Radboud University

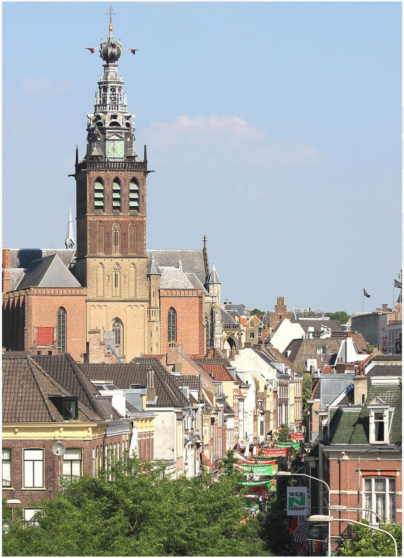
The transition to post-quantum cryptography

Peter Schwabe
peter@cryptojedi.org
https://cryptojedi.org

February 19, 2018

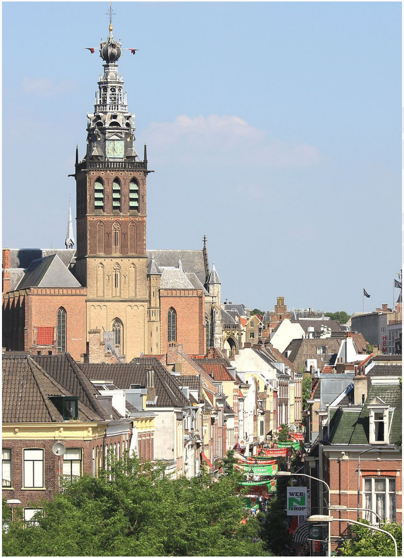
About me

- Assistant professor at Radboud University
- Working on high-speed high-security crypto software
- Last time in Nancy: 2012 (?)



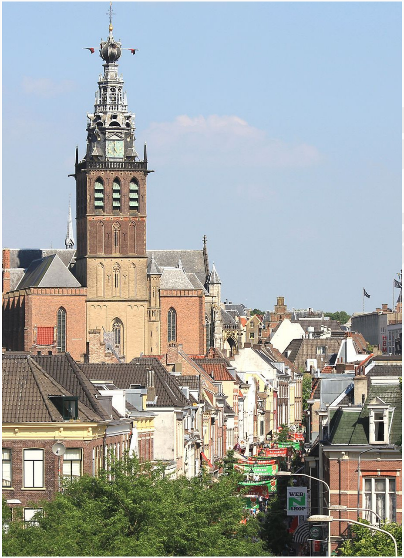
About me

- Assistant professor at Radboud University
- Working on high-speed high-security crypto software
- Last time in Nancy: 2012 (?)
- Typically speaking about crypto-software optimization
- Lot of assembly on slides. . .



About me

- Assistant professor at Radboud University
- Working on high-speed high-security crypto software
- Last time in Nancy: 2012 (?)
- Typically speaking about crypto-software optimization
- Lot of assembly on slides...
- This talk:
  - higher level
  - no assembly
  - most vague performance numbers I've ever used



Essential crypto today

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

Essential crypto today

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

Asymmetric crypto

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)

Essential crypto today

Symmetric crypto

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

Asymmetric crypto

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)

The asymmetric monoculture

- All widely deployed asymmetric crypto relies on
  - the **hardness of factoring**, or
  - the **hardness of (elliptic-curve) discrete logarithms**

. . . Shor, 1996

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.

Will there be quantum computers?

"In the past, people have said, maybe it's 50 years away, it's a dream, maybe it'll happen sometime. I used to think it was 50. Now I'm thinking like it's 15 or a little more. It's within reach. It's within our lifetime. It's going to happen."

—Mark Ketchen (IBM), Feb. 2012, about quantum computers

Definition

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

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 (so far, mainly PKE)

The NIST competition

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.

“Key exchange”

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(pk, sk) \leftarrow \text{KeyGen}()$
  - $(c, k) \leftarrow \text{Encaps}(pk)$
  - $k \leftarrow \text{Decaps}(c, sk)$

"Key exchange"

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(pk, sk) \leftarrow \text{KeyGen}()$
  - $(c, k) \leftarrow \text{Encaps}(pk)$
  - $k \leftarrow \text{Decaps}(c, sk)$

Status of the NIST competition

- In total 69 submissions accepted as "complete and proper"
- Several already broken
- 3 withdrawn

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in 5–7 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in 5–7 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect
- Message of this talk: **this is a terrible idea!**
- Would generate a generation of rather poor protocols
  - mediocre performance (designed pre-quantum, instantiated post-quantum)
  - Suboptimal security properties

NIST finalists as drop-in replacements?

- Can wait until NIST standardizes some algorithms in 5–7 years
- Plug these algorithms into existing protocols and systems
- My impression: that's what many systems designers expect
- Message of this talk: **this is a terrible idea!**
- Would generate a generation of rather poor protocols
  - mediocre performance (designed pre-quantum, instantiated post-quantum)
  - Suboptimal security properties
- Bad crypto is very hard to get rid of (think MD5)
- We probably have *one shot* to get this done properly
  - Systems *will* have to transition to PQ crypto
  - Let's work on getting the best out of this transition!
  - Requires interaction between cryptographers and systems designers

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters
- Performance:
  - All operations between 50 000 and 200 000 cycles
  - Keys and ciphertexts: 32 bytes
  - Signatures: 64 bytes

The starting point: ECC

- Today: build asymmetric crypto from elliptic-curve arithmetic
- Given P on a curve, $s \in \mathbb{Z}$, compute $Q = sP$
- ECDLP: hard to compute s, given P and Q
- Use for ECDH for key encapsulation and encryption
- Use for ECDSA or Schnorr signatures
- Use same curves, same parameters
- Performance:
  - All operations between 50 000 and 200 000 cycles
  - Keys and ciphertexts: 32 bytes
  - Signatures: 64 bytes
- Let's look at post-quantum candidates (at NIST security level 3)

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes
- "Classic McEliece" KEM NIST submission:
  - Encapsulation: $< 300\,000$ cycles
  - Decapsulation: $< 500\,000$ cycles
  - Key generation: billions of cycles
  - Cipher text: 226 bytes
  - Public key: $\approx 1\,\text{MB}$

PQ-KEMs, part 1: code-based

- Idea: Take error-correcting code for up to t errors
- Keep *decoding* algorithm secret
- Encryption: map message to code word, add t errors
- Most prominent example: McEliece (1978), uses binary Goppa codes
- "Classic McEliece" KEM NIST submission:
  - Encapsulation: $< 300\,000$ cycles
  - Decapsulation: $< 500\,000$ cycles
  - Key generation: billions of cycles
  - Cipher text: 226 bytes
  - Public key: ≈ 1 MB
- Probably good choice for, e.g., GPG, but not for low-latency applications
- Possible solution: use QCMDPC codes (NIST candidate "BIKE")
- Less studied, less conservative, problems with CCA security

PQ-KEMs, part 2: lattice-based

- Typically based on (variants of) LWE
- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$ and "noise distribution" χ
- Given samples $\mathbf{As} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$, find $\mathbf{s}$

PQ-KEMs, part 2: lattice-based

- Typically based on (variants of) LWE
- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$ and "noise distribution" χ
- Given samples $\mathbf{As} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$, find $\mathbf{s}$
- Only one LWE-based candidate: Frodo
  - All operations around 3 Mio cycles
  - Public key and ciphertext: $\approx$ 15 KB

PQ-KEMs, part 2: lattice-based

- Typically based on (variants of) LWE
- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$ and "noise distribution" χ
- Given samples $\mathbf{As} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$, find $\mathbf{s}$
- Only one LWE-based candidate: Frodo
  - All operations around 3 Mio cycles
  - Public key and ciphertext: $\approx$ 15 KB
- More than 20 lattice-based KEMS, others use structured lattices
- Typical performance:
  - All operations: $\approx$ 100 000 cycles
  - Public keys and ciphertexts: $\approx$ 1 KB

PQ-KEMs, part 2: lattice-based

- Typically based on (variants of) LWE
- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$ and "noise distribution" χ
- Given samples $\mathbf{As} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$, find $\mathbf{s}$
- Only one LWE-based candidate: Frodo
  - All operations around 3 Mio cycles
  - Public key and ciphertext: $\approx$ 15 KB
- More than 20 lattice-based KEMS, others use structured lattices
- Typical performance:
  - All operations: $\approx$ 100 000 cycles
  - Public keys and ciphertexts: $\approx$ 1 KB
- Structured lattices considered less conservative
- Many different design choices and tradeoffs

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding
- Rather young construction, more study needed
- Active attacks in 2016 by Galbraith, Petit, Shani, and Ti
- Secure SIDH (or SIKE) is **not** "analogous to the Diffie-Hellman key exchange"

PQ-KEMs, part 3: SIKE

- Started as "supersingular-isogeny Diffie-Hellman" (SIDH), Jao, De Feo, 2011
- Given two elliptic curves E, E' from the same isogeny class
- Find path of small isogenies from E to E'
- Security related to claw finding, but no reduction from claw finding
- Rather young construction, more study needed
- Active attacks in 2016 by Galbraith, Petit, Shani, and Ti
- Secure SIDH (or SIKE) is **not** "analogous to the Diffie-Hellman key exchange"
- SIKE performance:
  - Keygen: $\approx$ 30 Mio cycles
  - Encaps/Decaps: $\approx$ 50 Mio cycles each
  - Public key/ciphertext: $<$ 600 bytes each

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$
- Example: NIST candidate Gui
  - Signing: 1.7 billion cycles
  - Verification: $\approx 600\,000$ cycles
  - Signature: 63 bytes
  - Public key: almost 2 MB

PQ-Signatures, part 1: $\mathcal{MQ}$-based

- Find solution to system of m quadratic eqns in n variables over $\mathbb{F}_q$
- Additional assumption: attacker cannot exploit structure
- No reduction from $\mathcal{MQ}$
- Example: NIST candidate Gui
  - Signing: 1.7 billion cycles
  - Verification: $\approx$ 600 000 cycles
  - Signature: 63 bytes
  - Public key: almost 2 MB
- Can also construct signatures with reduction from $\mathcal{MQ}$
- Example: MQDSS
  - Signing $\approx$ 8.5 Mio cycles
  - Verification $\approx$ 5.8 Mio cycles
  - Signature: $\approx$ 40 KB
  - Public key: 72 bytes

PQ-Signatures, part 2: lattice-based

- Based on, e.g., LWE and SIS:
  - Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
  - Find nonzero $\mathbf{x} \in \mathbb{Z}^{\ell}$, s.t.: $\mathbf{A}\mathbf{x} = \mathbf{0} \in \mathbb{Z}_q^k$ and $\|\mathbf{x}\| < \beta$

PQ-Signatures, part 2: lattice-based

- Based on, e.g., LWE and SIS:
  - Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
  - Find nonzero $\mathbf{x} \in \mathbb{Z}^{\ell}$, s.t.: $\mathbf{Ax} = \mathbf{0} \in \mathbb{Z}_q^k$ and $\|\mathbf{x}\| < \beta$
- As for KEMs, typically use structured lattices
- Example: Dilithium
  - Signing: $\approx$ 1.8 Mio cycles
  - Verification: $\approx$ 400 000 cycles
  - Public key: $\approx$ 1.5 KB
  - Signature: $\approx$ 2.7 KB

PQ-Signatures, part 3: hash-based

- Start from (hash-based) one-time signatures
- Build Merkle trees and certification trees on top

PQ-Signatures, part 3: hash-based

- Start from (hash-based) one-time signatures
- Build Merkle trees and certification trees on top
- Two hash-based signatures in NIST PQC:
  - SPHINCS^{+}: more conservative, does not need collision resistance
  - Gravity-SPHINCS: performance optimized, requires collision resistance

PQ-Signatures, part 3: hash-based

- Start from (hash-based) one-time signatures
- Build Merkle trees and certification trees on top
- Two hash-based signatures in NIST PQC:
  - SPHINCS$^+$: more conservative, does not need collision resistance
  - Gravity-SPHINCS: performance optimized, requires collision resistance
- Many tradeoffs possible between
  - Speed (signing is generally slow)
  - Security (trivially via hash sizes)
  - Size (roughly 10-50KB)
  - Maxium number of signatures per key

PQ-Signatures, part 3: hash-based

- Start from (hash-based) one-time signatures
- Build Merkle trees and certification trees on top
- Two hash-based signatures in NIST PQC:
  - SPHINCS^+: more conservative, does not need collision resistance
  - Gravity-SPHINCS: performance optimized, requires collision resistance
- Many tradeoffs possible between
  - Speed (signing is generally slow)
  - Security (trivially via hash sizes)
  - Size (roughly 10-50KB)
  - Maxium number of signatures per key
- Example: SPHINCS
  - Signing: $\approx$ 52 Mio cycles
  - Verification: $\approx$ 1.5 Mio cycles
  - Signature: $\approx$ 40 KB
  - Public key is small
  - Up to 2^{50} signatures

Stateful signatures

- Hash-based signatures are already (almost) standardized
- XMSS standard draft submitted to IETF for conflict review
- Also highly parametrizable, for example:
  - Signing: $\approx$ 12.5 Mio cycles
  - Verification: $\approx$ 1 Mio cycles
  - Signature: $\approx$ 2.8 KB
  - Public key: 64 bytes
  - Up to 2^{20} signatures

Stateful signatures

- Hash-based signatures are already (almost) standardized
- XMSS standard draft submitted to IETF for conflict review
- Also highly parametrizable, for example:
  - Signing: $\approx$ 12.5 Mio cycles
  - Verification: $\approx$ 1 Mio cycles
  - Signature: $\approx$ 2.8 KB
  - Public key: 64 bytes
  - Up to 2^{20} signatures
- Issue with XMSS: it's stateful
- Security demands that secret key is updated for every signature
- Major problem, for examples, with backups

Stateful signatures

- Hash-based signatures are already (almost) standardized
- XMSS standard draft submitted to IETF for conflict review
- Also highly parametrizable, for example:
  - Signing: $\approx$ 12.5 Mio cycles
  - Verification: $\approx$ 1 Mio cycles
  - Signature: $\approx$ 2.8 KB
  - Public key: 64 bytes
  - Up to 2^{20} signatures
- Issue with XMSS: it's stateful
- Security demands that secret key is updated for every signature
- Major problem, for examples, with backups
- Stateful sigs are required for forward security
- XMSS gives forward security for free

Stateful signatures

- Hash-based signatures are already (almost) standardized
- XMSS standard draft submitted to IETF for conflict review
- Also highly parametrizable, for example:
  - Signing: $\approx$ 12.5 Mio cycles
  - Verification: $\approx$ 1 Mio cycles
  - Signature: $\approx$ 2.8 KB
  - Public key: 64 bytes
  - Up to 2^{20} signatures
- Issue with XMSS: it's stateful
- Security demands that secret key is updated for every signature
- Major problem, for examples, with backups
- Stateful sigs are required for forward security
- XMSS gives forward security for free
- **Start thinking systems with *stateful* signatures**

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not
- For software updates: probably not (?)

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not
- For software updates: probably not (?)
- Signed e-mails: probably not (?)

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not
- For software updates: probably not (?)
- Signed e-mails: probably not (?)
- For TLS (signed DH): yes, definitely!

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not
- For software updates: probably not (?)
- Signed e-mails: probably not (?)
- For TLS (signed DH): yes, definitely!
- But we don't need signed DH for authenticated key exchange!
- Can build AKE using CCA-secure KEMs only
- See the OPTLS proposal (Krawczyk, Wee, 2015)
- This is what systems like Signal, Noise already do
- Probably much better performance in a post-quantum world

Do we need fast signatures?

- Is it a problem if generating a signature takes, say, a second?
- If this is acceptable, can optimize, for size or verification speed
- For certificates (offline signing): probably not
- For software updates: probably not (?)
- Signed e-mails: probably not (?)
- For TLS (signed DH): yes, definitely!
- But we don't need signed DH for authenticated key exchange!
- Can build AKE using CCA-secure KEMs only
- See the OPTLS proposal (Krawczyk, Wee, 2015)
- This is what systems like Signal, Noise already do
- Probably much better performance in a post-quantum world
- **Design AKE from CCA-secure KEMs only**

Do we need fast key generation?

... **for KEMs**

- Potentially interesting tradeoffs between keygen and encapsulation time
- First intuition: ephemeral keys need fast keygen

Do we need fast key generation?

... **for KEMs**

- Potentially interesting tradeoffs between keygen and encapsulation time
- First intuition: ephemeral keys need fast keygen
- However: CCA-secure KEMs support key caching
- Asymptotically keygen cost becomes negligible

Do we need fast key generation?

... **for KEMs**

- Potentially interesting tradeoffs between keygen and encapsulation time
- First intuition: ephemeral keys need fast keygen
- However: CCA-secure KEMs support key caching
- Asymptotically keygen cost becomes negligible
- Requires modifications on protocol layer
- Cannot rely on protocol randomness from ephemeral keys anymore
- **Conclusion: desirable but not necessary?**

Do we need fast key generation?

… **for KEMs**

- Potentially interesting tradeoffs between keygen and encapsulation time
- First intuition: ephemeral keys need fast keygen
- However: CCA-secure KEMs support key caching
- Asymptotically keygen cost becomes negligible
- Requires modifications on protocol layer
- Cannot rely on protocol randomness from ephemeral keys anymore
- **Conclusion: desirable but not necessary?**

… **for Signatures**

- First intuition: keygen can be slow
- Second look: not terribly slow
- Smartcard producers need to generate lots of keys
- Not a *huge* concern, RSA keygen is slow already today

Decapsulation failures

- (Hashed) ECDH:
    - no failures for honest users
    - with some care: no advantage for dishonest users

Decapsulation failures

- (Hashed) ECDH:
  - no failures for honest users
  - with some care: no advantage for dishonest users
- Many post-quantum KEMs:
  - Tradeoff between security and failure probability
  - Failures reveal information about secret keys

Decapsulation failures

- (Hashed) ECDH:
  - no failures for honest users
  - with some care: no advantage for dishonest users
- Many post-quantum KEMs:
  - Tradeoff between security and failure probability
  - Failures reveal information about secret keys
- No problem for purely ephemeral keys (but need fast keygen!)
- Question: what failure probability is tolerable?
- Adam Langley: "2^{60} is fine"

Decapsulation failures

- (Hashed) ECDH:
  - no failures for honest users
  - with some care: no advantage for dishonest users
- Many post-quantum KEMs:
  - Tradeoff between security and failure probability
  - Failures reveal information about secret keys
- No problem for purely ephemeral keys (but need fast keygen!)
- Question: what failure probability is tolerable?
- Adam Langley: "2^{60} is fine"
- Obvious question: how about 2^{50}, 2^{40}, 2^{30}, ...?
- This is important to know for size/speed optimization

Do we want CPA-secure KEMs?

- With Diffie-Hellman we get CCA security “for free”
- Post-quantum KEMs need additional effort for CCA security

Do we want CPA-secure KEMs?

- With Diffie-Hellman we get CCA security "for free"
- Post-quantum KEMs need additional effort for CCA security
- Popular example Fujisaki-Okamoto transform:
  - Encapsulation generates all randomness deterministically from seed
  - Encrypt seed to recipient
  - Decapsulation recovers the seed, re-encrypts
  - Rejects if ciphertexts don't match

Do we want CPA-secure KEMs?

- With Diffie-Hellman we get CCA security "for free"
- Post-quantum KEMs need additional effort for CCA security
- Popular example Fujisaki-Okamoto transform:
    - Encapsulation generates all randomness deterministically from seed
    - Encrypt seed to recipient
    - Decapsulation recovers the seed, re-encrypts
    - Rejects if ciphertexts don't match
- Post-quantum CPA-secure KEMs
    - are simpler, smaller, faster
    - need additional primitives in protocols
    - are less robust (e.g., think key caching)

Do we want CPA-secure KEMs?

- With Diffie-Hellman we get CCA security "for free"
- Post-quantum KEMs need additional effort for CCA security
- Popular example Fujisaki-Okamoto transform:
  - Encapsulation generates all randomness deterministically from seed
  - Encrypt seed to recipient
  - Decapsulation recovers the seed, re-encrypts
  - Rejects if ciphertexts don't match
- Post-quantum CPA-secure KEMs
  - are simpler, smaller, faster
  - need additional primitives in protocols
  - are less robust (e.g., think key caching)
- **My intuition: only standardize/use CCA-secure KEMs**

We're going to miss DH

- Diffie-Hellman is extremely versatile:
- Can use it, for example, for **non-interactive key exchange (NIKE)**
  - Bob knows Alice' long-term public key A
  - Alice knows Bob's long-term public key B
  - They can each compute $k = h(A, B, aB) = h(A, B, bA)$
  - Used in various protocols, e.g., WireGuard

We're going to miss DH

- Diffie-Hellman is extremely versatile:
- Can use it, for example, for **non-interactive key exchange (NIKE)**
  - Bob knows Alice' long-term public key A
  - Alice knows Bob's long-term public key B
  - They can each compute $k = h(A, B, aB) = h(A, B, bA)$
  - Used in various protocols, e.g., WireGuard
- **Extremely expensive** to instantiate post-quantum
- Example: SIDH-based, Azarderakhsh, Jao, Leonardi, 2017:
  - Public key and ciphertext: $> 30\,\text{KB}$ each
  - Computing time: > 1 minute

We're going to miss DH

- Diffie-Hellman is extremely versatile:
- Can use it, for example, for **non-interactive key exchange (NIKE)**
  - Bob knows Alice' long-term public key A
  - Alice knows Bob's long-term public key B
  - They can each compute $k = h(A, B, aB) = h(A, B, bA)$
  - Used in various protocols, e.g., WireGuard
- **Extremely expensive** to instantiate post-quantum
- Example: SIDH-based, Azarderakhsh, Jao, Leonardi, 2017:
  - Public key and ciphertext: $> 30\,\text{KB}$ each
  - Computing time: > 1 minute
- **Conclusion: Design protocols that don't need NIKE**

A vision for a post-quantum-secure Internet

- Let's assume no major cryptanalytic breakthrough
- Agree on "the right spot" in the lattice-based KEM design space
- Agree on details of hash-based signatures

A vision for a post-quantum-secure Internet

- Let's assume no major cryptanalytic breakthrough
- Agree on "the right spot" in the lattice-based KEM design space
- Agree on details of hash-based signatures
- Build Internet crypto from
  - CCA-secure lattice-based KEM
  - Stateful and stateless hash-based signatures

A vision for a post-quantum-secure Internet

- Let's assume no major cryptanalytic breakthrough
- Agree on "the right spot" in the lattice-based KEM design space
- Agree on details of hash-based signatures
- Build Internet crypto from
  - CCA-secure lattice-based KEM
  - Stateful and stateless hash-based signatures
- Build AKE from KEMs alone, no signatures
- This means that certificates are signed KEM public keys

A vision for a post-quantum-secure Internet

- Let's assume no major cryptanalytic breakthrough
- Agree on "the right spot" in the lattice-based KEM design space
- Agree on details of hash-based signatures
- Build Internet crypto from
  - CCA-secure lattice-based KEM
  - Stateful and stateless hash-based signatures
- Build AKE from KEMs alone, no signatures
- This means that certificates are signed KEM public keys
- Use HSMs and smartcards for stateful, forward-secure signatures
- If impossible: fall back to slower, larger stateless signatures
- Carefully optimize hash-based signatures per application

Conclusion

- There are now 60+ concrete post-quantum schemes now
- There is software for all of those (which sometimes even works)
- Try them out, put them into systems, see what fails
- Start rethinking protocols and systems for a post-quantum world

Resources online

- Personal website:
  https://cryptojedi.org
- NIST PQC website:
  https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
- PQC Lounge:
  https://www.safecrypto.eu/pqclounge/
- NIST mailing list:
  https://www.safecrypto.eu/pqclounge/