The migration to post-quantum cryptography

Peter Schwabe

Max Planck Institute for Security and Privacy

October 21, 2025

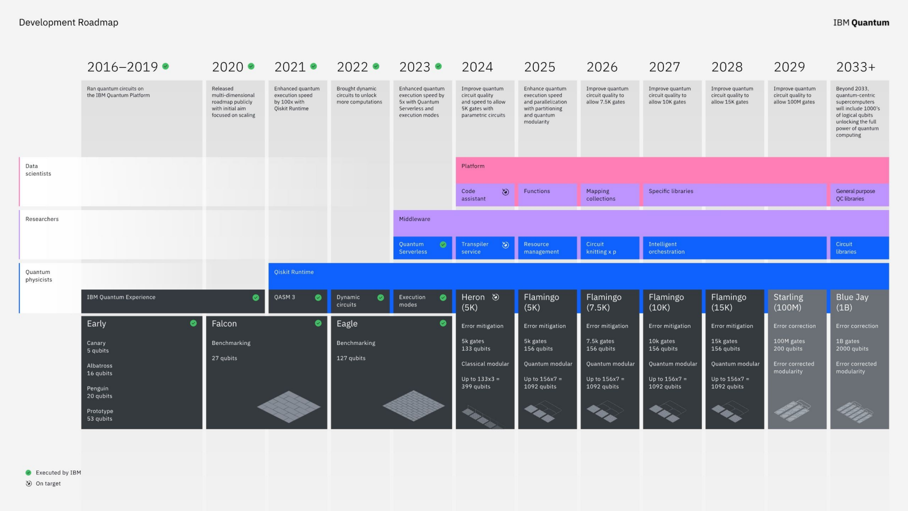
[A small demo]

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” (HNDL)

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" (HNDL)

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.

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST

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”

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST

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”

“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"!

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive **X** to **Y**.

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive X to Y.

- CBOM: *Cryptographic bill of materials*, know where you are using primitive X.

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive X to Y.

- **CBOM:** *Cryptographic bill of materials*, know where you are using primitive X.
- **API design:** Call *functionalities* rather than *primitives*

"Cryptographic Agility" to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive **X** to **Y**.

- **CBOM:** *Cryptographic bill of materials*, know where you are using primitive **X**.
- **API design:** Call *functionalities* rather than *primitives*
- **Protocol and system design:**
  - Plan for version transitions that break compatibility (example: Signal) 😃
  - Negotiate primitives at runtime (example: TLS) 😬

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive **X** to **Y**.

- **CBOM:** *Cryptographic bill of materials*, know where you are using primitive **X**.
- **API design:** Call *functionalities* rather than *primitives*
- **Protocol and system design:**
  - Plan for version transitions that break compatibility (example: Signal) 😃
  - Negotiate primitives at runtime (example: TLS) 😬
- **Upgrade with confidence:** Integrate formal modelling and verification in CI

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive **X** to **Y**.

- **CBOM:** *Cryptographic bill of materials*, know where you are using primitive **X**.
- **API design:** Call *functionalities* rather than *primitives*
- **Protocol and system design:**
  - Plan for version transitions that break compatibility (example: Signal) 😃
  - Negotiate primitives at runtime (example: TLS) 😬
- **Upgrade with confidence:** Integrate formal modelling and verification in CI
- **HW resource planning:** Plan for increased computational and memory requirements

“Cryptographic Agility” to the rescue!

Definition (?)

Cryptographic Agility: the ability to efficiently switch from cryptographic primitive X to Y.

- **CBOM:** *Cryptographic bill of materials*, know where you are using primitive X.
- **API design:** Call *functionalities* rather than *primitives*
- **Protocol and system design:**
  - Plan for version transitions that break compatibility (example: Signal) 😃
  - Negotiate primitives at runtime (example: TLS) 😬
- **Upgrade with confidence:** Integrate formal modelling and verification in CI
- **HW resource planning:** Plan for increased computational and memory requirements
- **Protocol standards:** Updates needed wherever endpoints are not controlled by one party

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!

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

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

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

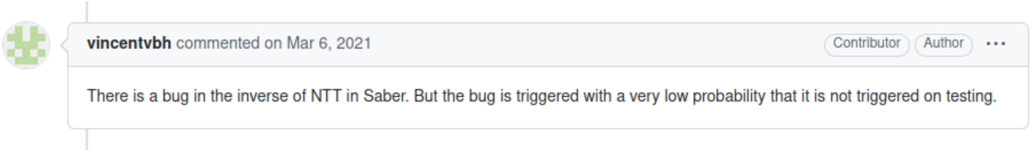
"...two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?"

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



Challenge 3: Bugs, bugs everywhere

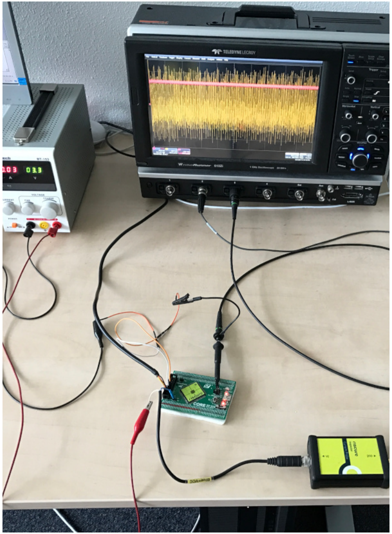
vincentvbh commented on Mar 6, 2021 Contributor Author ...

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

Constant-time is not enough

Leaking binaries from constant-time code

- Source-code constant-time is insufficient
- Mainstream compilers have to concept of secret
- Optimizations re-introduce timing leaks

Constant-time is not enough

Leaking binaries from constant-time code

- Source-code constant-time is insufficient
- Mainstream compilers have to concept of secret
- Optimizations re-introduce timing leaks

Advanced microarchitectural attacks

- Spectre (2018): leakage during *speculative execution*
- Hertzbleed (2022): Translate power leakage to timing leakage
- GoFetch (2024): Load instructions leak *data*
- ...

Hybrid deployments

Combine PQC with ECC: don’t make systems less secure in the attempt to make them more secure against future quantum attackers!

Hybrid deployments

Combine PQC with ECC: don't make systems less secure in the attempt to make them more secure against future quantum attackers!

- Cryptanalysis of PQ schemes is not as stable as for ECC
  - SIKE. . . (was deployed, **hybrid**, by Google and Cloudflare)
  - Late breaks of GeMSS and Rainbow

Hybrid deployments

Combine PQC with ECC: don't make systems less secure in the attempt to make them more secure against future quantum attackers!

- Cryptanalysis of PQ schemes is not as stable as for ECC
  - SIKE... (was deployed, **hybrid**, by Google and Cloudflare)
  - Late breaks of GeMSS and Rainbow
- Helps with Challenges $3 + 4$
  - Implementation attack against PQC does not break system
  - Improve implementation security over time
  - Mostly orthogonal to HNDL protection (?)

Hybrid deployments

Combine PQC with ECC: don't make systems less secure in the attempt to make them more secure against future quantum attackers!

- Cryptanalysis of PQ schemes is not as stable as for ECC
  - SIKE... (was deployed, **hybrid**, by Google and Cloudflare)
  - Late breaks of GeMSS and Rainbow
- Helps with Challenges $3 + 4$
  - Implementation attack against PQC does not break system
  - Improve implementation security over time
  - Mostly orthogonal to HNDL protection (?)

Broad consensus: PQC deployments today should be hybrid!

How about challenge 1 (performance)?

Computational complexity

- Today's systems use ECC
- ML-KEM is about as costly as ECC
- Hybrid costs about $2\times$ slowdown

How about challenge 1 (performance)?

Computational complexity

- Today's systems use ECC
- ML-KEM is about as costly as ECC
- Hybrid costs about $2\times$ slowdown
- Argument needs some more care with HW acceleration
- Anyway already have ECC
- Anyway will need PQC

How about challenge 1 (performance)?

Computational complexity

- Today's systems use ECC
- ML-KEM is about as costly as ECC
- Hybrid costs about $2\times$ slowdown
- Argument needs some more care with HW acceleration
- Anyway already have ECC
- Anyway will need PQC

Sizes

- PQC cryptographic objects are much bigger than for ECC
- X25519 PK: 32 B
- Additing 32 Bytes to 1KB makes almost no difference

Three approaches to “hybrididzation”

Protocol-level

- **+** Potential for optimal performance
- **+/-** Flexible choice of building blocks
- **-** High (per-protocol) analysis effort

Three approaches to “hybrididzation”

Protocol-level

- **+** Potential for optimal performance
- **+/-** Flexible choice of building blocks
- **-** High (per-protocol) analysis effort

Generic combiner

- **+** Low analysis effort (analyze *once*)
- **+/-** Flexible choice of building blocks
- **-** Computational overhead for being generic

Three approaches to “hybrididzation”

Protocol-level

- + Potential for optimal performance
- +/- Flexible choice of building blocks
- - High (per-protocol) analysis effort

Generic combiner

- + Low analysis effort (analyze *once*)
- +/- Flexible choice of building blocks
- - Computational overhead for being generic

Hybrid primitive

- + Low analysis effort (analyze *once*)
- + Close-to optimal performance
- +/- Cryptographically opinionated

We haven't even talked about signatures. . .

Same, same, but different. . .

- There is no HNDL attacks $\rightarrow$ migration less urgent for most systems
- Migration now for devices that are long time in the field, e.g.,
  - software updates for cars, motorcycles, etc.
  - secure boot on HW root of trust

We haven't even talked about signatures. . .

Same, same, but different. . .

- There is no HNDL attacks $\rightarrow$ migration less urgent for most systems
- Migration now for devices that are long time in the field, e.g.,
  - software updates for cars, motorcycles, etc.
  - secure boot on HW root of trust
- Performance even more challenging for Dilithium than for Kyber

We haven’t even talked about signatures. . .

Same, same, but different. . .

- There is no HNDL attacks $\rightarrow$ migration less urgent for most systems
- Migration now for devices that are long time in the field, e.g.,
  - software updates for cars, motorcycles, etc.
  - secure boot on HW root of trust
- Performance even more challenging for Dilithium than for Kyber
- May be worth re-designing protocols to avoid signatures, e.g.,
  - KEMTLS (avoid handshake signatures in TLS)
  - Merkle-tree certificates for TLS

So, what can/should you do now?

Inventory / CBOM

- Create and maintain overview of all used crypto primitives
- Idea: gamification to find the systems in the tail end

So, what can/should you do now?

Inventory / CBOM

- Create and maintain overview of all used crypto primitives
- Idea: gamification to find the systems in the tail end

Start migration

- Start with most critical systems (HNDL)
- Take the “easy wins”!

So, what can/should you do now?

Inventory / CBOM

- Create and maintain overview of all used crypto primitives
- Idea: gamification to find the systems in the tail end

Start migration

- Start with most critical systems (HNDL)
- Take the “easy wins”!

Join a consortium

- Multiple public/private initiatives to exchange experiences
- Please communicate lessons learned to the outside!

So, what can/should you do now?

Inventory / CBOM

- Create and maintain overview of all used crypto primitives
- Idea: gamification to find the systems in the tail end

Start migration

- Start with most critical systems (HNDL)
- Take the “easy wins”!

Join a consortium

- Multiple public/private initiatives to exchange experiences
- Please communicate lessons learned to the outside!

Do these things in parallel!

Learn more

- NIST PQC website:
  https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
- BSI recommendations:
  https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Quantentechnologien-und-Post-Quanten-Kryptografie/quantentechnologien-und-post-quanten-kryptografie_node.html
- EU migration roadmap:
  https://digital-strategy.ec.europa.eu/en/library/coordinated-implementation-roadmap-transition-post-quantum-cryptography
- NCCoE migration roadmap:
  https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms
- MITRE/PQCC migration roadmap:
  https://pqcc.org/post-quantum-cryptography-migration-roadmap/
- PQCA software resources:
  https://pqca.org/