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 $2^{(\mathrm{constant}+o(1))(\lg n)^{1/3}(\lg \lg n)^{2/3}}$ 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.

Cryptographers:
How can we encrypt,
decrypt, sign, verify, etc.?

Functioning cryptographic systems:
DES, Triple DES, AES,
RSA, McEliece encryption,
Merkle hash-tree signatures,
Merkle–Hellman knapsack encryption,
Buchmann–Williams class-group encryption,
ECDSA, $\mathrm{HFE}^{\mathrm{v}-}$, NTRU, etc.

Cryptanalysts:
What can an attacker do using $< 2^b$
operations on a classical computer?

Unbroken cryptographic systems:
Triple DES (for $b \le 112$), AES (for $b \le 256$),
RSA with $b^{3+o(1)}$-bit modulus,
McEliece with code length $b^{1+o(1)}$,
Merkle signatures with "strong" $b^{1+o(1)}$-bit hash,
BW with "strong" $b^{2+o(1)}$-bit discriminant,
ECDSA with "strong" $b^{1+o(1)}$-bit curve,
$\mathrm{HFE}^{\mathrm{v}-}$ with $b^{1+o(1)}$ polynomials,
NTRU with $b^{1+o(1)}$ bits, etc.

Algorithm designers and implementors:
Exactly how small and fast
are the unbroken cryptosystems?

Most efficient unbroken cryptosystems:
e.g., can verify signature in time $b^{2+o(1)}$
using ECDSA with "strong" $b^{1+o(1)}$-bit curve

Users

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.

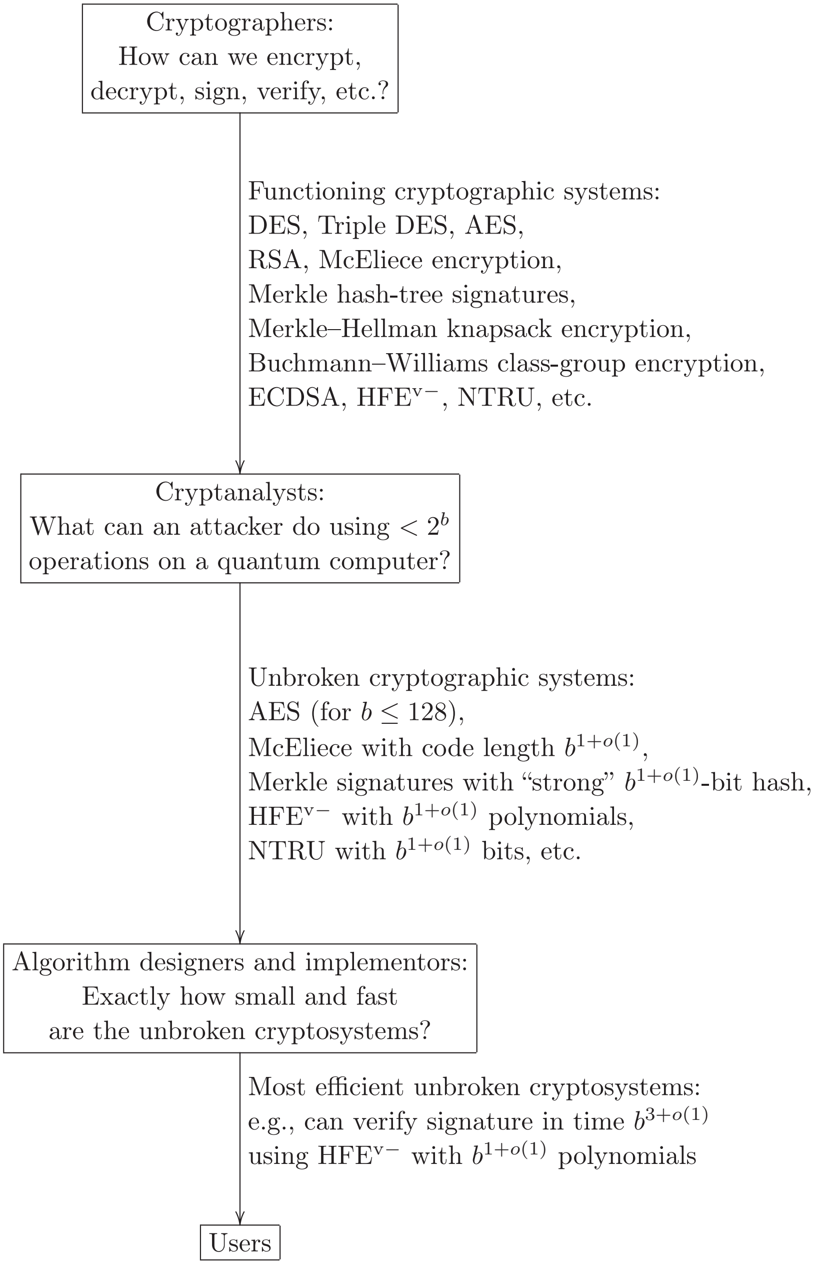

Fig. 2. Post-quantum cryptography. Warning: Sizes and times are simplified to $b^{1+o(1)}$, $b^{2+o(1)}$, etc. Optimization of any specific b requires a more detailed analysis.

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 $\ell \in L$, $2^i < 2^j$, $2^i + 2^j \leq 2b$; $\ell x^{2^i} v_j$ with $\ell \in L$, $2^i \leq 2b$; $\ell v_i v_j$; 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$^{\text{v}-}$ 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$^{\text{v}-}$ 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