sum a_{ij} z_i w_j + \sum b_i z_i + \sum c_j w_j + d = 0, \tag{29}$$

which are called *Patarin relations* or *bilinear relations*. For any C^* public key, we can compute $\mathbf{z}$ from $\mathbf{w}$, and substitute enough $(\mathbf{w}, \mathbf{z})$ pairs and solve for a_{ij}, b_i, c_j, and d. A basis for the solution space gives us all the linearization relations. If we given the ciphertext, i.e., the values of z_i, these n bilinear relations will produce linear equations satisfied by components of the the plaintext $\mathbf{w}$.

In similar systems like 3IC (Sec. 4.8), for example, Linearization Equations are also present in large numbers as in $X_1 Y_2 = X_2 Y_3 = X_3 Y_1$.

In most cases including 3IC and C^*, either there are not enough linearization relations or some relations will become redundant after the substitution of the z_j, linearizations equations does not actually find all the w_i, but it narrows down the search space by enough that we are able to find w_i easily.

Unlocking via Bilinear Relations and Others

Normally, the number of linearization equations has to be high enough such that the remaining variables can be guessed by brute force. It is shown in [36, 37] that even when the number of linearization equations is not so large, their existence can lead to defeat.

Ding and Schmidt noted that the low-rank central polynomials — often rank 2 — in currently existing implementation schemes for the TTM cryptosystem makes it possible to extend the linearization method by Patarin [81] to attack all current TTM implementation schemes (cf. Sec. 5.1). *For the Ding-Schmidt attack, the number of linearization equations is not that high, but the "lock polynomial" that defends a TTM instance against a simple rank attack is eliminated.*

HOLEs (Higher-Order Linearization Equations)

The discerning reader can figure out immediately that the linearization relation does not actually need to be linear in $\mathbf{z}$, only in $\mathbf{w}$. A Higher-Order Linearization Equation (HOLE) is a linearization relation that is higher degree in the components of $\mathbf{z}$. In particular, a SOLE (second order linearization equation) would look like

$$\sum_{i<j} a_{ijk}\, z_i z_j w_k + \sum_{i\le j} b_{ij}\, z_i z_j + \sum c_{ij}\, z_i w_j + \sum d_i\, z_i + \sum e_j\, w_j + f = 0$$

It is natural for the reader to think that this shouldn't happen very often, and it doesn't. However, the possibility that we can use such relations restricts our options when designing systems, as witness the trap that befell MFE.

Let the associated matrix of a square matrix M (replace each entry with the cofactor of that position) be M^*. Hence $(\det M)M^{-1} = M^*$, $MM^* = (\det M)1_x$, where 1_x is the identity matrix. With the same notations as Sec. 4.7, we set

$$B_1 = A_1 A_2 := \begin{bmatrix} Y_4 & Y_5 \\ Y_6 & Y_7 \end{bmatrix},\ B_2 = A_1 A_3 := \begin{bmatrix} Y_8 & Y_9 \\ Y_{10} & Y_{11} \end{bmatrix}.$$

Hence $(\det B_2 B_2^*)\, B_1 = A_3^{-1} A_2$, or $A_3 B_2^* B_1 = (\det B_2) A_2$, or (cf. [35])

$$\begin{pmatrix} X_9 & X_{10} \\ X_{11} & X_{12} \end{pmatrix} \begin{pmatrix} Y_{11} & -Y_9 \\ -Y_{10} & Y_8 \end{pmatrix} \begin{pmatrix} Y_4 & Y_5 \\ Y_6 & Y_7 \end{pmatrix} = (Y_8 Y_{11} - Y_9 Y_{10}) \begin{pmatrix} X_5 & X_6 \\ X_7 & X_8 \end{pmatrix}. \tag{30}$$

There are many ways to write down other equations that are homogeneous of degree two in the Y_i's and linear in the X_i's, but [101] showed some will lead to redundant equations. A set sure to lead to independent linear relations is

$$\begin{pmatrix} X_5 & X_6 \\ X_7 & X_8 \end{pmatrix} \begin{pmatrix} Y_{15} & -Y_{14} \\ -Y_{13} & Y_{12} \end{pmatrix} \begin{pmatrix} Y_8 & Y_{10} \\ Y_9 & Y_{11} \end{pmatrix} = (Y_{12} Y_{15} - Y_{13} Y_{14}) \begin{pmatrix} X_1 & X_2 \\ X_3 & X_4 \end{pmatrix} \tag{31}$$

That's at least $8k$ linear dependencies out of $12k$ variables. A cryptanalyst's task has gotten much easier. [101] used another trick – the fact that *squaring is linear in a char-2 field* – to get it down to $2k$ remaining variables at most and concluded that solving for the remainder is easy. *The existence of linearization relations at a higher degree when the designers certainly were trying their best to avoid such shows multivariate encryption schemes design in the triangular style to be full of potholes and very difficult without a higher algebraic breakthrough.*

5.2 Lazard-Faugère System Solvers

To mount a direct attack, we try to solve the m equations $\mathcal{P}(\mathbf{w}) = \mathbf{z}$ in the n variables $w_1, \ldots w_n$. If $m \geq n$, we are (over-)determined, which is good. If $m < n$, we are underdetermined. For most cases we can't do much more than to guess at $m - n$ variables randomly and continue with $m = n$ [20].

Today, the difficulty of solving "generic" or randomly chosen systems of nonlinear equations is generally conceded. However, it is hard to quantify exactly how non-generic a system is. Furthermore, many techniques of algebraic cryptanalysis requires system-solving methods at the end for more or less generic systems. So we must handle many instances of the $\mathcal{MQ}$ problem, where we want to solve the system $p_1 = p_2 = \cdots = p_m = 0$, where each p_i is a quadratic polynomial in $\mathbf{x} = (x_1, \ldots, x_n)$. Coefficients and variables are in the field $\mathbb{K} = \mathbb{F}_q$.

At the moment, the best known methods to solve equations are the descendants of Buchberger's algorithm [12] to compute a Gröbner basis, first investigated by Daniel Lazard's group [67]. Macaulay generalized Sylvester's matrix to multivariate polynomials [69]. The idea is to construct a matrix whose lines contain the multiples of the polynomials in the original system, the columns representing a basis of monomials up to a given degree. It was observed by D. Lazard [67] that for a large enough degree, ordering the columns according to a monomial ordering and performing row reduction without column pivoting on the matrix is equivalent to Buchberger's algorithm. Reductions to 0 correspond to lines that are linearly dependent upon the previous ones and the leading term of a polynomial is given by the leftmost nonzero entry in the corresponding line.

Lazard's idea was rediscovered in 1999 by Courtois, Klimov, Patarin, and Shamir [24] as **XL**. Courtois *et al* proposed several adjuncts [19,25,26] to XL. One tweak called XL2 merits a mention as an easy to understand precursor to $\mathbf{F_4}$. Another of these proved to be a real improvement for $\mathbf{F_4}/\mathbf{F_5}$ as well as XL. This is FXL, where F means "fixing" (guessing at) variables.

Some time *prior* to this, J. -C. Faugère had proposed a much improved Gröbner bases algorithm called $\mathbf{F_4}$ [48]. A later version, $\mathbf{F_5}$ [49], made headlines [50] when it was used to solve HFE Challenge 1 in 2002. Commercially, $\mathbf{F_4}$ is only implemented in the computer algebra system MAGMA [17].

How to solve likely non-generic systems better is an important topic that we come back to in the last section. For the rest of this paper, we will denote the monomial $x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$ by $\mathbf{x}^{\mathbf{b}}$, and its total degree $|\mathbf{b}| = b_1 + \cdots + b_n$. The set of degree-$D$-or-lower monomials is denoted $\mathcal{T} = \mathcal{T}^{(D)} = \{\mathbf{x}^{\mathbf{b}} : |\mathbf{b}| \le D\}$. $|\mathcal{T}|$ is the number of degree $\le D$ monomials and denoted $T^{(D)} = T$.

XL

Multiply each equation p_i, $i = 1 \cdots m$ by all monomials $\mathbf{x}^{\mathbf{b}} \in \mathcal{T}^{(D-2)}$. Reduce as a linear system of the equations $\mathcal{R}^{(D)} = \{\mathbf{x}^{\mathbf{b}} p_j(\mathbf{x}) = 0 : 1 \le j \le m,\ |\mathbf{b}| \le D-2\}$, with the monomials $\mathbf{x}^{\mathbf{b}} \in \mathcal{T}^{(D)}$ as independent variables. Repeat with higher D until we have a solution, a contradiction, or reduce the system to a univariate equation in some variable. The number of equations and independent equations are denoted $R^{(D)} = R = |\mathcal{R}|$ and $I^{(D)} = I = \dim(\mathrm{span}\mathcal{R})$.

If we accept solutions in arbitrary extensions of $K = \mathbb{F}_q$, then $T = \binom{n+D}{D}$ regardless of q. However, most crypto applications require solutions in $\mathbb{F}_q$ only. The above expression for T then only holds for large q, since we may identify x_i^q with x_i and cut substantially the number of monomials we need to manage. This "Reduced XL" (cf. C. Diem [27]) can lead to extreme savings compared to "Original XL," e.g., if $q = 2$, then $T = \sum_{j=0}^{D} \binom{n}{j}$.

Proposition 1 ([5,106]). *The number of monomials is* $T = [t^D] \dfrac{(1-t^q)^n}{(1-t)^{n+1}}$ *which reduces to* $\binom{n+D}{D}$ *when* q *is large. We can then find* $R = R^{(D)} = mT^{(D-2)}$.

We note that the XL of [24, 25] terminates more or less reliably when $T - I \le \min(D, q-1)$, but sparse matrix computation is only possible when $T - I \le 1$ [105]. Further, Lazard-Faugère methods work for equations of any degree [6, 106]. If $\deg(p_i) = d$, we will only multiply the equation p_i with monomials up to degree $D - d$ in generating $\mathcal{R}^{(D)}$. The principal result is:

Proposition 2 ([106, Theorem 7]). *If the equations* p_i, *with* $\deg p_i := d_i$, *and (*)* *relations* $\mathcal{R}^{(D)}$ *has no dependencies except the obvious ones generated by* $p_i p_j = p_j p_i$ *and* $p_i^q = p_i$, *then*

$$T - I = [t^D]\, G(t) = [t^D]\, \frac{(1-t^q)^n}{(1-t)^{n+1}} \prod_{j=1}^{m} \left(\frac{1-t^{d_j}}{1-t^{q\,d_j}} \right). \tag{32}$$

There is always a certain degree D_{XL} above which Eq. 32 and hence the underlined condition (*) above cannot continue to hold if the system has a solution, because the right hand side of Eq. 32 goes nonpositive. This is $D_{XL} := \min\{D : [t^D]\, G(t) \le 0\}$, called the degree of regularity for XL. If (*) holds for as long as possible (which means for degrees up to D_{XL}), we say that the system is K**-semi-regular** or q**-semi-regular** (cf. [5, 106]).

Diem proves [27] for char 0 fields, and conjectures for all K that (i) a generic system (no algebraic relationship betweem the coefficients) is K-semi-regular and (ii) if $(p_i)_{i=1\cdots m}$ are *not* K-semi-regular, I can only decrease from the Eq. 32 prediction. Most experts seem to believe the conjecture [27] that a *random* system behaves like a generic system with probability close to 1.

Corollary 1. $T-I = [t^D]\left((1-t)^{-n-1}\prod_{j=1}^{m}\left(1-t^{d_i}\right)\right)$ *for generic equations if* $D \leq \min(q, D_{XL}^{\infty})$*, where* D_{XL}^{∞} *is the degree of the lowest term with a non-positive coefficient in* $G(t) = \left((1-t)^{-n-1}\prod_{j=1}^{m}\left(1-t^{d_i}\right)\right)$.

We would note that (F)XL can only be a solver and not a true Gröbner basis method as are $\mathbf{F_4}/\mathbf{F_5}$. However, the analysis much parallels that of $\mathbf{F_4}/\mathbf{F_5}$ by Dr. Faugère et al, hence our categorical name "Lazard-Faugère" solvers.

Proposition 3 (XL with Wiedemann). *With a sparse matrix solver like the Wiedemann algorithm to solve the final matrix equation, XL has running time*

$$C_{XL} \gtrsim 3\, t\, T^2 \text{ multiplications}, \tag{33}$$

where t *is the average number of terms in an equation.*

Gröbner Bases and $\mathbf{F_4}/\mathbf{F_5}$

XL2 [25] is a tweak of XL as follows: Tag each equation with its maximal degree. Run an elimination on the system with monomials in degree-lex. In the remaining (row echelon form) system, multiply by each variable $x_1, x_2 \cdots$ *all remaining equations with the maximum tagged degree* and eliminate again. When we cannot eliminate all remaining monomials of the maximum degree, increment the operating degree and reallocate more memory.

XL+XL2 can be considered a primitive or inferior matrix form of $\mathbf{F_4}$ or $\mathbf{F_5}$ [3]. $\mathbf{F_4}$ inserts elimination between expansion stages, which compresses the number of rows that needs to be handled. $\mathbf{F_5}$ is a further refinement of $\mathbf{F_4}$. The set of equations is actually generated one by one (or the matrix row by row). In the process, an algebraic criterion is used to determine, ahead of an elimination process, whether a row will be reduced to zero or not and only the meaningful rows are retained. A complication resulting from the tagging is that the elimination must be done in a strictly ordered way. This corresponds in the matrix form to no row exchanges in a Gaussian. There are two separate degrees in $\mathbf{F_4}/\mathbf{F_5}$, an apparent "operating degree" D_{F4} and a higher intrinsic degree equal to that of the equivalent XL system. For the full power of $\mathbf{F_4}$ or $\mathbf{F_5}$, auxillary algorithms such as FGLM are needed. See [48,49] for complete details.

Proposition 4 ([5]). *If the eqs.* p_i *are* q*-semi-regular, at the operating degree*

$$D_{reg} := \min \left\{ D : [t^D] \, \frac{(1-t^q)^n}{(1-t)^n} \, \prod_{i=1}^{m} \left(\frac{1-t^{d_i}}{1-t^{qd_i}} \right) < 0 \right\}$$

both $\mathbf{F_4}$-$\mathbf{F_5}$ will terminate. Note that by specializing to a large field, we find

$$D_{reg}^{\infty} := \min \left\{ D : [t^D] \, (1-t)^{-n} \, \prod_{i=1}^{m} \left(1-t^{d_i}\right) < 0 \right\}. \tag{34}$$

If we compare this formula with Cor. 1, we see that the only difference is a substitution of n for $n+1$. In other words, we are effectively running with one fewer variable in the large field case. This explains why $\mathbf{F_4}$-$\mathbf{F_5}$ can be much faster than XL. However, the savings is smaller over small fields like $\mathbb{F}_2$, and even for large fields, removing one variable may not be enough of a savings, because the systems that we aim to solve will spawn millions of monomials (variables). Eliminating in the usual way means that we will run out of memory before time.

Proposition 5. $\mathbf{F_4}/\mathbf{F_5}$ *runs in ($\omega :=$ the "order of matrix multiplications")*

$$C_{XL} \propto c_\omega \, T^\omega \text{ multiplications.} \tag{35}$$

According to the description we received from the MAGMA project and Dr. Faugère, even though memory management is very critical, elimination is still relatively straightforward in current implementations of $\mathbf{F_4}$-$\mathbf{F_5}$, and in the process we see reasonably dense matrices, not extremely sparse ones. All said, $\mathbf{F_4}$-$\mathbf{F_5}$ *are still the most sophisticated general system-solving algorithms today. The famous complete solution of HFE challenge 1 is a run of* $\mathbf{F_5}$*, specialized and optimized for* $\mathbb{F}_2$*, which took 4 days on a 4-CPU Alpha workstation. While the HFE challenge 1 was an instance with a particularly low rank (4), it was usually argued that it should always break HFE for practical r [60].* Recently, it is disputed [40] for odd char $\mathbb{K}$. We await more developments.

5.3 Differential Attacks

Structural attack on MPKC are of two related types:

Invariants: invariants (mostly, subspaces) that can be guessed.
Symmetries: transformations that leave certain quantities unchanged and hence can be computed by a system of equations.

Of course, these two are related, given that invariants are defined according to symmetry. Previous designers sometimes neglected the importance of symmetry. In this section we present the symmetry or invariants used in the new differential attacks on the C^* family of cryptosystems as exemplified by the Differential Attacks, from the school of Stern at the École Normale Supérieure.

Attacking Internal Perturbations

The cryptanalysis of PMI was a novelty for a technique usually associated with symmetric key cryptography, since PMI was a PKC. We use the idea that for a randomly chosen $\mathbf{b}$, the probability is q^{-r} that it lies in the kernel $\mathcal{K}$ of the linear part of $\mathbf{v}$. When that happens, $\mathbf{v}(\mathbf{x}+\mathbf{b}) = \mathbf{v}(\mathbf{x})$ for any $\mathbf{x}$. Since q^{-r} is not too small, if we can distinguish between a vector $\mathbf{b} \in T^{-1}\mathcal{K}$ (back-mapped into $\mathbf{x}$-space) and $\mathbf{b} \notin T^{-1}\mathcal{K}$, we can bypass the protection of the perturbation, find our bilinear relations and accomplish the cryptanalysis.

In [54], Fouque, Granboulan and Stern built a *one-sided distinguisher* using a test on the kernel of the *polar form* or *symmetric difference* $D\mathcal{P}(\mathbf{w},\mathbf{b}) = \mathcal{P}(\mathbf{b}+\mathbf{w}) - \mathcal{P}(\mathbf{b}) - \mathcal{P}(\mathbf{w})$. We say that $t(\mathbf{b}) = 1$ if $\dim \ker_{\mathbf{w}} D\mathcal{P}(\mathbf{b},\mathbf{w}) = 2^{\gcd(n,\alpha)} - 1$, and $t(\mathbf{b}) = 0$ otherwise. If $\mathbf{b} \in \mathcal{K}$, then $t(\mathbf{b}) = 1$ with probability one, otherwise it is less than one. In fact if $\gcd(n,\alpha) > 1$, it is is an almost perfect distinguisher. If not, we can employ two other tricks. In the more important of the two, we observe $\mathcal{K}$ is a vector space, so $\Pr(t(\mathbf{b}+\mathbf{b}') = 0 | t(\mathbf{b}') = 0)$ will be relatively high if $\mathbf{b} \in \mathcal{K}$ and relatively low otherwise. We omit the gory details and refer the reader to [54] for the complete differential cryptanalysis.

This brilliantly executes a powerful attack. But there is apparently a surprisingly simple defense dating back to [85] (which introduced SFLASH). By using the "plus" (+) variant, i.e., *appending a random quadratics to* $\mathcal{P}$, enough false positives are generated to overwhelm the distinguishing test of [54]. The extra equations also serve as a distinguisher when there are extraneous solutions.

Again, we do not include all the details. Basically, the more "plus" equations, the less discriminating power of the abovementioned test. Based on empirical results of Ding and Gower [32], when $r = 6$, $a = 12$ should be sufficient, and $a = 14$ would be a rather conservative estimate for the amount of "plus" needed to mask the PMI structure.

The Skew Symmetric Transformation

The symmetry found by Stern etc. can be explained by considering the case of C^* cryptosystem. We recollect that the symmetric differential of any function G, defined formally just like in Eq. 36:

$$DG(\mathbf{a},\mathbf{x}) := G(\mathbf{x}+\mathbf{a}) - G(\mathbf{x}) - G(\mathbf{a}) + G(0).$$

is bilinear and symmetric in its variables $\mathbf{a}$ and $\mathbf{x}$. In the first version of this attack [47], we look at the the differential of the public map $\mathcal{P}$, and look for so-called skew-symmetric maps with respect to this bilinear function, namely, the linear maps M such that

$$D\mathcal{P}(\mathbf{c}, M(\mathbf{w})) + D\mathcal{P}(M(\mathbf{c}), \mathbf{w}) = 0$$

The reason that this works is that the central map $\mathcal{Q}$ and the public key, which encapsulates the vital information in the central map, unfortunately has very strong symmetry in the sense that all the differentials from these maps share some common nontrivial skew-symmetric map M. Since $\mathcal{Q}(\mathbf{x}) = \mathbf{x}^{1+q^\alpha}$, its differential is

$$D\mathcal{Q}(\mathbf{a}, \mathbf{x}) = \mathbf{a}^{q^\alpha}\mathbf{x} + \mathbf{a}\mathbf{x}^{q^\alpha}.$$

As pointed out in [47], the maps M skew-symmetric with respect to this $D\mathcal{Q}(\mathbf{a}, \mathbf{x})$ are precisely those induced from the multiplication by some element ζ satisfying the condition

$$\zeta^{q^\alpha} + \zeta = 0.$$

Clearly this skew-symmetry will hold if we translate it into $\mathbf{w}$-space. Further it can be seen that the skew-symmetry continues to hold even when we discard some components of $\mathcal{P}$. In terms of the public key, this means that if we write

$$D\mathcal{P}(\mathbf{c}, \mathbf{w}) := (\mathbf{c}^T H_1 \mathbf{w},\, \mathbf{c}^T H_2 \mathbf{w}, \ldots,\, \mathbf{c}^T H_m \mathbf{w})$$

and try to solve $M^T H_i + H_i M = 0$ for all $i = 1 \cdots m$ simultaneously, we should find just k-multiples of the identity if n and α are coprime, and a d-dimensional subspace in the space of linear maps if $d = \gcd(n, \alpha) > 1$.

For a randomly chosen map G, it should be expected that only trivial solutions $M = u1_n$, where $u \in \mathbb{K}$, will satisfy this condition. This means that there is a very strong condition on C^{*-} cryptosystems. This symmetry can be utilized to break C^{*-} systems for which $d = \gcd(n, \alpha) > 1$.

The Multiplicative Symmetry

We call the second symmetry the multiplicative symmetry, which again comes from the differential $D\mathcal{P}(\mathbf{c}, \mathbf{w})$. Let ζ be an element in the big field $\mathbb{L}$. Then we have

$$D\mathcal{Q}(\zeta \cdot a, x) + D\mathcal{Q}(a, \zeta \cdot x) = (\zeta^{q^\alpha} + \zeta) D\mathcal{Q}(a, x).$$

This is also a very strong symmetry, namely it implies that if

$$\mathrm{M}_\zeta = \mathrm{M}_S^{-1} \circ (X \mapsto \zeta X) \circ \mathrm{M}_S$$

is the linear map in $\mathbb{K}^n$ corresponding to multiplication by ζ, then

$$\mathrm{span}\{\mathrm{M}_\zeta^T H_i + H_i \mathrm{M}_\zeta : i = 1 \cdots n\} = \mathrm{span}\{H_i : i = 1 \cdots n\}.$$

I.e., the space spanned by the quadratic polynomials from the central map is invariant under the skew-symmetric action as defined above.

Clearly the public key of C^{*-} inherits some of that symmetry. Now not every skew-symmetric action by a matrix M_ζ that corresponds to an $\mathbb{L}$-multiplication that result in $\mathrm{M}_\zeta^T H_i + H_i \mathrm{M}_\zeta$ being in the span of the public-key differential matrices, because $S := \mathrm{span}\{H_i : i = 1 \cdots n - r\}$ as compared to

span$\{H_i : i = 1 \cdots n\}$ is missing r of the basis matrices. However, as the authors of [46] argued heuristically and backed up with empirical evidence, if we just pick the first three $\mathrm{M}_\zeta^T H_i + H_i \mathrm{M}_\zeta$ matrices, or any three random linear combinations of the form $\sum_{i=1}^{n-r} b_i(\mathrm{M}_\zeta^T H_i + H_i \mathrm{M}_\zeta)$ and demand that they fall in S, then

1. there is a good chance to find a nontrivial M_ζ satisfying that requirement;
2. this matrix really correspond to a multiplication by ζ in $\mathbb{L}$;
3. applying the skew-symmetric action of this M_ζ to the public-key matrices leads to other matrices in span$\{H_i : i = 1 \cdots n\}$ that is not in S.

Why *three*? There are $n(n-1)/2$ degrees of freedom in the H_i, so to form a span of $n-r$ matrices takes $n(n-3)/2 + r$ linear relations among its components ($n-r$ and not n because if we are attacking C^{*-}, we are missing r components of the public key). There are n^2 degrees of freedom in an $n \times n$ matrix U. So, if we take a random public key, it is always possible to find a U such that

$$U^T H_1 + H_1 U,\, U^T H_2 + H_2 U \in S = \mathrm{span}\{H_i : i = 1 \cdots n - r\},$$

provided that $3n > 2r$. However, if we ask that

$$U^T H_1 + H_1 U,\, U^T H_2 + H_2 U,\, U^T H_3 + H_3 U \in S,$$

there are many more conditions than degrees of freedom, hence it is unlikely to find a nontrivial solution for truly random H_i. Conversely, for a set of public keys from C^*, tests [46] shows that it almost surely eventually recovers the missing r equations and break the scheme. The only known attempted defense is [45].

5.4 Rank Attacks

We can consider *Rank attacks* to cover the UOV attacks (next section). But here we only cover attacks that specifically targets high or low rank. Let H_i be the symmetric matrix corresponding to the quadratic part of $z_i(\mathbf{w})$. Without loss of generality, we may let the fewest number of appearances of all variables in the cross-terms of the central equations be the last variable x_n appearing s times.

High Rank Attacks

Since *rank attack* often meant attacking *low rank*, some also call the High Rank attack the *Dual Rank Attack*. The High Rank Attack first appeared with [18] where Coppersmith *et al* defeated a Triangular construction.

Algorithm 1 *High Rank Attack of Goubin-Courtois and Yang-Chen [58, 107]:*

1. *Compute the differential* $\mathcal{P}(\mathbf{w}+\mathbf{c})-\mathcal{P}(\mathbf{w})-\mathcal{P}(\mathbf{c})$ *and take its* j*-th component (which is bilinear in* $\mathbf{w}$ *and* $\mathbf{c}$*) as* $\mathbf{c}^T H_j \mathbf{w}$*.* H_k *is representing the quadratic crossterms in the* k*-th polynomial of the public key. Note that the* H_i *are symmetric, so if* $\operatorname{char}\mathbb{K}=2$*,* $\mathbf{x}^T H_i \mathbf{x}=0$*. This was not made clear in [58].*
2. *Form an arbitrary linear combination* $H=\sum_i \alpha_i H_i$*. Find* $V=\ker H$*.*
3. *When* $\dim V \geq 1$*, set* $(\sum_j \lambda_j H_j)V=\{\mathbf{0}\}$ *and check if the solution set* $\hat{V}$ *of the* (λ_i) *form a subspace dimension* $m-s$*. Note: a matrix in* $K^{n\times n}$ *have at most* n *different eigenvalues, so at least* $1-(n/q)$ *of the time it does.*
4. *With probability* q^{-s} *we have found a small subspace representing* x_n*. For an UOV construction, we have found* V *corresponding to constant* $x_1 \cdots x_{v_u}$*.*

As each trial run consists of running an elimination and some testing, we can realistically do this with $\sim \left(sn^2+\frac{n^3}{6}\right)q^s$ *field multiplications, by taking linear combinations from only* $(s+1)$ *of the matrices* H_i *and hope not to get too unlucky. An upper bound is* $\left[mn^2+\frac{n^3}{6}+\frac{n}{q}(m^3/3+mn^2)\right]q^s$*.*

The above formulation of the high rank attack works for "plus"-modified Triangular systems; it is also easier to understand than the [18] formulation. Against UOV, we might possibly do even better on this attack with differentials [44].

MinRank Attack

We first describe the Goubin-Courtois version.

Algorithm 2 *[58] Let* r *be the smallest rank in linear combinations of central equations, which without loss of generality we take to be the first central equation itself. Goubin and Courtois outline how to find the smallest ranked combination (and hence break Triangle-Plus-Minus) in expected time* $O(q^{\lceil \frac{m}{n} \rceil r} m^3)$*:*

1. *Take* $P=\sum_{i=1}^{m}\lambda_i H_i$*, an undetermined linear combination of the symmetric matrices representing the homogeneous quadratic portions of the public keys.*
   A quadratic $C_{ab}x_a x_b + C_{cd}x_c x_d + \cdots$ *with all indices distinct will have a corresponding symmetric matrix with kernel* $\{\mathbf{x}: \ 0=x_a=x_b=x_c=x_d=\cdots\}$*. We will call this the kernel of the quadratic and use the shorthand* $\ker y_i$ *(or* $\ker_{\mathbf{x}} y_i$ *to specify what space). With* p *cross-terms with distinct indices, the rank of the matrix is* $2p$*. For example, in the scheme TTS/2′, the first equation is* $y_8 = x_8 + a_8 x_0 x_7 + b_8 x_1 x_6 + c_8 x_2 x_5 + d_8 x_3 x_4$*. Hence* $\ker_{\mathbf{x}} y_8 = \{\mathbf{x}: \ x_0=\cdots=x_7=0\}$ *for TTS/2′.*
2. *Guess at a random* k*-tuple* $(\mathbf{w}_1,\ldots,\mathbf{w}_k)$ *of vectors in* $\mathbb{K}^n$*, where* $k=\lceil \frac{m}{n} \rceil$*. Set* $P\mathbf{w}_1=\cdots=P\mathbf{w}_k=\mathbf{0}$ *and solve for* λ_i *via Gaussian elimination. If uniquely solvable* P *is likely the quadratic part of* y_1*, the first central equation.*

3. *Assume the matrix corresponding to y_1 has the minrank of r, then its kernel (the inverse image $H_1^{-1}(\mathbf{0})$) has dimension $n-r$, hence when we guess at $(\mathbf{w}_1, \ldots, \mathbf{w}_k)$ randomly, they have a probability of at least q^{-kr} to be all in $H_1^{-1}(\mathbf{0})$. This P is the quadratic portion of y_1 and the coefficients λ_i the row of M_T^{-1} (up to a factor).*

Yang and Chen have extended the effectiveness of this attack [107]. Such that if c mostly distinct kernels have the same r, we can accomplish our task in $1/c$ the time. In an exaggerated example, against UOV [9, 44], we can substitute r with $v_1 + 1$ if the latter is smaller.

MinRank Attacks on Big-Field Schemes

The break of HFE challenge 1 by Faugère and Joux [50], a *direct solution* of the 80 equations in 80 variables, is not the first serious attempt on HFE.

That honor belongs rather to a rank-based attack. Kipnis and Shamir suggested [66] the idea first. The attack proceeds by moving the problem back to the extension field, where all the underlying structure can be seen. This is a very natural approach if we intend to exploit the design structure of HFE in the attack. To put it simply: the minimum rank of linear combinations of the H_i should be exactly r (as in Sec. 4.3). This is the MinRank problem [13] and is in general exponential, but can be easier if r is small.

Kipnis and Shamir later suggested to take an linear combination of the H_i and take all $(r+1) \times (r+1)$ submatrices to have determinant zero. This clearly leads to a huge assortment of equations. To solve this system, they introduce an idea which they call *relinearization*, which led to the well-known XL paper [24]. It has been argued that using a Lazard-Faugère solver on this system of equations is effective [23] and equally effective as the direct attack. Sec. 5.2 has more on equation-solving.

5.5 Distilling Oil from Vinegar and Other Attacks on UOV

To a forge a signature for a UOV scheme as in Sec. 4.4, one needs to solve the equation $\mathcal{P}(\mathbf{w}) = \mathbf{y}$. When $o = v$ as with the original Oil-and-Vinegar, this turned out to be fairly easy due to the attack by Kipnis and Shamir [65].

The basic idea here is that one treats each component $y_i = p_i(\mathbf{w})$ of the public key $\mathcal{P}$ as a bilinear form. Equivalently, take their associated symmetric matrices via the *symmetric differential* as follows:

$$Dp_i(\mathbf{w}, \mathbf{c}) := p_i(\mathbf{w} + \mathbf{c}) - p_i(\mathbf{w}) - p_i(\mathbf{c}) + p_i(0) := \mathbf{c}^T H_i \mathbf{w}, \qquad (36)$$

A basic fact of OV: each matrix M_i (cf. Eq. 13) is in the rough form form of $\begin{bmatrix} * & * \\ * & 0 \end{bmatrix}$ *but not the matrices H_i. This reduces a cryptanalysis to the algebraic problem of finding a basis change for a set of bilinear forms into a common form.*

The problem is interesting enough that we will sketch you one solution. Recall that $v = o = n/2 = m$. We will call the vectors $\mathbf{x}$ that have all vinegar coordinates $x_1, \ldots, x_v$ equal to zero, to be the Oil Space $\mathcal{O}$, i.e. the collection of $\mathbf{x}$-vectors looking like $\begin{bmatrix} 0 \\ * \end{bmatrix}$, and similarly a $\mathbf{x}$-vector in the Vinegar Space $\mathcal{V}$ has all oil coordinates $x_{v+1}, \ldots, x_n$ equal to zero and looks like $\begin{bmatrix} * \\ 0 \end{bmatrix}$. Clearly, if each M_i is nonsingular, we have

$$\begin{bmatrix} * & * \\ * & 0 \end{bmatrix} \begin{bmatrix} 0 \\ * \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}, \text{ or } M_i \mathcal{O} = \mathcal{V} \; \forall i.$$

Hence, we have $\left(M_j^{-1} M_i\right) \mathcal{O} = \mathcal{O}$. It then follows that

$$\left(H_j^{-1} H_i\right) (S^{-1}\mathcal{O}) = (S^{-1}\mathcal{O}),$$

which in English states that any $H_j^{-1} H_i$ has the common invariant subspace (finding which is a known problem) of $S^{-1}\mathcal{O}$, or the oil subspace expressed in $\mathbf{w}$ coordinate form. Knowing $S^{-1}\mathcal{O}$ is sufficient to find an equivalent form for S. Later it was shown by Kipnis *et al* [64] that the same argument works if $v < o$; even if $v > o$ it can be done in time directly proportional to q^{v-o}, and hence $v - o$ cannot be too small. When there are two or three times more vinegar variables than oil variables the method appears to be secure, despite the claims of [11].

Reconciliation

There is more than the Kipnis-Shamir attack to transform the public maps of an UOV scheme to the Eq. 13 Common form. We could instead [44] attempts to find a sequence of change of basis that let us invert the public map, as in an improved brute force attack.

First, no matter what M_T is, it won't change the basic shape, so we let T be the identity map for the moment. What can S be like? Suppose we pick M_S as totally random, most often (see below) it decompose to

$$\mathrm{M}_S := \begin{bmatrix} *_{v\times v} & *_{v\times o} \\ *_{o\times v} & *_{o\times o} \end{bmatrix} = \begin{bmatrix} 1_{v\times v} & *_{v\times o} \\ 0_{o\times v} & 1_{o\times o} \end{bmatrix} \begin{bmatrix} *_{v\times v} & 0_{v\times o} \\ *_{o\times v} & *_{o\times o} \end{bmatrix} \tag{37}$$

where 1 means identity matrix, 0 means just zeros and $*$ means random or anything. In fact, this decomposition always hold unless the lower-right $o \times o$ submatrix is singular. It should be clear that the $\begin{bmatrix} *_{v\times v} & 0_{v\times o} \\ *_{o\times v} & *_{o\times o} \end{bmatrix}$ portion of M_S, as a coordinate change leaves the M_i's with the same shape. I.e., if we can find the correct $\begin{bmatrix} 1_{v\times v} & *_{v\times o} \\ 0_{o\times v} & 1_{o\times o} \end{bmatrix}$ portion and perform the basis change in reverse, we will again make the resulting public map into the same form (all zeroes

on the lower right) and be easily inverted. Hence, no more security at all. More about this phenomenon ("equivalent keys") in MPKCs can be found in, say, [103].

Let this essential part of M_S to be recreated be P. I.e., the linear transformation $\mathbf{w} \mapsto \mathbf{x} = P\mathbf{w}$ create all zeroes on the lower right. We can decompose this P into a product of $P := P_{v+1}P_{v+2}\cdots P_n$, where each matrix look like

$$P_n = 1_n + \left[\begin{array}{ccc|c} 0 & \cdots & 0 & a_1 \\ 0 & \cdots & 0 & a_2 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_v \\ \hline 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{array}\right]; \quad P_{n-1} = 1_n + \left[\begin{array}{ccc|c|c} 0 & \cdots & 0 & a'_1 & 0 \\ 0 & \cdots & 0 & a'_2 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & a'_v & 0 \\ \hline 0 & \cdots & 0 & 0 & 0 \\ \hline \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 \end{array}\right]; \cdots$$

Indeed, the multiplication is actually commutative among the various P_i's. *Let us then start with the differential matrices H_i and simultaneously transform them to make their lower-right corner a square of* 0*'s using exactly such P_i's.*

Algorithm 3 (UOV Reconciliation Attack) *The following is an attack on a UOV scheme with o oil and $v = n - o$ vinegar variables (which has the smaller indices):*

1. *Perform basis change $w_i := w'_i - \lambda_i w'_n$ for $i = 1 \cdots v$, $w_i = w'_i$ for $i = v+1 \cdots n$. Evaluate $\mathbf{z}$ in $\mathbf{w}'$.*
2. *Let all coefficients of $(w'_n)^2$ be zero and solve for the λ_i. We may use any method such as $\mathbf{F_4}/\mathbf{F_5}$ or FXL. There will be m equations in v unknowns.*
3. *Repeat the process to find P_{n-1}. Now we set $w'_i := w''_i - \lambda_i w''_{n-1}$ for $i = 1 \cdots v$, and set every $(w''_{n-1})^2$ and $w''_n w''_{n-1}$ term to zero (i.e., more equations in the system) after making the substitution. This time it should be faster since we solve $2m$ equations in v unknowns.*
4. *Continue in this fashion for $P_{n-2}, \ldots, P_{v+1}$ (easier, even more equations).*

In the state-of-the-art system-solving today, we can expect the complexity to be determined in solving the initial system. Hence, if $v < m$, solving m equations in v variables will be easier than m equations in n equations.

Proposition 6. *The Reconciliation Attack fails with probability* $\approx \frac{1}{q-1}$.

Proof (Sketch). Provided that lower-right $o \times o$ submatrix of M_S is nonsingular, we can see that the construction of P_n will eliminate the quadratic term in the last variable. P_{n-1} will eliminate all quadratic terms in the last two variables, and so on, and each sequential construction will not disturb the structure built by the prior transformations. The number of nonsingular $k \times k$ matrices over $\mathbb{F}_q$ is $(q^k - 1)(q^k - q)(q^k - q^2)\cdots(q^k - q^{k-1})$, because the first row has 1 possibility to be zero, the second row q possibilities to be a

multiple of the first, the third row q^2 possibilities to be dependent on the first two, etc., so the chance that the above attack works is roughly

$$\left(1-\frac{1}{q}\right)\left(1-\frac{1}{q^2}\right)\cdots\left(1-\frac{1}{q^k}\right) > 1-\left(\frac{1}{q}+\frac{1}{q^2}+\cdots+\frac{1}{q^k}\right) > 1-\frac{1}{q-1}.$$

Attacking Rainbow and TTS

Alg. 3 is just a *unbalanced oil and vinegar attack*. Rainbow systems have multiple layers (cf. 4.4). So the symmetric matrix M_i for the quadratic part of a Rainbow central polynomial q_i looks more like

$$M_i = \left[\begin{array}{ccc|ccc} \alpha_{11}^{(i)} & \cdots & \alpha_{1v}^{(i)} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{v1}^{(i)} & \cdots & \alpha_{vv}^{(i)} & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right] \text{ if } i \le m-o; \tag{38}$$

$$= \left[\begin{array}{ccc|ccc} \alpha_{11}^{(i)} & \cdots & \alpha_{1v}^{(i)} & \alpha_{1,v+1,}^{(i)} & \cdots & \alpha_{1n}^{(i)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{v1}^{(i)} & \cdots & \alpha_{vv}^{(i)} & \alpha_{v,v+1,}^{(i)} & \cdots & \alpha_{vn}^{(i)} \\ \hline \alpha_{v+1,1,}^{(i)} & \cdots & \alpha_{v+1,v,}^{(i)} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1}^{(i)} & \cdots & \alpha_{nv}^{(i)} & 0 & \cdots & 0 \end{array}\right] \text{ if } i > m-o.$$

I.e., the last o equations looks like Eq. 19, but the initial $m-o$ equations only have non-zero entries in the upperleft submatrix. The attack below exploits this. Actually it applies to all final schemes with a final UOV booster stage, since we do not use in the attack the property that the first $m-o$ usually are UOV matrices themselves, i.e., has a block of zeros on the lower right.

At this point, we should no longer consider T as the identity. Let us think about what the matrix M_T does in Rainbow. At the moment that we distill the P_n portion out, $m-o$ of the new M_i's should show a zero last column. *However we don't;* M_T *mixes the* M_i*'s together so that they in fact don't – we will see most of the time only the lower right entry as zero. But if we take any* $o+1$ *of those last columns, there will be a non-trivial linear dependency. We can verify that by setting one of those columns as the linear combination as the other* o*, the resulting equations are still quadratic!*

[This idea was first mentioned by Y.-H. Hu in a private discussion.]

Algorithm 4 (Rainbow Band Separation) *The Reconciliation attack may be extended for a Rainbow scheme where the final stage has* o *oil and* $v = n-o$ *vinegar variables (which has the smaller indices):*

1. *Perform basis change* $w_i := w'_i - \lambda_i w'_n$ *for* $i = 1 \cdots v$, $w_i = w'_i$ *for* $i = v+1 \cdots n$. *Evaluate* $\mathbf{z}$ *in* $\mathbf{w}'$.
2. *Find* m *equations by setting all coefficients of* $(w'_n)^2$ *to be zero; there are* v *variables in the* λ_i*'s.*
3. *Set all cross-terms involving* w'_n *in* $\mathbf{z}_1 - \sigma_1^{(1)}\mathbf{z}_{v+1} - \sigma_2^{(1)}\mathbf{z}_{v+2} - \cdots - \sigma_o^{(1)}\mathbf{z}_m$ *to be zero and find* $n-1$ *more equations. Note that* $(w'_n)^2$ *terms are assumed gone already, so we can no longer get a useful equation.*
4. *Solve* $m+n-1$ *quadratic equations in* $o+v=n$ *unknowns. We may use any method (e.g.,* $\mathbf{F_4}$ *or XL).*
5. *Repeat the process to find* P_{n-1}. *Now set* $w'_i := w''_i - \lambda_i w''_{n-1}$ *for* $i = 1 \cdots v$, *and set every* $(w''_{n-1})^2$ *and* $w''_n w''_{n-1}$ *term to zero after making the substitution. Also set* $\mathbf{z}_2 - \sigma_1^{(2)}\mathbf{z}_{v+1} - \sigma_2^{(2)}\mathbf{z}_{v+2} - \cdots - \sigma_o^{(2)}\mathbf{z}_m$ *to have a zero second-to-last column. This time there are* $2m+n-2$ *equations in* n *unknowns.*
6. *Continue similarly to find* $P_{n-2}, \ldots, P_{v+1}$ *(now easier with more equations).*

To repeat, **the Alg. 4 attack works for all constructions with a UOV final stage, including all Rainbow and TTS constructions.** That explains why the current proposed parameters of Rainbow [44] looks like those in Sec. 3.1.

6 The Future

In the last ten years, MPKCs have seen very active and fast developments, producing many interesting new ideas, tools and constructions in both theory and its applications. Due to the consideration of quantum computer threat and the potential of its applications in ubiquitous computing devices, we foresee that the research in MPKCs will move on to the next level in the next decade. Here, we would like present some of our thoughts on the future of the research in multivariate public key cryptography.

6.1 Construction of MPKCs

The real breakthrough of MPKCs should be attributed to the work by Matsumoto and Imai in 1988 [70], a fundamental catalyst. The new idea of Matsumoto and Imai should be called the "Big Field" construction, where we build first a map in a degree n extension field (Big Field) $\mathbb{L}$ over a small finite field $\mathbb{K}$, then move it down to a vector space over the small finite field with the identification map $\phi : \mathbb{L} \longrightarrow \mathbb{K}^n$, the standard $\mathbb{K}$-linear isomorphism between $\mathbb{L}$ and $\mathbb{K}^n$.

Great efforts are still being devoted to developing MPKCs using this idea [101], [42], [35] and [55]. This is also the idea behind the new Zhuang-Zi algorithm [33], where we lift the problem of solving a set of multivariate

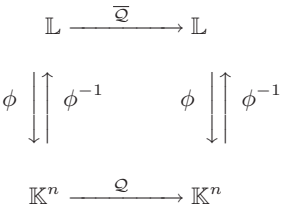

Fig. 1. Identifying maps on a $\mathbb{K}$-vector space with those on extension fields $\mathbb{L}/\mathbb{K}$.

polynomial equations over a small finite field to solving a set of single variable equations over an extension field. Recently, a new idea of reviving HFE using field of odd characteristics was proposed [40].

What we have seen is that what really drives the development of the designs in MPKCs are indeed new mathematical ideas that bring new mathematical structures and insights in the construction of MPKCs. We believe the mathematical idea we have used are just some of the very basic ideas developed in mathematics and there is great potential in pushing this idea further using some of the more sophisticated mathematical constructions in algebraic geometry. Therefore, there is great potential to study and search for further mathematical ideas and structures that could be used to construct MPKCs. One particularly interesting problem would be to make the TTM cryptosystems work where a systematic approach should be established. This definitely demands some deep insights and the usage of some intrinsic combinatorial structures from algebraic geometry.

From the point of view of practical applications, there are two critical problems that deserve more attention in designing new MPKCs. The first one is the problem of the public key size. For a MPKC with m polynomials and n variables, the public key size normally has $m(n+2)(n+1)/2$ terms, where m is at least 25 and n is at least 30. Compared with all other public key cryptosystems, for example RSA, one disadvantage is that in general a MPKC has a relatively large public key (tens of Kbytes). This is not a problem from the point view of modern computers, such as the PCs we use, but it could be a problem if we want to use it for small devices with limited memory resources. This would also be a problem if a device with limited communication abilities needs to send the public key for each transaction, for example in the case of authentication.

One idea is to do something like in [96], where a cryptosystem is built with a very small number of variables (5) but with a higher degree (4) over a much bigger base field (32 bits). In other words, we can try high degree constructions with fewer variables but over a much bigger field. In general, any new idea for how to reduce the public key size or in how to manage it in practical applications would be really appreciated.

A second idea is that of using sparse polynomials constructions. The first explicit usage of such constructions should be attributed to the works of Yang

and Chen [16]. But some of the early such constructions were broken exactly because of the usage of sparse polynomials [41], which brought unexpected weakness to the system. However, we believe that the idea of using sparse polynomials is an excellent idea, especially from the point view of practical applications. From the theoretical point of view, one critical question that needs to be addressed carefully is that of whether or not the use of specific sparse polynomials has any substantial impact on the security of the given cryptosystem. The answer to this problem will help us to establish the principles for how we should choose sparse polynomials that do not affect the security of the given cryptosystem. An unexpected consequence of answering this problem is that it might also shed some light on the problem mentioned above about reducing the size of the public key.

6.2 Attack on MPKCs and Provable Security

Several major methods have been developed to attack the MPKCs. They can be roughly grouped into the following two categories.

- **Structure-based** – These attacks rely solely on the specific structures of the corresponding MPKC. Here, we may use several methods, for example, the rank attack, the invariant subspace attack, the differential attack, the extension field structure attack, the low degree inverse, and others.
- **General Attack** – This attack uses the general method of solving a set of multivariate polynomial equations, for example using the Gröbner basis method, including the Buchberger algorithm, its improvements (such as F_4 and F_5), the XL algorithm, and the new Zhuang-Zi algorithm.

Of course, we may also combine both methods to attack a specific MPKC.

It is clear that for a given multivariate cryptosystem, we should first try the general attack and then we may then look for methods that use the weaknesses of the underlying structure.

Though a lot of work has been done in analyzing the efficiency of different attacks, we still do not fully understand the full potential or the limitations of some of the attack algorithms, such as the MinRank algorithm, Gröbner basis algorithms, the XL algorithm, and the new Zhuang-Zi algorithm. For example, we still know very little about how these general attacks will work on the internal perturbation type systems such as PMI+ [32, 34], though we do have some experimental data to give us some ideas about how things work. Another interesting question is to find out exactly why and how the improved Gröbner basis algorithms like F_4 and F_5 work on HFE and its simple variants with low parameter D [49, 50]. The question is why the hidden structure of HFE can be discovered by these algorithms.

Much work is still needed to understand both the theory and practice of how efficiently general attack algorithms work and how to implement them efficiently. From the theoretical point of view, to answer these problems, the

foundation again lies in modern algebraic geometry as in [27]. One critical step would be to prove the maximum rank conjecture pointed out in [27], which is currently the theoretical basis used to estimate the complexity of the XL algorithm and the F_4 and F_5 algorithms for example. Another interesting problem is to mathematically prove some of the commonly used complexity estimate formulas in [105].

One more important problem we would like to emphasize is the efficient implementation of general algorithms. Even for the same algorithm, the efficiency of various implementations can be substantially different. For example, one critical problem in implementing F_4 or F_5, or the XL type algorithms, is that the programs tend to use a large amount of memory for any nontrivial problem. Often the computation fails not because of time constraints but because the program runs out of memory. Therefore, efficient implementations of these algorithms with good memory management should be studied and tested carefully.

Chen, Yang, and Chen [109] developed a new XL implementation with a Wiedemann solver that is probably as close to optimal as might be possible. They showed that in a few cases the simple FXL algorithm can even outperform the more sophisticated F_4 and F_5 algorithms. More new ideas of improving the algorithms, such as using the concept of mutant [30, 31], are also being developed. In general, any new idea or technique in implementing these algorithms efficiently could have very serious practical implications.

In order to convince industry to actually use MPKCs in practical applications, the first and the most important problem is the concern of security. Industry must be convinced that MPKCs are indeed secure. A good answer to this problem is to prove that a given MPKC is indeed secure with some reasonable theoretical assumptions; that is, we need to solve the problem of provable security of MPKCs. From this point of view, the different approaches taken in attacking MPKCs present a very serious problem in terms of provable security. Many people have spent a considerable amount of time thinking about this problem, but there are still no substantial results in this area. One possible approach should be from the point view of algebraic geometry; that is, we need to study further all the different attacks and somehow put them into one theoretical framework using some (maybe new) abstract notion. This would allow us to formulate some reasonable theoretical assumptions, which is the foundation of any type of provable security. This is likely a very hard problem.

6.3 Practical Applications

Currently, a very popular notion in the computing world is the phrase "ubiquitous computing." This phrase describes a world where computing in some form is virtually everywhere, usually in the form of some small computing device such as RFID, wireless sensors, PDA, and others. Some of these devices often have very limited computing power, batteries, memory capacity,

and communication capacity. Still, because of its ever growing importance in our daily lives, the security of such a system will become an increasingly important concern. It is clear that public key cryptosystems like RSA cannot be used in these settings due to the complexity of the computations.

In some way, MPKCs may provide an alternative in this area. In particular, there are many alternative multivariate signature schemes such as Rainbow, TTS and TRMC. Recently [4, 110] it is shown that systems like TTS and Rainbow have great potential for application in small computing devices. Due to its high efficiency, a very important direction in application of MPKCs is to seek new applications where the classical public key cryptosystems like RSA cannot work satisfactorily. This will also likely be the area where MPKCs will find a real impact in practical applications.

6.4 Broad Connections

As MPKCs develops, it starts to interact more and more with other topics, one example is the algebraic attacks. Algebraic attacks are a very popular research topic in attacking symmetric block ciphers like AES [26] and stream ciphers [2] and analyzing hash functions [94]. We would like to point out that the origin of such an idea is actually from MPKCs, and in particular Patarin's linearization equation attack method. From recent developments we see that there is a trend that the research of MPKCs will interact very closely with that in symmetric ciphers and stream ciphers. We believe some of the new ideas we have seen in MPKCs will have much more broad applications in the area of algebraic attacks. The idea of multivariate construction was also applied to the symmetric constructions. Recently, new methods had been proposed to build secure hash functions using random quadratic maps [43] [10]. These constructions are very simple and therefore easy to study. They may also have very good property in terms of provable security. Similar ideas may have further applications in designing stream ciphers and block ciphers. We foresee that the theory of functions on a space over a finite field (multivariate functions) will play an increasingly important role in the unification of the research in all these related areas.

It is evident that the research in MPKCs has already presented new mathematical challenges that demand new mathematical tools and ideas. In the future, we expect to see a mutually beneficial interaction between MPKCs and algebraic geometry to grow rapidly. We further believe that MPKCs will provide excellent motivation and critical problems in the development of the theory of functions over finite fields. There is no doubt that the area of MPKC will welcome the new mathematical tools and insights that will be critical for its future development.

References

1. Akkar, M.L., Courtois, N., Duteuil, R., and Goubin, L.: A fast and secure implementation of Sflash. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003*, volume 2567 of *LNCS*, pages 267–278. Springer (2003).
2. Armknecht, F. and Krause, M.: Algrebraic attacks on combiners with memory. In *Crypto 2003, August 17-21, Santa Barbara, CA, USA*, volume 2729 of *LNCS*, pages 162–176. Springer (2003).
3. Ars, G., Faugère, J.C., Imai, H., Kawazoe, M., and Sugita, M.: Comparison between XL and Gröbner Basis algorithms. In AsiaCrypt [88], pages 338–353.
4. Balasubramanian, S., Bogdanov, A., Rupp, A., Ding, J., and Carter, H.W.: Fast multivariate signature generation in hardware: The case of rainbow. Poster Session, FCCM 2008.
5. Bardet, M., Faugère, J.C., and Salvy, B.: On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74 (2004). Previously INRIA report RR-5049.
6. Bardet, M., Faugère, J.C., Salvy, B., and Yang, B.Y.: Asymptotic expansion of the degree of regularity for semi-regular systems of equations. In P. Gianni, editor, *MEGA 2005 Sardinia (Italy)* (2005).
7. Berbain, C., Billet, O., and Gilbert, H.: Efficient implementations of multivariate quadratic systems. In *Proc. SAC 2006*. Springer (in press, dated 2006-09-15).
8. Berlekamp, E.R.: Factoring polynomials over finite fields. *Bell Systems Technical Journal*, 46:1853–1859 (1967). Republished in: Elwyn R. Berlekamp. "Algebraic Coding Theory". McGraw Hill, 1968.
9. Billet, O. and Gilbert, H.: Cryptanalysis of rainbow. In *Security and Cryptography for Networks*, volume 4116 of *LNCS*, pages 336–347. Springer (2006).
10. Billet, O., Robshaw, M.J.B., and Peyrin, T.: On building hash functions from multivariate quadratic equations. In J. Pieprzyk, H. Ghodosi, and E. Dawson, editors, *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 82–95. Springer (2007). ISBN 978-3-540-73457-4.
11. Braeken, A., Wolf, C., and Preneel, B.: A study of the security of Unbalanced Oil and Vinegar signature schemes. In *The Cryptographer's Track at RSA Conference 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 29–43. Alfred J. Menezes, ed., Springer (2005). Also at `http://eprint.iacr.org/2004/222/`.
12. Buchberger, B.: *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. Ph.D. thesis, Innsbruck (1965).
13. Buss, J.F., Frandsen, G.S., and Shallit, J.O.: The computational complexity of some problems of linear algebra. Research Series RS-96-33, BRICS, Department of Computer Science, University of Aarhus (1996). `http://www.brics.dk/RS/96/33/`, 39 pages.
14. Cantor, D.G. and Zassenhaus, H.: A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(587–592) (1981).
15. Chen, J.M. and Moh, T.T.: On the Goubin-Courtois attack on TTM. Cryptology ePrint Archive (2001). Http://eprint.iacr.org/2001/072.

16. Chen, J.M. and Yang, B.Y.: A more secure and efficacious TTS signature scheme. In J.I. Lim and D.H. Lee, editors, *ICISC*, volume 2971 of *LNCS*, pages 320–338. Springer (2003). ISBN 3-540-21376-7.
17. Computational Algebra Group, University of Sydney: *The MAGMA Computational Algebra System for Algebra, Number Theory and Geometry.* `http://magma.maths.usyd.edu.au/magma/`.
18. Coppersmith, D., Stern, J., and Vaudenay, S.: The security of the birational permutation signature schemes. *Journal of Cryptology*, 10:207–221 (1997).
19. Courtois, N.: Algebraic attacks over $GF(2^k)$, application to HFE challenge 2 and Sflash-v2. In PKC [53], pages 201–217. ISBN 3-540-21018-0.
20. Courtois, N., Goubin, L., Meier, W., and Tacier, J.D.: Solving underdefined systems of multivariate quadratic equations. In *Public Key Cryptography — PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 211–227. David Naccache and Pascal Paillier, editors, Springer (2002).
21. Courtois, N., Goubin, L., and Patarin, J.: *Quartz: Primitive specification (second revised version)* (2001). `https://www.cosic.esat.kuleuven.be/nessie` Submissions, Quartz, 18 pages.
22. Courtois, N., Goubin, L., and Patarin, J.: *Sflash: Primitive specification (second revised version)* (2002). `https://www.cosic.esat.kuleuven.be/nessie`, Submissions, Sflash, 11 pages.
23. Courtois, N.T., Daum, M., and Felke, P.: On the security of HFE, HFEv- and Quartz. In *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 337–350. Y. Desmedt, ed., Springer (2002). `http://eprint.iacr.org/2002/138`.
24. Courtois, N.T., Klimov, A., Patarin, J., and Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Bart Preneel, ed., Springer (2000). Extended Version: `http://www.minrank.org/xlfull.pdf`.
25. Courtois, N.T. and Patarin, J.: About the XL algorithm over gf(2). In *The Cryptographer's Track at RSA Conference 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157. Springer (2003).
26. Courtois, N.T. and Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Yuliang Zheng, ed., Springer (2002).
27. Diem, C.: The XL-algorithm and a conjecture from commutative algebra. In AsiaCrypt [88], pages 323–337. ISBN 3-540-23975-8.
28. Diffie, W. and Hellman, M.E.: New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654 (1976). ISSN 0018-9448.
29. Ding, J.: A new variant of the Matsumoto-Imai cryptosystem through perturbation. In PKC [53], pages 305–318.
30. Ding, J., Buchmann, J., Mohamed, M.S.E., Mohamed, W.S.A.E., and Weinmann, R.P.: Mutant xl. accepted for the First International Conference on Symbolic Computation and Cryptography, SCC 2008.
31. Ding, J., Carbarcas, D., Schmidt, D., Buchmann, J., and Tohaneanu, S.: Mutant groebner basis algorithms. accepted for the First International Conference on Symbolic Computation and Cryptography, SCC 2008.

32. Ding, J. and Gower, J.: Inoculating multivariate schemes against differential attacks. In *PKC*, volume 3958 of *LNCS*. Springer (2006). Also available at `http://eprint.iacr.org/2005/255`.
33. Ding, J., Gower, J., and Schmidt, D.: Zhuang-Zi: A new algorithm for solving multivariate polynomial equations over a finite field. Cryptology ePrint Archive, Report 2006/038 (2006). `http://eprint.iacr.org/`, 6 pages.
34. Ding, J., Gower, J.E., Schmidt, D., Wolf, C., and Yin, Z.: Complexity estimates for the F_4 attack on the perturbed Matsumoto-Imai cryptosystem. In *CCC*, volume 3796 of *LNCS*, pages 262–277. Springer (2005).
35. Ding, J., Hu, L., Nie, X., Li, J., and Wagner, J.: High order linearization equation (hole) attack on multivariate public key cryptosystems. In *PKC*, volume 4450 of *LNCS*, pages 230–247. Springer (2007).
36. Ding, J. and Schmidt, D.: A common defect of the TTM cryptosystem. In *Proceedings of the technical track of the ACNS'03, ICISA Press*, pages 68–78 (2003). Http://eprint.iacr.org/2003/085.
37. Ding, J. and Schmidt, D.: The new TTM implementation is not secure. In K. Feng, H. Niederreiter, and C. Xing, editors, *Workshop on Coding Cryptography and Combinatorics, CCC2003 Huangshan (China)*, volume 23 of *Progress in Computer Science and Applied Logic*, pages 113–128. Birkhauser Verlag (2004).
38. Ding, J. and Schmidt, D.: Cryptanalysis of HFEv and internal perturbation of HFE. In PKC [91], pages 288–301.
39. Ding, J. and Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In *Conference on Applied Cryptography and Network Security — ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175. Springer (2005).
40. Ding, J., Schmidt, D., and Werner, F.: Algebraic attack on hfe revisited. In *Accepted for ISC 2008*, Lecture Notes in Computer Science. Springer. Presented at Western European Workshop on Research in Cryptology 2007.
41. Ding, J., Schmidt, D., and Yin, Z.: Cryptanalysis of the new tts scheme in ches 2004. *Int. J. Inf. Sec.*, 5(4):231–240 (2006).
42. Ding, J., Wolf, C., and Yang, B.Y.: ℓ-invertible cycles for multivariate quadratic public key cryptography. In *PKC*, volume 4450 of *LNCS*, pages 266–281. Springer (2007).
43. Ding, J. and Yang, B.Y.: Multivariate polynomials for hashing. In *Inscrypt*, Lecture Notes in Computer Science. Springer (2007). To appear, cf. `http://eprint.iacr.org/2007/137`.
44. Ding, J., Yang, B.Y., Chen, C.H.O., Chen, M.S., and Cheng, C.M.: New differential-algebraic attacks and reparametrization of rainbow. In *Applied Cryptography and Network Security*, Lecture Notes in Computer Science. Springer (2008). To appear, cf. `http://eprint.iacr.org/2008/108`.
45. Ding, J., Yang, B.Y., Dubois, V., Cheng, C.M., and Chen, O.C.H.: Breaking the symmetry: a way to resist the new differential attack. `http://eprint.iacr.org/2007/366`.
46. Dubois, V., Fouque, P.A., Shamir, A., and Stern, J.: Practical cryptanalysis of sflash. In *Advances in Cryptology — CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 1–12. Alfred Menezes, ed., Springer (2007). ISBN 978-3-540-74142-8.

47. Dubois, V., Fouque, P.A., and Stern, J.: Cryptanalysis of sflash with slightly modified parameters. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 264–275. Springer (2007). ISBN 3-540-72539-3.
48. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139:61–88 (1999).
49. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press (2002).
50. Faugère, J.C. and Joux, A.: Algebraic cryptanalysis of Hidden Field Equations (HFE) using Gröbner bases. In *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Dan Boneh, ed., Springer (2003).
51. Faugère, J.C. and Perret, L.: Polynomial equivalence problems: Algorithmic and theoretical aspects. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 30–47. Springer (2006). ISBN 3-540-34546-9.
52. Fell, H. and Diffie, W.: Analysis of public key approach based on polynomial substitution. In *Advances in Cryptology — CRYPTO 1985*, volume 218 of *Lecture Notes in Computer Science*, pages 340–349. Hugh C. Williams, ed., Springer (1985).
53. Feng Bao, Robert H. Deng, and Jianying Zhou (editors): *Public Key Cryptography — PKC 2004*, (2004). ISBN 3-540-21018-0.
54. Fouque, P.A., Granboulan, L., and Stern, J.: Differential cryptanalysis for multivariate schemes. In Eurocrypt [90]. 341–353.
55. Fouque, P.A., Macario-Rat, G., Perret, L., and Stern, J.: Total break of the ℓIC- signature scheme. In *Public Key Cryptography*, pages 1–17 (2008).
56. Geddes, K.O., Czapor, S.R., and Labahn, G.: *Algorithms for Computer Algebra*. Amsterdam, Netherlands: Kluwer (1992).
57. Geiselmann, W., Meier, W., and Steinwandt, R.: An attack on the Isomorphisms of Polynomials problem with one secret. Cryptology ePrint Archive, Report 2002/143 (2002). `http://eprint.iacr.org/2002/143`, version from 2002-09-20, 12 pages.
58. Goubin, L. and Courtois, N.T.: Cryptanalysis of the TTM cryptosystem. In *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 44–57. Tatsuaki Okamoto, ed., Springer (2000).
59. Gouget, A. and Patarin, J.: Probabilistic multivariate cryptography. In P.Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 1–18. Springer (2006). ISBN 3-540-68799-8.
60. Granboulan, L., Joux, A., and Stern, J.: Inverting hfe is quasipolynomial. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.
61. Hasegawa, S. and Kaneko, T.: An attacking method for a public key cryptosystem based on the difficulty of solving a system of non-linear equations. In *Proc. 10th Symposium on Information Theory and Its applications*, pages JA5–3 (1987).
62. Kasahara, M. and Sakai, R.: A construction of public-key cryptosystem based on singular simultaneous equations. In *Symposium on Cryptography and Information Security — SCIS 2004*. The Institute of Electronics, Information and Communication Engineers (2004). 6 pages.

63. Kasahara, M. and Sakai, R.: A construction of public key cryptosystem for realizing ciphtertext of size 100 bit and digital signature scheme. *IEICE Trans. Fundamentals*, E87-A(1):102–109 (2004). Electronic version: `http://search.ieice.org/2004/files/e000a01.htm\#e87-a,1,102`.
64. Kipnis, A., Patarin, J., and Goubin, L.: Unbalanced Oil and Vinegar signature schemes. In *Advances in Cryptology — EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Jacques Stern, ed., Springer (1999).
65. Kipnis, A. and Shamir, A.: Cryptanalysis of the oil and vinegar signature scheme. In *Advances in Cryptology — CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 257–266. Hugo Krawczyk, ed., Springer (1998).
66. Kipnis, A. and Shamir, A.: Cryptanalysis of the HFE public key cryptosystem. In *Advances in Cryptology — CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Michael Wiener, ed., Springer (1999). `http://www.minrank.org/hfesubreg.ps` or `http://citeseer.nj.nec.com/kipnis99cryptanalysis.html`.
67. Lazard, D.: Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In *EUROCAL 83*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer (1983).
68. Levy-dit-Vehel, F. and Perret, L.: Polynomial equivalence problems and applications to multivariate cryptosystems. In *Progress in Cryptology — INDOCRYPT 2003*, volume 2904 of *Lecture Notes in Computer Science*, pages 235–251. Thomas Johansson and Subhamoy Maitra, editors, Springer (2003).
69. Macaulay, F.S.: *The algebraic theory of modular systems*, volume xxxi of *Cambridge Mathematical Library*. Cambridge University Press (1916).
70. Matsumoto, T. and Imai, H.: Public quadratic polynomial-tuples for efficient signature verification and message-encryption. In *Advances in Cryptology — EUROCRYPT 1988*, volume 330 of *Lecture Notes in Computer Science*, pages 419–545. Christoph G. Günther, ed., Springer (1988).
71. Matsumoto, T., Imai, H., Harashima, H., and Miyagawa, H.: High speed signature scheme using compact public key (1985). National Conference of system and information of the Electronic Communication Association of year Sowa 60, S9-5.
72. Moh, T.: A public key system with signature and master key function. *Communications in Algebra*, 27(5):2207–2222 (1999). Electronic version: `http://citeseer/moh99public.html`.
73. Moh, T.T.: The recent attack of Nie *et al* on TTM is faulty. Http://eprint.iacr.org/2006/417.
74. Moh, T.T.: Two new examples of TTM. Http://eprint.iacr.org/2007/144.
75. Nagata, M.: *On Automorphism Group of* $K[x, y]$, volume 5 of *Lectures on Mathematics*. Kyoto University, Kinokuniya, Tokyo (1972).
76. NESSIE: New European Schemes for Signatures, Integrity, and Encryption. Information Society Technologies programme of the European commission (IST-1999-12324). `http://www.cryptonessie.org/`.
77. Okamoto, E. and Nakamura, K.: Evaluation of public key cryptosystems proposed recently. In *Proc 1986's Symposium of cryptography and information security*, volume D1 (1986).

78. Ong, H., Schnorr, C., and Shamir, A.: Signatures through approximate representations by quadratic forms. In *Advances in cryptology, Crypto '83*, pages 117–131. Plenum Publ. (1984).
79. Ong, H., Schnorr, C., and Shamir, A.: Efficient signature schemes based on polynomial equations. In G.R. Blakley and D. Chaum, editors, *Advances in cryptology, Crypto '84*, volume 196 of *LNCS*, pages 37–46. Springer (1985).
80. Patarin, J.: The oil and vinegar signature scheme. Dagstuhl Workshop on Cryptography, September, 1997.
81. Patarin, J.: Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. In *Advances in Cryptology — CRYPTO 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Don Coppersmith, ed., Springer (1995).
82. Patarin, J.: Asymmetric cryptography with a hidden monomial. In *Advances in Cryptology — CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 45–60. Neal Koblitz, ed., Springer (1996).
83. Patarin, J.: Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In *Advances in Cryptology — EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Ueli Maurer, ed., Springer (1996). Extended Version: `http://www.minrank.org/hfe.pdf`.
84. Patarin, J., Courtois, N., and Goubin, L.: Flash, a fast multivariate signature algorithm. In C. Naccache, editor, *Progress in cryptology, CT-RSA*, volume 2020 of *LNCS*, pages 298–307. Springer (2001).
85. Patarin, J., Goubin, L., and Courtois, N.: C^*_{-+} and HM: Variations around two schemes of T. Matsumoto and H. Imai. In *Advances in Cryptology — ASIACRYPT 1998*, volume 1514 of *Lecture Notes in Computer Science*, pages 35–49. Kazuo Ohta and Dingyi Pei, editors, Springer (1998). Extended Version: `http://citeseer.nj.nec.com/patarin98plusmn.html`.
86. Patarin, J., Goubin, L., and Courtois, N.: Improved algorithms for Isomorphisms of Polynomials. In *Advances in Cryptology — EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 184–200. Kaisa Nyberg, ed., Springer (1998). Extended Version: `http://www.minrank.org/ip6long.ps`.
87. Perret, L.: A fast cryptanalysis of the isomorphism of polynomials with one secret problem. In Eurocrypt [90]. 17 pages.
88. Pil Joong Lee, ed.: *Advances in Cryptology — ASIACRYPT 2004*, (2004). ISBN 3-540-23975-8.
89. Pollard, J.M. and Schnorr, C.P.: An efficient solution of the congruence $x^2 + ky^2 = m \pmod n$. *IEEE Trans. Inform. Theory*, 33(5):702–709 (1987).
90. Ronald Cramer, ed.: *Advances in Cryptology — EUROCRYPT 2005*, (2005). ISBN 3-540-25910-4.
91. Serge Vaudenay, ed.: *Public Key Cryptography — PKC 2005*, (2005). ISBN 3-540-24454-9.
92. Shamir, A.: Efficient signature schemes based on birational permutations. In *Advances in Cryptology — CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 1–12. Douglas R. Stinson, ed., Springer (1993).
93. Shestakov, I.P. and Umirbaev, U.U.: The Nagata automorphism is wild. *Proc. Natl. Acad. Sci. USA*, 100:12561–12563 (2003).

94. Sugita, M., Kawazoe, M., and Imai, H.: Gröbner basis based cryptanalysis of sha-1. Cryptology ePrint Archive, Report 2006/098 (2006). `http://eprint.iacr.org/`.
95. Tsujii, S., Kurosawa, K., Itoh, T., Fujioka, A., and Matsumoto, T.: A public key cryptosystem based on the difficulty of solving a system of nonlinear equations. *ICICE Transactions (D) J69-D*, 12:1963–1970 (1986).
96. Tsujii, S., Fujioka, A., and Hirayama, Y.: Generalization of the public key cryptosystem based on the difficulty of solving a system of non-linear equations. In *ICICE Transactions (A) J72-A*, volume 2, pages 390–397 (1989). English version is appended at http://eprint.iacr.org/2004/336.
97. Tsujii, S., Fujioka, A., and Itoh, T.: Generalization of the public key cryptosystem based on the difficulty of solving a system of non-linear equations. In *Proc. 10th Symposium on Information Theory and Its applications*, pages JA5–3 (1987).
98. Wang, L.C. and Chang, F.H.: Tractable rational map cryptosystem (version 2). `http://eprint.iacr.org/2004/046`, ver. 20040221:212731.
99. Wang, L.C. and Chang, F.H.: Tractable rational map cryptosystem (version 4). `http://eprint.iacr.org/2004/046`, ver. 20060203:065450.
100. Wang, L.C., Hu, Y.H., Lai, F., Chou, C.Y., and Yang, B.Y.: Tractable rational map signature. In PKC [91], pages 244–257. ISBN 3-540-24454-9.
101. Wang, L.C., Yang, B.Y., Hu, Y.H., and Lai, F.: A "medium-field" multivariate public-key encryption scheme. In *CT-RSA 2006*, volume 3860 of *LNCS*, pages 132–149. David Pointcheval, ed., Springer (2006). ISBN 3-540-31033-9.
102. Wolf, C., Braeken, A., and Preneel, B.: Efficient cryptanalysis of RSE(2)PKC and RSSE(2)PKC. In *Conference on Security in Communication Networks — SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 294–309. Springer (2004). Extended version: `http://eprint.iacr.org/2004/237`.
103. Wolf, C. and Preneel, B.: Superfluous keys in $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic asymmetric systems. In PKC [91], pages 275–287. Extended version `http://eprint.iacr.org/2004/361/`.
104. Wolf, C. and Preneel, B.: Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report 2005/077 (2005). `http://eprint.iacr.org/2005/077/`, 64 pages.
105. Yang, B.Y. and Chen, J.M.: All in the XL family: Theory and practice. In *ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86. Springer (2004).
106. Yang, B.Y. and Chen, J.M.: Theoretical analysis of XL over small fields. In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer (2004).
107. Yang, B.Y. and Chen, J.M.: Building secure tame-like multivariate public-key cryptosystems: The new TTS. In *ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 518–531. Springer (2005).
108. Yang, B.Y., Chen, J.M., and Chen, Y.H.: TTS: High-speed signatures on a low-cost smart card. In *CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 371–385. Springer (2004).
109. Yang, B.Y., Chen, O.C.H., and Chen, J.M.: The limit of XL implemented with sparse matrices. Workshop record, PQCrypto workshop, Leuven 2006. Http://postquantum.cr.yp.to/pqcrypto2006record.pdf.

110. Yang, B.Y., Cheng, D.C.M., Chen, B.R., and Chen, J.M.: Implementing minimized multivariate public-key cryptosystems on low-resource embedded systems. In *SPC 2006*, volume 3934 of *Lecture Notes in Computer Science*, pages 73–88. Springer (2006).

Index