The migration to post-quantum cryptography

Peter Schwabe

Max Planck Institute for Security and Privacy

October 9, 2025

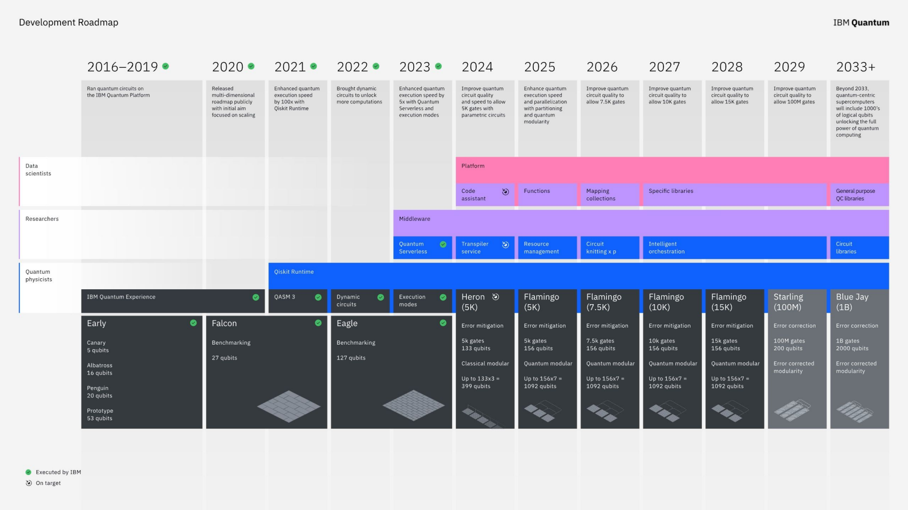
[A small demo]

Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

Peter W. Shor†

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.



See `https://www.ibm.com/quantum/blog/ibm-quantum-roadmap-2025`

Post-quantum crypto (PQC)

Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.

Post-quantum crypto (PQC)

Definition

Post-quantum crypto is (asymmetric) crypto that resists attacks using classical *and quantum* computers.

5 main directions

- Lattice-based crypto (PKE and Sigs)
- Code-based crypto (mainly PKE)
- Multivariate-based crypto (mainly Sigs)
- Hash-based signatures (only Sigs)
- Isogeny-based crypto (it's complicated. . .)

Should you care now?

"Harvest now, decrypt later"

https://en.wikipedia.org/wiki/Utah_Data_Center#/media/File:EFF_photograph_of_NSA's_Utah_Data_Center.jpg

Should you care now?

"Harvest now, decrypt later"

https://en.wikipedia.org/wiki/Utah_Data_Center#/media/File:EFF_photograph_of_NSA's_Utah_Data_Center.jpg

Mosca's theorem

$$X + Y > Z$$

- X: For how long do you need encrypted data to be secure?
- Y: How long does it take you to migrate to PQC
- Z: Time it will take to build a cryptographically relevant quantum computer

If $X + Y > Z$, you should worry.

NIST PQC – how it started

Count of Problem Category	Column Labels		
Row Labels	**Key Exchange**	**Signature**	**Grand Total**
?	1		1
Braids	1	1	2
Chebychev	1		1
Codes	19	5	24
Finite Automata	1	1	2
Hash		4	4
Hypercomplex Numbers	1		1
Isogeny	1		1
Lattice	24	4	28
Mult. Var	6	7	13
Rand. walk	1		1
RSA	1	1	2
Grand Total	**57**	**23**	**80**

Overview tweeted by Jacob Alperin-Sheriff on Dec 4, 2017.

NIST PQC – how it went

NIST PQC

Nov. 2017 69 proposals	$\xrightarrow{\text{Round 1}}$	Feb. 2019 26 proposals	$\xrightarrow{\text{Round 2}}$	Jul. 2020 7+8 proposals	$\xrightarrow{\text{Round 3}}$	Jul. 2022 4 “winners”

NIST PQC – how it went

NIST PQC

Nov. 2017
69 proposals

Round 1 →

Feb. 2019
26 proposals

Round 2 →

Jul. 2020
7+8 proposals

Round 3 →

Jul. 2022
4 "winners"

"The public-key encryption and key-establishment algorithm that will be standardized is ***CRYSTALS-KYBER****. The digital signatures that will be standardized are CRYSTALS-Dilithium, FALCON, and SPHINCS*$^{+}$*. While there are multiple signature algorithms selected, NIST recommends* ***CRYSTALS-Dilithium*** *as the primary algorithm to be implemented"*

—NIST IR 8413-upd1

[Back to our demo]

So, all good? Is the world safe again?

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions
- **2008**: ***Rogue CA certificate*** using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- 1991: MD5 is proposed by Rivest
- 1993: Collisions in MD5 compression function (den Boer, Bosselaers)
- 1996: Dobbertin, Bosselaers, Preneel: concerns about MD5
- 2004: Wang presents MD5 collisions
- 2008: *Rogue CA certificate* using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)
- 2012: Flame malware exploits MD5 weaknesses

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions
- **2008**: ***Rogue CA certificate*** using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)
- **2012**: Flame malware exploits MD5 weaknesses

Replacing MD5 was "easy"!

Challenge 1: Performance

X25519 speed

- keygen: 28187 Skylake cycles
- shared: 87942 Skylake cycles

Kyber-768 speed

- keygen: 39750 Skylake cycles
- encaps: 53936 Skylake cycles
- decaps: 42339 Skylake cycles

Challenge 1: Performance

X25519 speed

- keygen: 28187 Skylake cycles
- shared: 87942 Skylake cycles

Kyber-768 speed

- keygen: 39750 Skylake cycles
- encaps: 53936 Skylake cycles
- decaps: 42339 Skylake cycles

X25519 sizes

- public key: 32 bytes

Kyber-768 sizes

- public key: 1184 bytes
- ciphertext: 1088 bytes

Challenge 2: A KEM is not DH!

$$\begin{array}{ccc} \boxed{\text{Alice}} & & \boxed{\text{Bob}} \\ A \leftarrow g^a & & B \leftarrow g^b \\ & \xrightarrow{\quad A \quad} & \\ & \xleftarrow{\quad B \quad} & \\ K \leftarrow B^a = (g^b)^a = g^{ab} & & K \leftarrow A^b = (g^a)^b = g^{ab} \end{array}$$

Challenge 2: A KEM is not DH!

Alice		Bob
$A \leftarrow g^a$		$B \leftarrow g^b$
	$\xleftarrow{\quad B \quad}$	
	$\xrightarrow{\quad A \quad}$	
$K \leftarrow B^a = (g^b)^a = g^{ab}$		$K \leftarrow A^b = (g^a)^b = g^{ab}$

Challenge 2: A KEM is not DH!

Initiator		Responder
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.Gen}$		
	$\xrightarrow{\mathsf{pk}}$	
		$(\mathsf{ct}, K) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk})$
	$\xleftarrow{\mathsf{ct}}$	
$K \leftarrow \mathsf{KEM.Dec}(\mathsf{ct}, \mathsf{sk})$		

Challenge 3: Bugs, bugs everywhere

Dilithium commit on Dec. 28, 2017

```diff
-      t  = buf[pos];
-      t |= (uint32_t)buf[pos + 1] << 8;
-      t |= (uint32_t)buf[pos + 2] << 16;
-      t &= 0xFFFFF;
+      t0  = buf[pos];
+      t0 |= (uint32_t)buf[pos + 1] << 8;
+      t0 |= (uint32_t)buf[pos + 2] << 16;
+      t0 &= 0xFFFFF;

-      t  = buf[pos + 2] >> 4;
-      t |= (uint32_t)buf[pos + 3] << 4;
-      t |= (uint32_t)buf[pos + 4] << 12;
+      t1  = buf[pos + 2] >> 4;
+      t1 |= (uint32_t)buf[pos + 3] << 4;
+      t1 |= (uint32_t)buf[pos + 4] << 12;
```

- Bug in Dilithium sampler
- Two consecutive coefficients are equal
- Allows key recovery
- Reported by Peter Pessl on Dec. 27, 2017

Challenge 3: Bugs, bugs everywhere

Questions about the range analysis of iNTT for "Faster Kyber and Dilithium on the Cortex-M4" #226

Closed **JunhaoHuang** opened this issue on Mar 3 · 4 comments

JunhaoHuang commented on Mar 3 • edited ...

Hi team, I am reading the Kyber code regarding the recent paper "Faster Kyber and Dilithium on the Cortex-M4", and I have a question about the matrix-vector product and Better Accumulation part regarding the *f_stack* version code.

I see that using the better accumulation technique in the f_speed version code, we can reduce each element of the output vector of matrix-vector product down to (-q,q). Since poly_invntt is normally used after the matrix-vector product, the range of the input vector of **poly_invntt** lies in (-q,q) in the *f_speed* version code. The **invntt** function works in this situation.

What I wonder is that in the *f_stack* version code, the **matacc** function actually uses the previous double basemul accumulation function, and it should produce the result vector with element in (-kq, kq), k is the security parameter of Kyber. For Kyber1024, the range of each polynomial element that **invntt** takes should be (-4q,4q). However, the **invntt** function is the same as the f_speed version code. The first four layers of the light butterflies in **invntt** involve some additions and subtractions without multiplication. Therefore, For Kyber1024 in the *f_stack* version code, two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the *f_stack* code and why does it still work?

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

Challenge 3: Bugs, bugs everywhere

“. . . two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?”

Challenge 3: Bugs, bugs everywhere

"...two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?"

"...On your question on why it still works, I believe that this is an edge case that does not get triggered by the testing scripts."

Challenge 3: Bugs, bugs everywhere

vincentvbh commented on Mar 6, 2021 Contributor Author ...

There is a bug in the inverse of NTT in Saber. But the bug is triggered with a very low probability that it is not triggered on testing.

Challenge 3: Bugs, bugs everywhere

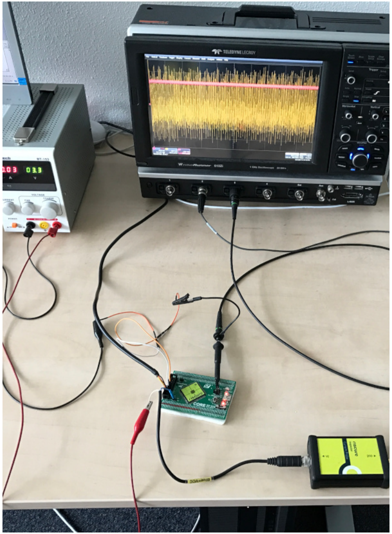
vincentvbh commented on Mar 6, 2021 Contributor Author ...

There is a bug in the inverse of NTT in Saber. But the bug is triggered with a very low probability that it is not triggered on testing.

Both NTT bugs found by Yang, Liu, Shi, Hwang, Tsai, Wang, and Seiler (TCHES 2022/4)

Challenge 4: Implementation Security

Challenge 4: Implementation Security



- Attackers see more than input/output:
  - Power consumption
  - Electromagnetic radiation
  - Timing
- **Side-channel attacks:**
  - Measure information
  - Use to obtain secret data

Challenge 4: Side-channel countermeasures

Hardware side-channels

- Require physical access to device
- Examples: Power, EM attacks
- Protection through dedicated countermeasures
- Typical slowdown of much more than 100%
- Progress, but no “conclusion”; we don’t know how to protect PQC!

Challenge 4: Side-channel countermeasures

Hardware side-channels

- Require physical access to device
- Examples: Power, EM attacks
- Protection through dedicated countermeasures
- Typical slowdown of much more than 100%
- Progress, but no "conclusion"; we don't know how to protect PQC!

Software side-channels

- Leak through microarchitectural side-channels
- No physical access required, can run *remotely*
- Traditional countermeasure: **constant-time**
  - No branching on secrets
  - No memory access at secret location
  - No variable-time arithmetic on secrets

An arms race with compilers

"KyberSlash"

```
t  = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
```

- Division by constant *usually* turns into multiplications
- Turns into `DIV` instructions for certain compiler flags
- `DIV` with secret divident leaks

Compiler (re-)introduced secret branch

```
for(j=0;j<8;j++) {
  mask = -(int16_t)((msg[i] >> j)&1);
  r->coeffs[8*i+j] = mask & ((KYBER_Q+1)/2);
}
```

- Carefully hand-crafted to avoid secret branch
- Secret branch re-introduced by clang $\geq$15

Advanced microarchitectural attacks

MELTDOWN

Hertzbleed

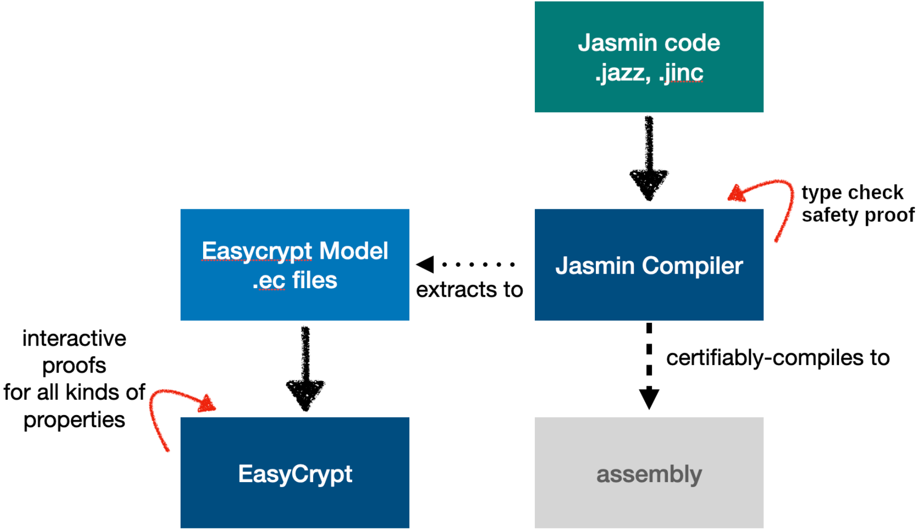
CACHE OUT

High-assurance PQC

FORMOSA CRYPTO

- Effort to **formally verify** crypto
- Currently three main projects:
  - EasyCrypt proof assistant
  - jasmin programming language
  - Libjade (PQ-)crypto library
- Core team of $\approx$ 30–40 people
- Discussion forum with >350 people

The toolchain and workflow



End-to-end formally verified ML-KEM

- Reference and AVX2-optimized implementations in Jasmin
- Proven (memory-/type-)safety of implementations
- Future-proof constant-time (using Intel's DOIT)
- Principled erasure of sensitive stack/register data at termination
- Systematic protections against Spectre v1
- (Extension to all Spectre variants needs merging)

End-to-end formally verified ML-KEM

- Reference and AVX2-optimized implementations in Jasmin
- Proven (memory-/type-)safety of implementations
- Future-proof constant-time (using Intel's DOIT)
- Principled erasure of sensitive stack/register data at termination
- Systematic protections against Spectre v1
- (Extension to all Spectre variants needs merging)
- ML-KEM specification in EasyCrypt
- Implementations proven functionally correct (EasyCrypt)
- Reductionist proof of IND-CCA security (EasyCrypt)

End-to-end formally verified ML-KEM

- Reference and AVX2-optimized implementations in Jasmin
- Proven (memory-/type-)safety of implementations
- Future-proof constant-time (using Intel's DOIT)
- Principled erasure of sensitive stack/register data at termination
- Systematic protections against Spectre v1
- (Extension to all Spectre variants needs merging)
- ML-KEM specification in EasyCrypt
- Implementations proven functionally correct (EasyCrypt)
- Reductionist proof of IND-CCA security (EasyCrypt)
- Ongoing work: wrap in ML-KEM "crypto agent"
- Ongoing work: real-world production deployment

https://github.com/pq-code-package/mlkem-libjade

Learn more

NIST PQC

- NIST PQC website:
  https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
- NIST mailing list:
  https://csrc.nist.gov/projects/post-quantum-cryptography/email-list
  https://groups.google.com/a/list.nist.gov/g/pqc-forum

Formosa Crypto

- Main website: https://formosa-crypto.org
- Team chat: https://formosa-crypto.zulipchat.com/

Learn even more

Papers related to high-assurance ML-KEM (1/2)

- Almeida, Barbosa, Barthe, Grégoire, Laporte, Léchenet, Oliveira, Pacheco, Quaresma, Schwabe, Séré, and Strub. **Formally verifying Kyber – Episode IV: Implementation Correctness.** CHES 2023. https://eprint.iacr.org/2023/215
- Almeida, Arranz Olmos, Barbosa, Barthe, Dupressoir, Grégoire, Laporte, Léchenet, Low, Oliveira, Pacheco, Quaresma, Schwabe, and Strub. **Formally verifying Kyber – Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt.** Crypto 2024. https://eprint.iacr.org/2024/843
- Barbosa and Schwabe. **Kyber terminates.** Polynesian Journal of Mathematics. https://eprint.iacr.org/2023/708
- Barbosa, Kannwischer, Lim, Schwabe, and Strub. **Formally Verified Correctness Bounds for Lattice-Based Cryptography.** ACM CCS 2025. https://eprint.iacr.org/2025/1562

Learn even more

Papers related to high-assurance ML-KEM (2/2)

- Ammanaghatta Shivakumar, Barthe, Grégoire, Laporte, Oliveira, Priya, Schwabe, and Tabary-Maujean. **Typing High-Speed Cryptography against Spectre v1.** IEEE S&P 2023. https://eprint.iacr.org/2022/1270
- Arranz Olmos, Barthe, Gonzalez, Grégoire, Laporte, Léchenet, Oliveira, and Schwabe. **High-assurance zeroization.**, CHES 2024. https://eprint.iacr.org/2023/1713
- Arranz-Olmos, Barthe, Grégoire, Jancar, Laporte, Oliveira, and Schwabe. **Let's DOIT: Using Intel's Extended HW/SW Contract for Secure Compilation of Crypto Code.** CHES 2025. https://eprint.iacr.org/2025/759
- Arranz Olmos, Barthe, Chuengsatiansup, Grégoire, Laporte, Oliveira, Schwabe, Yarom, and Zhang. **Protecting Cryptographic Code Against Spectre-RSB (and, in Fact, All Known Spectre Variants).** ASPLOS 2025. https://eprint.iacr.org/2024/1070