Progress in Post-Quantum Cryptography

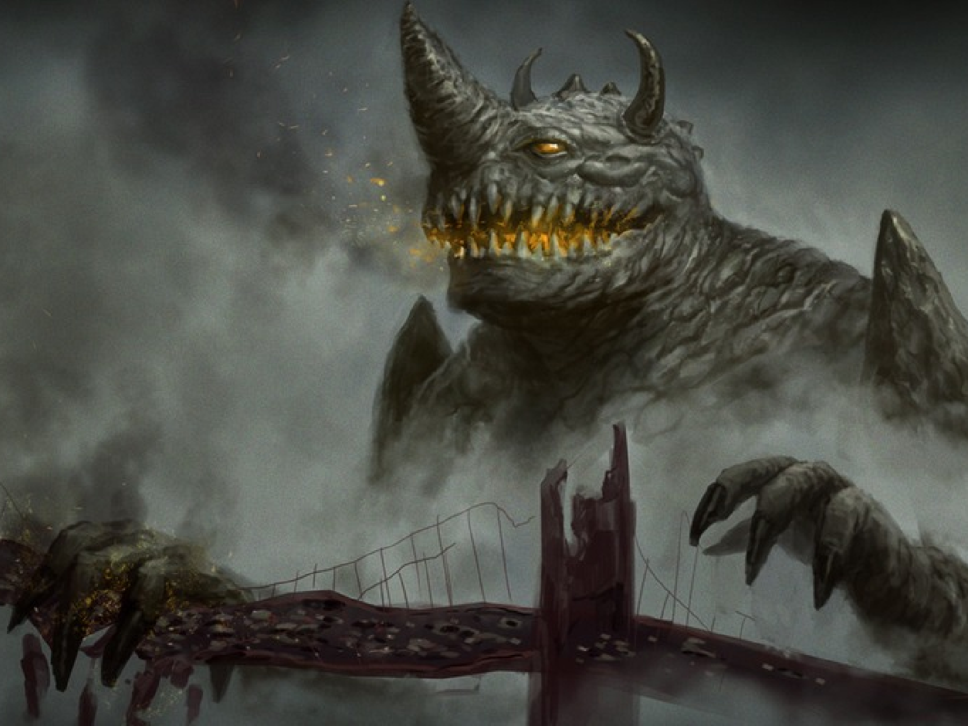
Tanja Lange

Technische Universiteit Eindhoven

PQCRYPTO
ICT-645622

29 April 2018

International View of the State-of-the-Art of Cryptography



Algorithms for Quantum Computation: Discrete Logarithms and Factoring

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as efficient when the number of steps of the algorithms grows as a polynomial in the size of the input. The class of prob-

. . . but universal quantum computers are coming

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.

. . . but universal quantum computers are coming

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "We're actually doing things that are making us think like, 'hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.' It's within reach."
- Fast-forward to 2022, or 2027. Universal quantum computers exist.

. . . but universal quantum computers are coming

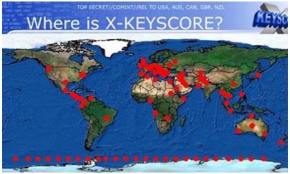
- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "We're actually doing things that are making us think like, 'hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.' It's within reach."
- Fast-forward to 2022, or 2027. Universal quantum computers exist.
- Shor's algorithm solves in polynomial time:
  - Integer factorization. RSA is dead.
  - The discrete-logarithm problem in finite fields. DSA is dead.
  - The discrete-logarithm problem on elliptic curves. ECDSA is dead.
- This breaks all current public-key cryptography on the Internet!

… but universal quantum computers are coming

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: “We’re actually doing things that are making us think like, ‘hey this isn’t 50 years off, this is maybe just 10 years off, or 15 years off.’ It’s within reach.”
- Fast-forward to 2022, or 2027. Universal quantum computers exist.
- Shor’s algorithm solves in polynomial time:
  - Integer factorization. RSA is dead.
  - The discrete-logarithm problem in finite fields. DSA is dead.
  - The discrete-logarithm problem on elliptic curves. ECDSA is dead.
- This breaks all current public-key cryptography on the Internet!
- Also, Grover’s algorithm speeds up brute-force searches.
- Example: Only 2^{64} quantum operations to break AES-128;
  2^{128} quantum operations to break AES-256.

Even higher urgency for long-term confidentiality

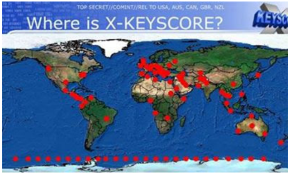
- Attacker can break currently used encryption (ECC, RSA) with a quantum computer.
- Even worse, today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. All data can be recovered in clear from recording traffic and breaking the public key scheme.
- How many years are you required to keep your data secret? From whom?



- Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement

Even higher urgency for long-term confidentiality

- Attacker can break currently used encryption (ECC, RSA) with a quantum computer.
- Even worse, today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. All data can be recovered in clear from recording traffic and breaking the public key scheme.
- How many years are you required to keep your data secret? From whom?



- Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement . . . and an important function of signatures is to protect operating system upgrades.
- Protect your upgrades *now* with post-quantum signatures.

Initial recommendations of long-term secure post-quantum systems

Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, Tanja Lange, Mohamed Saied Emam Mohamed, Christian Rechberger, Peter Schwabe, Nicolas Sendrier, Frederik Vercauteren, Bo-Yin Yang

Initial recommendations

- **Symmetric encryption** Thoroughly analyzed, 256-bit keys:
  - AES-256
  - Salsa20 with a 256-bit key

  Evaluating: Serpent-256, ...
- **Symmetric authentication** Information-theoretic MACs:
  - GCM using a 96-bit nonce and a 128-bit authenticator
  - Poly1305
- **Public-key encryption** McEliece with binary Goppa codes:
  - length $n = 6960$, dimension $k = 5413$, $t = 119$ errors

  Evaluating: QC-MDPC, Stehlé-Steinfeld NTRU, ...
- **Public-key signatures** Hash-based (minimal assumptions):
  - XMSS with any of the parameters specified in CFRG draft
  - SPHINCS-256

  Evaluating: HFEv-, ...

NIST Post-Quantum “Competition”

December 2016, after public feedback: NIST calls for submissions of post-quantum cryptosystems to standardize.

30 November 2017: NIST receives 82 submissions.

	Signatures	KEM/Encryption	Overall
Lattice-based	4	24	28
Code-based	5	19	24
Multi-variate	7	6	13
Hash-based	4		4
Other	3	10	13
Total	**23**	**59**	**82**

"Complete and proper" submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE. CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DAGS. Ding Key Exchange. DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON. FrodoKEM. GeMSS. Giophantus. Gravity-SPHINCS. Guess Again. Gui. HILA5. HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton. LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS. NewHope. NTRUEncrypt. NTRU-HRSS-KEM. NTRU Prime. NTS-KEM. Odd Manhattan. OKCN/AKCN/CNKE. Ouroboros-R. Picnic. pqNTRUSign. pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA. RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB. SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE. CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DAGS. Ding Key Exchange. DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON. FrodoKEM. GeMSS. Giophantus. Gravity-SPHINCS. Guess Again. Gui. HILA5. HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton. LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS. NewHope. NTRUEncrypt. NTRU-HRSS-KEM. NTRU Prime. NTS-KEM. Odd Manhattan. OKCN/AKCN/CNKE. Ouroboros-R. Picnic. pqNTRUSign. pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA. RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB. SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

Some attack scripts already posted causing **total break** or **serious tweaks**. Many more receiving detailed analysis.

Classic McEliece
conservative code-based cryptography

Daniel J. Bernstein, Tung Chou, Tanja Lange,
Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen,
Edoardo Persichetti, Christiane Peters, Peter Schwabe,
Nicolas Sendrier, Jakub Szefer, Wen Wang

Key sizes and key-generation speed

`mceliece6960119` parameter set:
1047319 bytes for public key.
13908 bytes for secret key.

`mceliece8192128` parameter set:
1357824 bytes for public key.
14080 bytes for secret key.

Key sizes and key-generation speed

`mceliece6960119` parameter set:
1047319 bytes for public key.
13908 bytes for secret key.

`mceliece8192128` parameter set:
1357824 bytes for public key.
14080 bytes for secret key.

Current software: billions of cycles to generate a key;
not much optimization effort yet.
All code runs in constant time.

Key sizes and key-generation speed

`mceliece6960119` parameter set:
1047319 bytes for public key.
13908 bytes for secret key.

`mceliece8192128` parameter set:
1357824 bytes for public key.
14080 bytes for secret key.

Current software: billions of cycles to generate a key;
not much optimization effort yet.
All code runs in constant time.

Very fast in hardware (PQCrypto 2018; CHES 2017):
a few million cycles at 231MHz
using 129059 modules, 1126 RAM blocks
on Altera Stratix V FPGA.

Short ciphertexts

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Short ciphertexts

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
295932 cycles for enc,
355152 cycles for dec (decoding, hashing, etc.).

Short ciphertexts

`mceliece6960119` parameter set:
  226 bytes for ciphertext.

`mceliece8192128` parameter set:
  240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
  295932 cycles for enc,
  355152 cycles for dec (decoding, hashing, etc.).

Again very fast in hardware:
17140 cycles for decoding.

Short ciphertexts

`mceliece6960119` parameter set:
226 bytes for ciphertext.

`mceliece8192128` parameter set:
240 bytes for ciphertext.

Constant time software (measured on Haswell, larger parameters):
295932 cycles for enc,
355152 cycles for dec (decoding, hashing, etc.).

Again very fast in hardware:
17140 cycles for decoding.

Can tweak parameters for even smaller ciphertexts, not much penalty in key size.

One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix H and syndrome s,
can attacker efficiently find e with $s = He$?

One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix H and syndrome s,
can attacker efficiently find e with $s = He$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.

One-wayness (OW-CPA)

Fundamental security question:
Given random parity-check matrix H and syndrome s,
can attacker efficiently find e with $s = He$?

1962 Prange: simple attack idea
guiding sizes in 1978 McEliece.

The McEliece system (with later key-size optimizations)
uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$
to achieve 2^λ security against Prange's attack.

Here $c_0 \approx 0.7418860694$.

PQCRYPTO
ICT-645622

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today.
Same $c_0 \approx 0.7418860694$.

40 years and more than 30 analysis papers later

1962 Prange; 1981 Clark–Cain, crediting Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk; 1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg; 1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau; 1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne; 1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier; 2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg; 2009 Bernstein (**post-quantum**); 2009 Finiasz–Sendrier; 2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae; 2012 Becker–Joux–May–Meurer; 2013 Hamdaoui–Sendrier; 2015 May–Ozerov; 2016 Canto Torres–Sendrier; 2017 Kachigar–Tillich (**post-quantum**); 2017 Both–May; 2018 Both–May; 2018 Kirshanova (**post-quantum**).

The McEliece system uses $(c_0 + o(1))\lambda^2(\lg \lambda)^2$-bit keys as $\lambda \to \infty$ to achieve 2^λ security against all attacks known today.
Same $c_0 \approx 0.7418860694$.

Replacing λ with 2λ stops all known *quantum* attacks.

Classic McEliece

McEliece's system prompted huge amount of followup work.

Some work improves efficiency while clearly preserving security:

- Niederreiter's dual PKE
  (use parity check matrix instead of generator matrix);
- many decoding speedups; ...

Classic McEliece uses all this, with constant-time implementations.

- Write $H = (I_{n-k}|T)$, public key is $(n-k) \times k$ matrix T,
  $n - k = w \log_2 q$. H constructed from binary Goppa code.
- Encapsulate using e of weight w.

`mceliece6960119` parameter set (2008 Bernstein–Lange–Peters):
$q = 8192$, $n = 6960$, $w = 119$.

`mceliece8192128` parameter set:
$q = 8192$, $n = 8192$, $w = 128$.

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.
2. Ciphertext includes another hash of e ("confirmation").

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.
2. Ciphertext includes another hash of e ("confirmation").
3. Dec includes recomputation and verification of ciphertext.

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.

2. Ciphertext includes another hash of e ("confirmation").

3. Dec includes recomputation and verification of ciphertext.

4. KEM never fails: if inversion fails or ciphertext does not match, return hash of $(\text{secret}, \text{ciphertext})$.

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.

2. Ciphertext includes another hash of e ("confirmation").

3. Dec includes recomputation and verification of ciphertext.

4. KEM never fails: if inversion fails or ciphertext does not match, return hash of $(\text{secret}, \text{ciphertext})$.

Further features of system that simplify attack analysis:

5. Ciphertext is deterministic function of input e: i.e., inversion recovers all randomness used to create ciphertexts.

IND-CCA2 conversions

Classic McEliece follows best practices from literature:

1. Session key: feed random e through standard hash function.

2. Ciphertext includes another hash of e ("confirmation").

3. Dec includes recomputation and verification of ciphertext.

4. KEM never fails: if inversion fails or ciphertext does not match, return hash of $(\text{secret}, \text{ciphertext})$.

Further features of system that simplify attack analysis:

5. Ciphertext is deterministic function of input e: i.e., inversion recovers all randomness used to create ciphertexts.

6. There are no inversion failures for legitimate ciphertexts.

Classic McEliece highlights

- Security asymptotics unchanged by 40 years of cryptanalysis.
- Short ciphertexts.
- Efficient and straightforward conversion of OW-CPA PKE into IND-CCA2 KEM.
- Constant-time software implementations.
- FPGA implementation of full cryptosystem.
- Open-source (public domain) implementations.
- No patents.

Recent attacks

"Code-based" does not imply secure!

Example: code-based signature scheme RaCoSS was broken 3 different ways

1. Bug in code: bit vs. byte confusion meant only evey 8th bit verified.
2. Preimages for RaCoSS' specia hash function: only
$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$
possible outputs.
3. The code dimensions give a lot of freedom to the attacker – our forged signature is better than a real one!

Recent attacks

"Code-based" does not imply secure!

Example: code-based signature scheme RaCoSS was broken 3 different ways

1. Bug in code: bit vs. byte confusion meant only evey 8th bit verified.
2. Preimages for RaCoSS' specia hash function: only

$$\binom{2400}{3} = 2301120800 \sim 2^{31.09}$$

   possible outputs.
3. The code dimensions give a lot of freedom to the attacker – our forged signature is better than a real one!

12 fully broken (efficient script posted) systems fall into

- Codes: Edon-K, pqsigRM, RaCoSS, RankSign.
- Lattices: Compact LWE.
- Multivariate: CFPKM, DME.
- Other: Guess Again, HK17, RVB, SRTP, Walnut DSA.

Further resources

- https://2017.pqcrypto.org/school: PQCRYPTO summer school with 21 lectures on video + slides + exercises.
- https://2017.pqcrypto.org/exec: Executive school (12 lectures), less math, more overview. So far slides, soon videos.
- https://pqcrypto.org: Our survey site.
  - Many pointers: e.g., to PQCrypto conferences;
  - Bibliography for 4 major PQC systems.
- https://pqcrypto.eu.org: PQCRYPTO EU project.
  - Expert recommendations.
  - Free software libraries.
  - More video presentations, slides, papers.
- https://twitter.com/pqc_eu: PQCRYPTO Twitter feed.
- https://twitter.com/PQCryptoConf: PQCrypto conference Twitter feed.
- https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions NIST PQC competition.