Post-quantum cryptography

y^7 | Lorenz Panny

Technische Universität München

muCCC, Munich, 18 July 2025

Big picture: Cryptography

Important public-key systems

The impending(?) quantum apocalypse

Post-quantum cryptography (PQC)

Cryptography from lattices

Post-quantum elliptic-curve cryptography

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.
- All users have the *same capabilities*.
  For instance: encrypting *and* decrypting.

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.
- All users have the *same capabilities*.
  For instance: encrypting *and* decrypting.
- Hence, *symmetric* or *secret-key*.

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.
- All users have the *same capabilities*.
  For instance: encrypting *and* decrypting.
- Hence, *symmetric* or *secret-key*.

Public-key cryptography (since $\approx$1976):

- Keys are now pairs: a *private key* and a *public key*.

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.
- All users have the *same capabilities*.
  For instance: encrypting *and* decrypting.
- Hence, *symmetric* or *secret-key*.

Public-key cryptography (since ≈ 1976):

- Keys are now pairs: a *private key* and a *public key*.
- They give the respective owners *different capabilities*.

Two kinds of cryptography

"Classical" cryptography (for thousands of years):

- *Secret keys* exchanged in advance via a secure channel.
- All users have the *same capabilities*.
  For instance: encrypting *and* decrypting.
- Hence, *symmetric* or *secret-key*.

Public-key cryptography (since $\approx$1976):

- Keys are now pairs: a *private key* and a *public key*.
- They give the respective owners *different capabilities*.
- Hence, *public-key* or *asymmetric*.

Example: Digital signatures

Example: Digital signatures

- Alice uses her private key to sign a (digital) document.

Example: Digital signatures

- Alice uses her private key to sign a (digital) document.
- Anyone can verify the signature using Alice's public key.

Example: Digital signatures

- Alice uses her private key to sign a (digital) document.
- Anyone can verify the signature using Alice's public key.

This mimics the *intended* properties of a "real" signature.

Example: Public-key encryption

Example: Public-key encryption

- Anyone can use Bob's public key to encrypt a message

Example: Public-key encryption

- Anyone can use Bob's public key to encrypt a message such that only he can decrypt it using his private key.

Example: Public-key encryption

- Anyone can use Bob's public key to encrypt a message such that only he can decrypt it using his private key.

Analogy: An open padlock for which Bob has the key.

Hard problems

- By design, asymmetric cryptography is always breakable

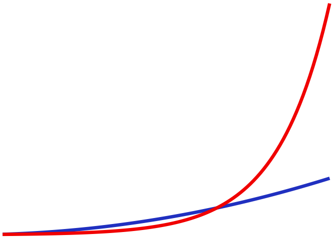
  —*at absurdly high costs*.

Hard problems

- By design, asymmetric cryptography is always breakable

  —*at absurdly high costs*.

- Security relies on computationally hard problems.

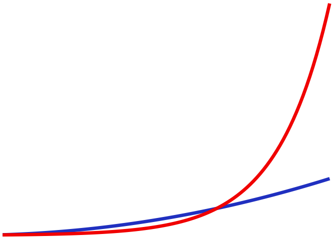


Hard problems

- By design, asymmetric cryptography is always breakable

  — *at absurdly high costs*.

- Security relies on computationally hard problems.



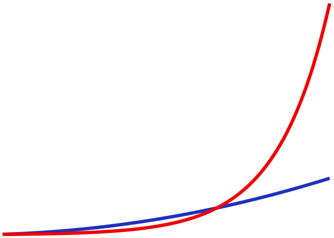
- Great source of hard problems: ***Algebra!***
  Finite fields, elliptic curves, number fields, class groups, ...

Hard problems

- By design, asymmetric cryptography is always breakable

  —*at absurdly high costs*.

- Security relies on computationally hard problems.



- Great source of hard problems: ***Algebra!***

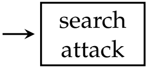
  Finite fields, elliptic curves, number fields, class groups, ...
  - Key feature: These objects have a lot of useful structure.
  - Sweet spot: just enough to make things *functional* but secure.

A cryptanalyst's life

Have: Supposedly hard computational problem.

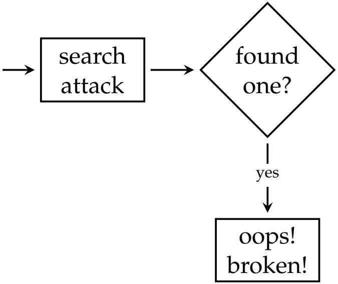
A cryptanalyst's life

Have: Supposedly hard computational problem.



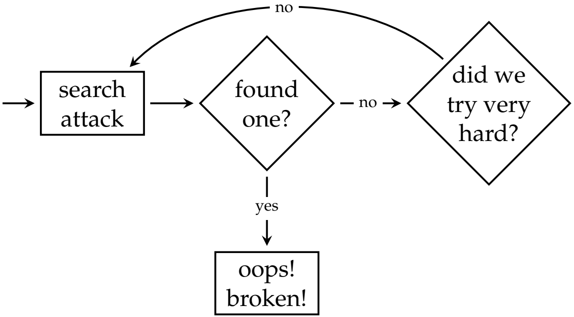
A cryptanalyst's life

Have: Supposedly hard computational problem.



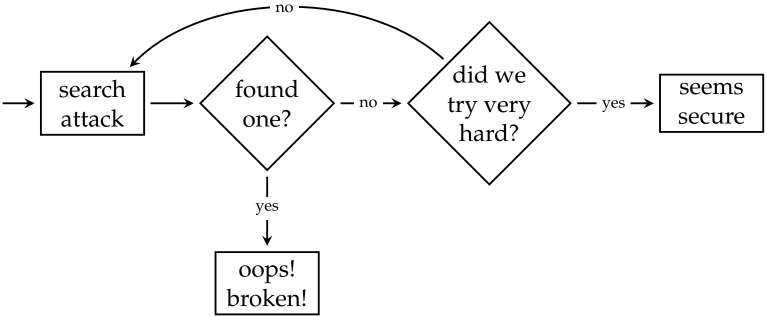
A cryptanalyst's life

Have: Supposedly hard computational problem.



A cryptanalyst's life

Have: Supposedly hard computational problem.



Big picture: Cryptography

Important public-key systems

The impending(?) quantum apocalypse

Post-quantum cryptography (PQC)

Cryptography from lattices

Post-quantum elliptic-curve cryptography

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot\,)$ and groups of elliptic-curve points over finite fields.

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot\,)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob
$a \leftarrow \{0, ..., q-1\}$		$b \leftarrow \{0, ..., q-1\}$

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob
$a \leftarrow \{0, \ldots, q-1\}$		$b \leftarrow \{0, \ldots, q-1\}$
$A := g^a$		$B := g^b$

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob
$a \leftarrow \{0, \ldots, q-1\}$		$b \leftarrow \{0, \ldots, q-1\}$
$A := g^a$		$B := g^b$

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob
$a \leftarrow \{0, \ldots, q-1\}$		$b \leftarrow \{0, \ldots, q-1\}$
$A := g^a$	$A \rightarrow$ Bob, $B \rightarrow$ Alice	$B := g^b$
$s := B^a$		$s := A^b$

Diffie–Hellman key exchange (1976)

Public parameters: A finite group G and an element $g \in G$ of (prime) order q.

Typical choices include $G \leq (\mathbb{F}_p \backslash \{0\}, \cdot)$ and groups of elliptic-curve points over finite fields.

Alice	public	Bob
$a \leftarrow \{0, \ldots, q-1\}$		$b \leftarrow \{0, \ldots, q-1\}$
$A := g^a$	$A \rightarrow$, $\leftarrow B$	$B := g^b$
$s := B^a$		$s := A^b$

$$B^a = (g^b)^a = g^{ab} = (g^a)^b = A^b.$$

Schnorr's identification protocol (1990ish / predecessor: ElGamal 1985)

- KeyGen(): Like Diffie–Hellman. Key pair is (a, A) where $A = g^a$.

Prover(a, A)		Verifier(A)
$r \leftarrow \{0, \ldots, q-1\}$		
"commitment" $R := g^r$	→	
	←	"challenge" $c \leftarrow \{0, \ldots, q-1\}$
"response" $s := (r - a \cdot c) \bmod q$	→	
		check $R \stackrel{?}{=} g^s \cdot A^c$

Schnorr's identification protocol (1990ish / predecessor: ElGamal 1985)

- KeyGen(): Like Diffie–Hellman. Key pair is (a, A) where $A = g^a$.

Prover(a, A)		Verifier(A)
$r \leftarrow \{0, \ldots, q-1\}$		
"commitment" $R := g^r$	→	
	←	"challenge" $c \leftarrow \{0, \ldots, q-1\}$
"response" $s := (r - a \cdot c) \bmod q$	→	
		check $R \stackrel{?}{=} g^s \cdot A^c$

Correctness: $g^s \cdot A^c = g^{r-a \cdot c} \cdot g^{a \cdot c} = g^r = R$.

Schnorr's identification protocol (1990ish / predecessor: ElGamal 1985)

- KeyGen(): Like Diffie–Hellman. Key pair is (a, A) where $A = g^a$.

Prover(a, A)		Verifier(A)
$r \leftarrow \{0, \ldots, q-1\}$		
"commitment" $R := g^r$	$\longrightarrow$	
	$\longleftarrow$	"challenge" $c \leftarrow \{0, \ldots, q-1\}$
"response" $s := (r - a \cdot c) \bmod q$	$\longrightarrow$	
		check $R \stackrel{?}{=} g^s \cdot A^c$

Correctness: $g^s \cdot A^c = g^{r-a \cdot c} \cdot g^{a \cdot c} = g^r = R$.

This can be transformed into a signature scheme.

RSA (1978)

RSA (1978)

- Private key: Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.

RSA (1978)

- <u>Private key:</u> Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.
- <u>Public key:</u> The product $n = p \cdot q$, an integer e.

RSA (1978)

- Private key: Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.
- Public key: The product $n = p \cdot q$, an integer e.
- "Encryption": $m \mapsto m^e \bmod n$.

RSA (1978)

- Private key: Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.
- Public key: The product $n = p \cdot q$, an integer e.
- "Encryption": $m \mapsto m^e \bmod n$.
- "Decryption": $c \mapsto c^d \bmod n$.

RSA (1978)

- Private key: Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.
- Public key: The product $n = p \cdot q$, an integer e.
- "Encryption": $m \mapsto m^e \bmod n$.
- "Decryption": $c \mapsto c^d \bmod n$.

Inverting $m \mapsto m^e \bmod n$ seems as hard as factoring n. $\smile$

RSA (1978)

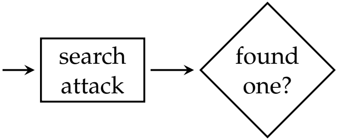
- Private key: Two large primes p, q.
  The modular inverse $d = e^{-1} \bmod \varphi$ where $\varphi = (p-1)(q-1)$.
- Public key: The product $n = p \cdot q$, an integer e.
- "Encryption": $m \mapsto m^e \bmod n$.
- "Decryption": $c \mapsto c^d \bmod n$.

Inverting $m \mapsto m^e \bmod n$ seems as hard as factoring n. :)

This is *Textbook RSA*, which is wildly insecure.
Proceed with caution, or do not proceed at all.

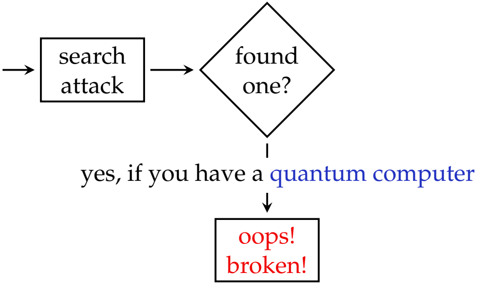
Enter quantum.

Have: Supposedly hard computational problem.



Enter quantum.

Have: Supposedly hard computational problem.



What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.

What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation "by hand": Interaction between brain and pen and paper.

What are computers, really?

- <u>Computing</u> essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation “by hand”: Interaction between brain and pen and paper.
  - 

What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation “by hand”: Interaction between brain and pen and paper.
  - Mechanical calculation device: Classical mechanics — gears etc.

What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation “by hand”: Interaction between brain and pen and paper.
  - Mechanical calculation device: Classical mechanics — gears etc.
  - Pocket calculator/laptop: Electronics of silicon-based semiconductors.

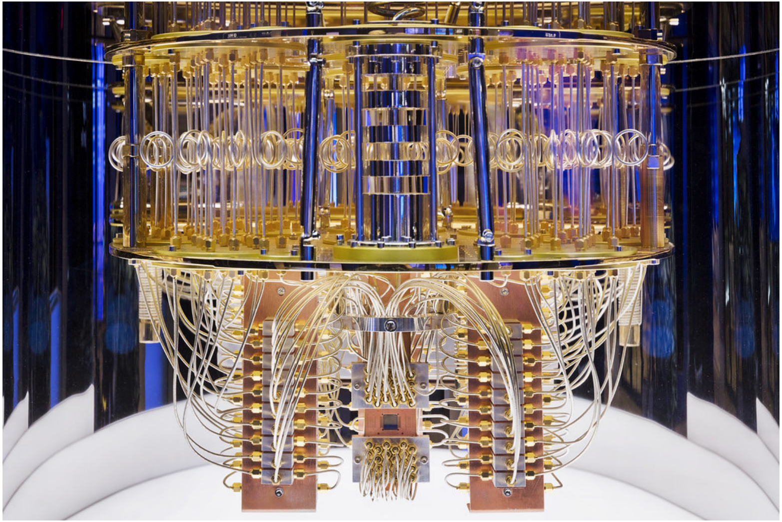
What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation “by hand”: Interaction between brain and pen and paper.
  - Mechanical calculation device: Classical mechanics — gears etc.
  - Pocket calculator/laptop: Electronics of silicon-based semiconductors.
  - Quantum computer: Quantum-mechanical properties of particles.

What are computers, really?

- Computing essentially means manipulating and exploiting real-world physical processes to find some desired answer.
  - Calculation "by hand": Interaction between brain and pen and paper.
  - Mechanical calculation device: Classical mechanics — gears etc.
  - Pocket calculator/laptop: Electronics of silicon-based semiconductors.
  - Quantum computer: Quantum-mechanical properties of particles.

$\rightsquigarrow$ Quantum computers are just "the next evolution" of using an increasingly bigger share of physics to compute things.



Quantum computing: Principle

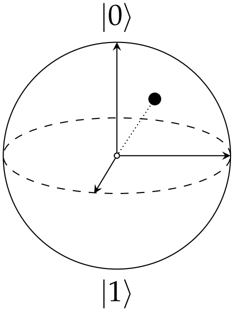
- “Normal” computer: Every bit always holds a value of *either* 0 *or* 1.

Quantum computing: Principle

- "Normal" computer: Every bit always holds a value of *either* 0 *or* 1.
- Quantum computer: "Qubits" can **in a certain well-defined way** hold a "mixture" of the values 0 *and* 1.

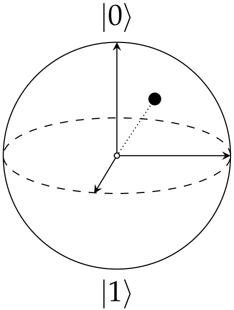
Quantum computing: Principle

- “Normal” computer: Every bit always holds a value of *either* 0 *or* 1.
- Quantum computer: “Qubits” can **in a certain well-defined way** hold a “mixture” of the values 0 *and* 1.



Quantum computing: Principle

- "Normal" computer: Every bit always holds a value of *either* 0 *or* 1.
- Quantum computer: "Qubits" can **in a certain well-defined way** hold a "mixture" of the values 0 *and* 1.



It is **not** true that *"quantum computers can simply try all keys in parallel"*.

Enter *post*-quantum. (1)

Common misconception:

«Quantum computers massively *speed up **all** computations*.

Enter *post*-quantum. (1)

Common misconception:

«Quantum computers massively *speed up **all** computations*.
Therefore, cryptography is doomed, and all hope is lost.»

Enter *post*-quantum. (1)

Common misconception:

«Quantum computers massively *speed up* ***all*** *computations*.
Therefore, cryptography is doomed, and all hope is lost.»

Not true at all!

Quantum computers struggle with plenty of tasks.

Enter *post*-quantum. (1)

Common misconception:

«Quantum computers massively *speed up* ***all*** *computations*.
Therefore, cryptography is doomed, and all hope is lost.»

Not true at all!

Quantum computers struggle with plenty of tasks.

Quantum cryptanalysis

Of primary relevance to cryptography are three algorithms:

Quantum cryptanalysis

Of primary relevance to cryptography are three algorithms:

- Grover's algorithm: Given a function $f : \{0,1\}^n \to \{0,1\}$ such that $\exists!\, x \in \{0,1\}^n$ with $f(x) = 1$, find that x.
  - **!!** Square-root complexity: from $O(2^n)$ to $O(2^{n/2})$.

Quantum cryptanalysis

Of primary relevance to cryptography are three algorithms:

- Grover's algorithm: Given a function $f : \{0,1\}^n \to \{0,1\}$ such that $\exists!\, x \in \{0,1\}^n$ with $f(x) = 1$, find that x.
  - **!!** Square-root complexity: from $O(2^n)$ to $O(2^{n/2})$.
- Shor's algorithm: Given a periodic function $f : \mathbb{Z}^r \to S$, find (a description of) the set of period vectors.
  - **!!** Polynomial-time complexity. (More on the next slide.)

Quantum cryptanalysis

Of primary relevance to cryptography are three algorithms:

- Grover's algorithm: Given a function $f : \{0,1\}^n \to \{0,1\}$ such that $\exists!\, x \in \{0,1\}^n$ with $f(x) = 1$, find that x.
  - !! Square-root complexity: from $O(2^n)$ to $O(2^{n/2})$.
- Shor's algorithm: Given a periodic function $f : \mathbb{Z}^r \to S$, find (a description of) the set of period vectors.
  - !! Polynomial-time complexity. (More on the next slide.)
- Kuperberg's algorithm: Given two functions $f_1, f_2 : G \to S$ such that $\exists!\, s \in G$ with $f_2(x) = f_1(x+s)$ for all x, find that s.
  - !! Subexponential complexity: from $|G|^{O(1)}$ to $2^{O\left(\sqrt{\log|G|}\right)}$.

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**. 💥

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**. 💥

Main <u>idea</u> (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**. 💥

Main idea (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

Shor can (among other things) compute the kernel of a map of the form

$$f\colon \quad (\mathbb{Z}^r, +) \twoheadrightarrow (G, \cdot\,), \quad (x_1, \ldots, x_r) \mapsto g_1^{x_1} \cdots g_r^{x_r}\,,$$

where $(G, \cdot\,)$ is a finite abelian group and $g_1, \ldots, g_r \in G$.

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**. 💥

Main idea (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

💡 Shor can (among other things) compute the kernel of a map of the form

$$f: \quad (\mathbb{Z}^r, +) \twoheadrightarrow (G, \cdot), \quad (x_1, ..., x_r) \mapsto g_1^{x_1} \cdots g_r^{x_r},$$

where $(G, \cdot)$ is a finite abelian group and $g_1, ..., g_r \in G$.

- Solving DLP $(g, h = g^x)$: Let $r = 2$ and $(g_1, g_2) = (g, h)$.

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**.

Main idea (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

Shor can (among other things) compute the kernel of a map of the form

$$f: \quad (\mathbb{Z}^r, +) \twoheadrightarrow (G, \cdot), \quad (x_1, \ldots, x_r) \mapsto g_1^{x_1} \cdots g_r^{x_r},$$

where $(G, \cdot)$ is a finite abelian group and $g_1, \ldots, g_r \in G$.

- Solving DLP $(g, h = g^x)$: Let $r = 2$ and $(g_1, g_2) = (g, h)$.
  Then $\ker(f)$ contains the vector $(x, -1)$.

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**.

Main idea (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

Shor can (among other things) compute the kernel of a map of the form

$$f\colon \quad (\mathbb{Z}^r, +) \twoheadrightarrow (G, \,\cdot\,), \quad (x_1, \dots, x_r) \mapsto g_1^{x_1} \cdots g_r^{x_r} \,,$$

where $(G, \,\cdot\,)$ is a finite abelian group and $g_1, \dots, g_r \in G$.

- Solving DLP $(g, h = g^x)$: Let $r = 2$ and $(g_1, g_2) = (g, h)$.
  Then $\ker(f)$ contains the vector $(x, -1)$.
- Factoring $n = pq$: Let $r = 1$ and $g_1 = \alpha$ be chosen at random from $(\mathbb{Z}/n)^\times$.

Shor's quantum algorithm

...can **solve DLP** (in any group) and **factor integers** in **polynomial time**.

Main idea (+ plenty of technical complications and optimization tricks):

Quantum period finding using the **quantum Fourier transform** (QFT).

Shor can (among other things) compute the kernel of a map of the form

$$f\colon \quad (\mathbb{Z}^r,+) \twoheadrightarrow (G,\,\cdot\,),\quad (x_1,\dots,x_r) \mapsto g_1^{x_1}\cdots g_r^{x_r}\,,$$

where $(G,\,\cdot\,)$ is a finite abelian group and $g_1,\dots,g_r \in G$.

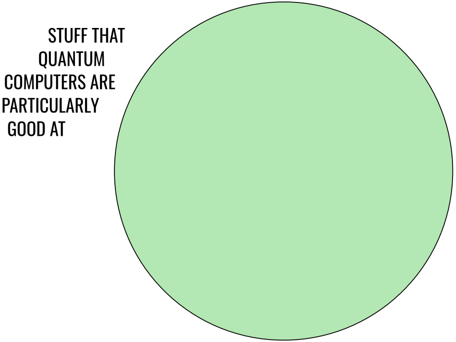
- Solving DLP $(g, h = g^x)$: Let $r = 2$ and $(g_1, g_2) = (g, h)$.
  Then $\ker(f)$ contains the vector $(x, -1)$.
- Factoring $n = pq$: Let $r = 1$ and $g_1 = \alpha$ be chosen at random from $(\mathbb{Z}/n)^\times$.
  Then $\ker(f) = \mathrm{ord}(\alpha)\mathbb{Z}$. (Exercise: With $\Pr \geq 1/2$, we get $\gcd(n, \alpha^{\mathrm{ord}(\alpha)/2} - 1) \in \{p, q\}$.)

Enter *post*-quantum. (2)

Unfortunate coincidence *(or is it?)*:

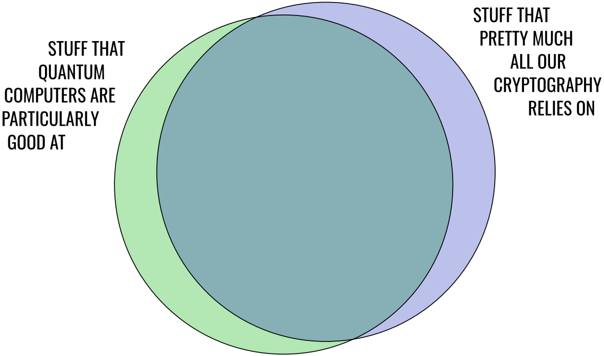
Enter *post*-quantum. (2)

Unfortunate coincidence *(or is it?)*:



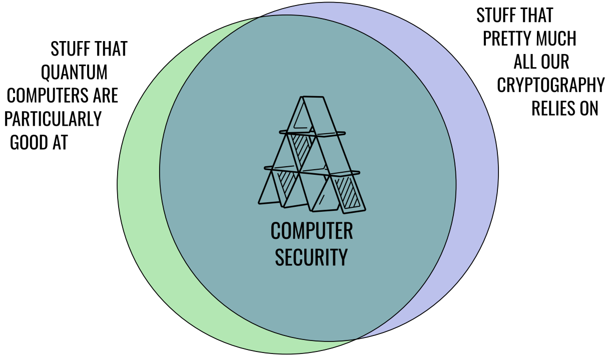
Enter *post*-quantum. (2)

Unfortunate coincidence *(or is it?)*:



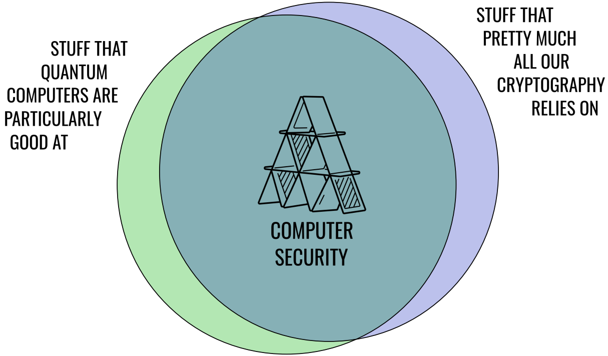
Enter *post*-quantum. (2)

Unfortunate coincidence *(or is it?)*:



Enter *post*-quantum. (2)

Unfortunate coincidence *(or is it?)*:



- Note: Public-key cryptography sustains much more damage from quantum attacks (due to Shor) than symmetric cryptography does (due to Grover).

  (For symmetric cryptography, doubling sizes is usually good enough (even conservative).)

Post-quantum cryptography (PQC)

...substitutes quantum-weak building blocks by

quantum-resistant alternatives.

Note on "quantum cryptography"

- Post-quantum cryptography is not to be confused with "quantum cryptography".

Note on "quantum cryptography"

- Post-quantum cryptography is not to be confused with "quantum cryptography".
- PQC runs on classical computers. Only the *attacker* is assumed to have a quantum computer.

Note on "quantum cryptography"

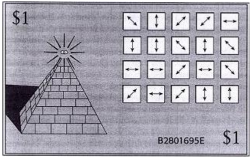
- Post-quantum cryptography is not to be confused
  with "quantum cryptography".

- PQC runs on classical computers.
  Only the *attacker* is assumed to have a quantum computer.

- In quantum cryptography, all users need quantum devices!

Note on "quantum cryptography"

- Post-quantum cryptography is <u>not to be confused</u> with "quantum cryptography".
- PQC runs on classical computers.
  <u>Only the *attacker*</u> is assumed to have a quantum computer.
- In quantum cryptography, all users need quantum devices!



Note on "quantum cryptography"

Position Paper on Quantum Key Distribution

French Cybersecurity Agency (ANSSI)

Federal Office for Information Security (BSI)

Netherlands National Communications Security Agency (NLNCSA)

Swedish National Communications Security Authority, Swedish Armed Forces

Note on “quantum cryptography”

Executive summary

Quantum Key Distribution (QKD) seeks to leverage quantum effects in order for two remote parties to agree on a secret key via an insecure quantum channel. This technology has received significant attention, sometimes claiming unprecedented levels of security against attacks by both classical and quantum computers.

Due to current and inherent limitations, QKD can however currently only be used in practice in some niche use cases. For the vast majority of use cases where classical key agreement schemes are currently used it is not possible to use QKD in practice. Furthermore, QKD is not yet sufficiently mature from a security perspective. In light of the urgent need to stop relying only on quantum-vulnerable public-key cryptography for key establishment, the clear priorities should therefore be the migration to post-quantum cryptography and/or the adoption of symmetric keying.

This paper is aimed at a general audience. Technical details have therefore been left out to the extent possible. Technical terms that require a definition are printed in italics and are explained in a glossary at the end of the document.

The post-quantum zoo

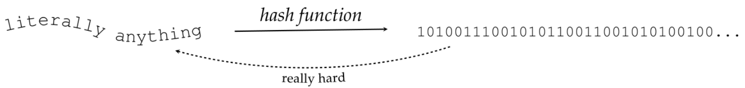
- PQC uses alternative hardness assumptions
  based on various (exciting!) types of mathematics.

The post-quantum zoo

- PQC uses alternative hardness assumptions based on various (exciting!) types of mathematics.

Hash-based signatures

Hash functions are random-looking functions that compress arbitrary data to short bitstrings. They should be hard to invert.



An individual can tie a hash value to their identity and later identify themself by revealing the corresponding input.
Selectively revealing inputs depending on a message leads to a signature scheme.

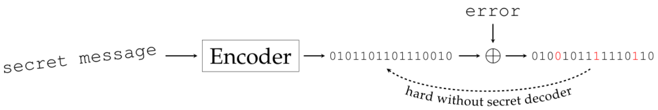
The post-quantum zoo

- PQC uses alternative hardness assumptions
  based on various (exciting!) types of mathematics.

Code-based crypto

Main application: **Encryption**.

Underlying problem: Correct errors in a codeword of a random-looking code.



Oldest proposal: McEliece 1978. Still *essentially unbroken*.

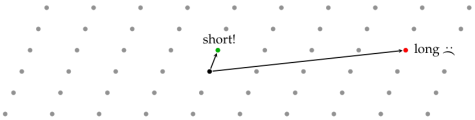
The post-quantum zoo

- PQC uses alternative hardness assumptions based on various (exciting!) types of mathematics.

Lattice-based crypto

Main applications: **Encryption**, **signatures**, and beyond.

Underlying problem: Find short vectors in a discrete additive subgroup of $\mathbb{R}^n$.



The post-quantum zoo

- PQC uses alternative hardness assumptions
  based on various (exciting!) types of mathematics.

Multivariate crypto

Main application: **Signatures**.

Underlying problem: Solve systems of quadratic equations over a finite field.

$$\begin{aligned}
10x^2 + 15z^2 + 19xy + 7xz + 27yz + 20x + y &\equiv 14 \pmod{31}\\
25x^2 + 30y^2 + 17z^2 + 30xy + 23xz + 27yz + 15x + 4y + 16z &\equiv 5 \pmod{31}\\
15x^2 + 9y^2 + 11z^2 + 18xy + 24xz + 16yz + 28x + 9y + 3z &\equiv 6 \pmod{31}\\
27x^2 + 10y^2 + 17z^2 + 7xz + 28yz + 4x + 13y + 27z &\equiv 12 \pmod{31}
\end{aligned}$$

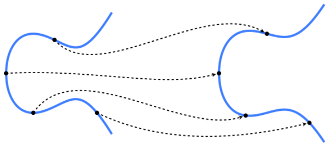
The post-quantum zoo

- PQC uses alternative hardness assumptions
  based on various (exciting!) types of mathematics.

Isogeny-based crypto

Main applications: **Key exchange**, **signatures**.

Underlying problem: Find an isogeny between two elliptic curves.
An *isogeny* is a surjective group homomorphism given by rational functions.



Shortcomings of PQC

Shortcomings of PQC

The good news:
There *are* plausible PQC replacements for most cryptographic functionality.

Shortcomings of PQC

The good news:
There *are* plausible PQC replacements for most cryptographic functionality.

The bad news: PQC is typically slower, bigger, less flexible, or all of these things.

Shortcomings of PQC

The <u>good news</u>:
There *are* plausible PQC replacements for most cryptographic functionality.

The <u>bad news</u>: PQC is typically slower, bigger, less flexible, or all of these things.

pre-quantum	post-quantum
6ee2da7b68b7a997e062d09d94c1c76de61b5c260a35273713ddcc29e09ac840	45c83435071624067d69587335b97bf564929709c8825a004b028ae09c40980a 07e8d4bd604527ee221e8bac67d34cbe762c26df8453aae8b8c82b59c51a8552 6aba8ddc4b5f63cf69a5b367d3153e460f497a209c495fca318862d6a5780086 5479a006012d82f7212b40284d310e01bcb11e122c1fd303e441807849a7ea47 976a99abb7ccc4b674ad66f68eca195789b277d23c3d67bc418ca7c908b21e53 984983ba0205e4689000ace97238b3699016fa95e7a3a59cec0be81363852756 2fa9bf10d715e7505f6e1c1433521a918a7df52760a0d8a9549569f10827c423 cddff82aae01a90111395487b9c82b7b5a7978d789679e66b75087bfbff0569f c94e94f93531b721315926388431f2a36ae0f701bac254befb437c58641d4560 c8738a98f30918945db0a6900ad2c2abfc3e0f4786a4555639d84dcdd031d8f0 508d8c774d68298bcac4f42c6a7ff585af491fa7d7c3bbb41727699ebb315c43 7b210d42626ebc66c916af1f3515374314e4f40309ca7289c7bc51c301d8180e dc792d4dd44c41b77bd47a972d8434a9f03bb3954236ec422be0c8e991a79af2 86b6a7c459a95ed44868ed8052f2db0f3741710228979507cff961564882b5ea 19515ee00d657c7141e9b05f9a24136a2f915620b664404b5397cc7842748973 d0716cc273b528d51383a63fc8a3c4a3b1a8bc965775d750add6996c929e29f4 1e42362a759baa76f5a3dc0552f1d83195960e45837901494a87f2a6dc3b5d8b 73a9695c1229a0c9bddb0b2d99aa350c6cac657745c1308af354e10595f3682a 34dc26d9d28e2e2c4634aca75e94384700c9c06b1bca348330ac1791fab14190 99cf1288283bab03dca09ab3593cf3b12739cb44c0c04c6b93d1ea831df6bcb8 807aa6aa8cbec64d749a9e47f851c47c6537e196f1fcc4d63b67d29a58e86b9a 72a199cbb793c5084e5bab20bd02289b4aaa64e4c119488531e8a651a3175014 8e1742c5390bb9995c123f3056ad44c476468ded4b88a49130e35b4b00803dd2 4718674ca708e436d5c15ee1d95367c623512653c83b27b41cb308f8c2929b19 3b5487a4ce6401ec27a1605f879e2d9c53bf27e165246401cad7840a077934b8

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

- Two tracks: Key exchange & signatures.

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

- Two tracks: Key exchange & signatures.
- Five target security levels, defined as "at least as hard to break as AES-128/SHA256/AES-192/SHA384/AES-256", respectively.

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

- Two tracks: Key exchange & signatures.
- Five target security levels, defined as "at least as hard to break as AES-128/SHA256/AES-192/SHA384/AES-256", respectively.
- In 2024, four algorithms selected:
  - "ML-KEM", a.k.a. "Kyber". Lattice-based key exchange.
  - "ML-DSA", a.k.a. "Dilithium". Lattice-based signature.
  - "SLH-DSA", a.k.a. "SPHINCS". (Stateless) hash-based signature.
  - "FN-DSA", a.k.a. "Falcon". Another lattice-based signature.

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

- Two tracks: Key exchange & signatures.
- Five target security levels, defined as "at least as hard to break as AES-128/SHA256/AES-192/SHA384/AES-256", respectively.
- In 2024, four algorithms selected:
  - "ML-KEM", a.k.a. "Kyber". Lattice-based key exchange.
  - "ML-DSA", a.k.a. "Dilithium". Lattice-based signature.
  - "SLH-DSA", a.k.a. "SPHINCS". (Stateless) hash-based signature.
  - "FN-DSA", a.k.a. "Falcon". Another lattice-based signature.
- In 2025, another algorithm selected:
  - "HQC". Code-based key exchange.

NISTPQC

Since 2016, the USA's **N**ational **I**nstitute for **S**tandards and **T**echnology has been running a standardization effort for post-quantum cryptography.

- Two tracks: Key exchange & signatures.
- Five target security levels, defined as "at least as hard to break as AES-128/SHA256/AES-192/SHA384/AES-256", respectively.
- In 2024, four algorithms selected:
  - "ML-KEM", a.k.a. "Kyber". Lattice-based key exchange.
  - "ML-DSA", a.k.a. "Dilithium". Lattice-based signature.
  - "SLH-DSA", a.k.a. "SPHINCS". (Stateless) hash-based signature.
  - "FN-DSA", a.k.a. "Falcon". Another lattice-based signature.
- In 2025, another algorithm selected:
  - "HQC". Code-based key exchange.

Note: "Key exchange" refers to **K**ey **E**ncapsulation **M**echanisms, essentially public-key encryption schemes that can only encrypt symmetric secret keys (but a priori not arbitrary messages).

(In particular, "key exchange" does not provide the interface of pre-quantum DH.)

Kyber: Numbers

Kyber-512

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (avx2)	
sk:	1632	gen:	122684	gen:	33856
pk:	800	enc:	154524	enc:	45200
ct:	768	dec:	187960	dec:	34572

...

Kyber-1024

Sizes (in bytes)		Haswell cycles (ref)		Haswell cycles (avx2)	
sk:	3168	gen:	307148	gen:	73544
pk:	1568	enc:	346648	enc:	97324
ct:	1568	dec:	396584	dec:	79128

Source: `https://pq-crystals.org/kyber/`

Dilithium: Numbers

Dilithium2

Sizes (in bytes)		Skylake cycles (ref)		Skylake cycles (avx2)	
		gen:	300751	gen:	124031
pk:	1312	sign:	1355434	sign:	333013
sig:	2420	verify:	327362	verify:	118412

...

Dilithium5

Sizes (in bytes)		Skylake cycles (ref)		Skylake cycles (avx2)	
sk:		gen:	819475	gen:	298050
pk:	2592	sign:	2856803	sign:	642192
sig:	4595	verify:	871609	verify:	279936

Source: `https://pq-crystals.org/dilithium/`

SPHINCS: Sizes

	public key size	secret key size	signature size
SPHINCS$^+$-128s	32	64	7 856
SPHINCS$^+$-128f	32	64	17 088
SPHINCS$^+$-192s	48	96	16 224
SPHINCS$^+$-192f	48	96	35 664
SPHINCS$^+$-256s	64	128	29 792
SPHINCS$^+$-256f	64	128	49 856

Table 8: Key and signature sizes in bytes

Source: `https://sphincs.org/data/sphincs+-round3-submission-nist.zip`

SPHINCS: Speed

	key generation	signing	verification
SPHINCS$^+$-SHA-256-128s-simple	84 964 790	644 740 090	861 478
SPHINCS$^+$-SHA-256-128s-robust	175 257 460	1 328 848 352	1 827 104
SPHINCS$^+$-SHA-256-128f-simple	1 334 220	33 651 546	2 150 290
SPHINCS$^+$-SHA-256-128f-robust	2 748 026	68 541 846	4 801 338
SPHINCS$^+$-SHA-256-192s-simple	125 310 788	1 246 378 060	1 444 030
SPHINCS$^+$-SHA-256-192s-robust	260 903 972	2 517 396 082	3 103 732
SPHINCS$^+$-SHA-256-192f-simple	1 928 970	55 320 742	3 492 210
SPHINCS$^+$-SHA-256-192f-robust	4 063 066	113 484 456	7 552 358
SPHINCS$^+$-SHA-256-256s-simple	80 943 202	1 025 721 040	1 986 974
SPHINCS$^+$-SHA-256-256s-robust	339 101 780	3 912 132 754	8 294 732
SPHINCS$^+$-SHA-256-256f-simple	5 067 546	109 104 452	3 559 052
SPHINCS$^+$-SHA-256-256f-robust	21 327 470	435 984 168	14 938 510

Table 6: Runtime benchmarks for SPHINCS$^+$-SHA-256 on AVX2

Source: `https://sphincs.org/data/sphincs+-round3-submission-nist.zip`

Summary

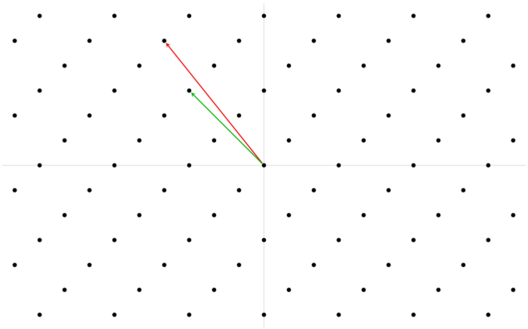
Cryptography will be okay,
but more expensive than before.

Summary

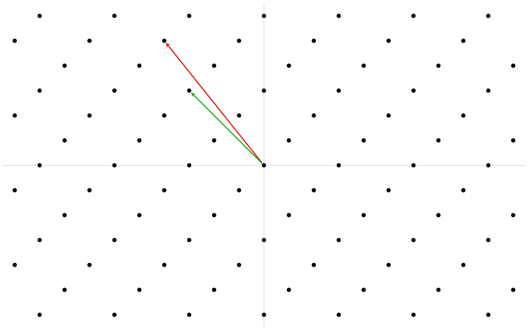
Cryptography will be okay, but more expensive than before.

General theme: You can have speeds $\approx$ comparable to pre-quantum ECC, or sizes $\approx$ comparable to pre-quantum ECC, but not at the same time. $\frown$

(Euclidean) lattices



(Euclidean) lattices



A (Euclidean) lattice of dimension n is a subset of $\mathbb{R}^m$ of the form

$$\Lambda = \{ v \cdot B \mid v \in \mathbb{Z}^n \},$$

where $B \in \mathbb{R}^{n \times m}$ is a full-rank matrix. We call B a basis matrix of Λ.

(In other words, Λ is the set of $\mathbb{Z}$-linear combinations of the rows of B.)

Essential lattice problems

Essential lattice problems

The **approximate** **shortest-vector problem** $\mathsf{SVP}_\gamma(\Lambda)$ is:

> Given a basis matrix B of Λ and an "approximation factor" $\gamma \geq 1$,
> find a vector $s \in \Lambda$ such that $\|s\| \leq \gamma \cdot \min_{v \in \Lambda \setminus \{0\}} \|v\|$.

Throughout, let $\lambda_1(\Lambda) = \min_{v \in \Lambda \setminus \{0\}} \|v\|$ denote the length of a shortest (nonzero) vector in Λ.

Essential lattice problems

The **approximate shortest-vector problem** $\mathsf{SVP}_\gamma(\Lambda)$ is:

Given a basis matrix B of Λ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s\| \leq \gamma \cdot \min_{v \in \Lambda \setminus \{0\}} \|v\|$.

Throughout, let $\lambda_1(\Lambda) = \min_{v \in \Lambda \setminus \{0\}} \|v\|$ denote the length of a shortest (nonzero) vector in Λ.

The **approximate closest-vector problem** $\mathsf{CVP}_\gamma(\Lambda, t)$ is:

Given a basis matrix B of Λ, a vector $t \in \mathbb{R}^m$ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s - t\| \leq \gamma \cdot \min_{v \in \Lambda} \|v - t\|$.

Essential lattice problems

The **approximate shortest-vector problem** $\mathsf{SVP}_\gamma(\Lambda)$ is:

Given a basis matrix B of Λ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s\| \leq \gamma \cdot \min_{v \in \Lambda \setminus \{0\}} \|v\|$.

Throughout, let $\lambda_1(\Lambda) = \min_{v \in \Lambda \setminus \{0\}} \|v\|$ denote the length of a shortest (nonzero) vector in Λ.

The **approximate closest-vector problem** $\mathsf{CVP}_\gamma(\Lambda, t)$ is:

Given a basis matrix B of Λ, a vector $t \in \mathbb{R}^m$ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s - t\| \leq \gamma \cdot \min_{v \in \Lambda} \|v - t\|$.

For random lattices and "small-ish" γ, these problems are hard as $n \to \infty$.

Essential lattice problems

The **approximate shortest-vector problem** $\mathsf{SVP}_\gamma(\Lambda)$ is:

Given a basis matrix B of Λ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s\| \leq \gamma \cdot \min_{v \in \Lambda \setminus \{0\}} \|v\|$.

Throughout, let $\lambda_1(\Lambda) = \min_{v \in \Lambda \setminus \{0\}} \|v\|$ denote the length of a shortest (nonzero) vector in Λ.

The **approximate closest-vector problem** $\mathsf{CVP}_\gamma(\Lambda, t)$ is:

Given a basis matrix B of Λ, a vector $t \in \mathbb{R}^m$ and an "approximation factor" $\gamma \geq 1$,
find a vector $s \in \Lambda$ such that $\|s - t\| \leq \gamma \cdot \min_{v \in \Lambda} \|v - t\|$.

For random lattices and "small-ish" γ, these problems are hard as $n \to \infty$.

There are *many* variants of these problems: Most importantly, "promise versions"
(SVP/CVP $\to$ uSVP/BDD) guarantee that an unusually short/close solution exists.

Lattice(-basis) reduction

Lattice problems in practice are almost always solved using lattice reduction.

Lattice(-basis) reduction

Lattice problems in practice are almost always solved using lattice reduction.

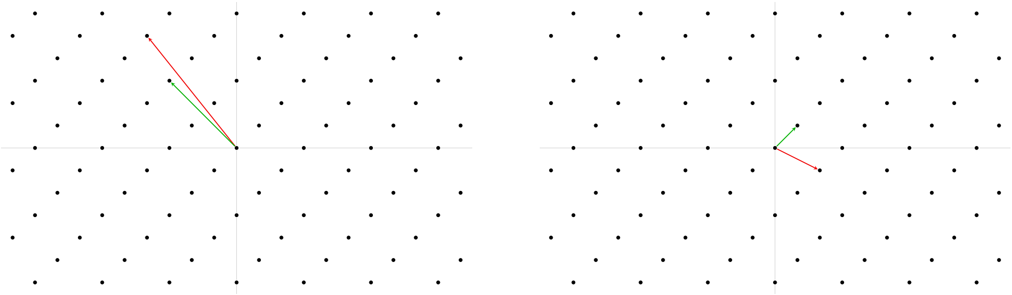
Recall:
The "quality" of the basis impacts the hardness of all kinds of lattice problems.

Lattice(-basis) reduction

Lattice problems in practice are almost always solved using lattice reduction.

Recall:
The "quality" of the basis impacts the hardness of all kinds of lattice problems.

General theme: The (1) **shorter** and (2) **closer to orthogonal** a basis is, the better.



The blueprint

...for **lattice-based cryptography** is as follows:

The blueprint

...for **lattice-based cryptography** is as follows:

- The private key is a "good" basis of a lattice Λ.

The blueprint

...for **lattice-based cryptography** is as follows:

- The private key is a "good" basis of a lattice Λ.
- The public key is a "bad" basis of Λ.

The blueprint

...for **lattice-based cryptography** is as follows:

- The private key is a “good” basis of a lattice Λ.
- The public key is a “bad” basis of Λ.
- The goal for an attacker is to solve a hard lattice problem in Λ.

The blueprint

...for **lattice-based cryptography** is as follows:

- The private key is a "good" basis of a lattice Λ.
- The public key is a "bad" basis of Λ.
- The goal for an attacker is to solve a hard lattice problem in Λ.
- The private-key holder can solve those problems using the good basis.

An encryption scheme (à la GGH '97)

- <u>KeyGen():</u> Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.

An encryption scheme (à la GGH '97)

- KeyGen(): Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.
- Encrypt(m, B'): View m as a vector in $\mathbb{Z}^n$ and define the ciphertext as $c := mB' + e$, where e is a small "error vector".

An encryption scheme (à la GGH '97)

- KeyGen(): Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.

- Encrypt(m, B'): View m as a vector in $\mathbb{Z}^n$ and define the ciphertext as $c := mB' + e$, where e is a small "error vector".

- Decrypt(c, B): Using B, find the vector $c - e = mB' \in \Lambda$. Compute $m := (c - e)B'^{-1}$.

An encryption scheme (à la GGH '97)

- KeyGen(): Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.

- Encrypt(m, B'): View m as a vector in $\mathbb{Z}^n$ and define the ciphertext as $c := mB' + e$, where e is a small "error vector".

- Decrypt(c, B): Using B, find the vector $c - e = mB' \in \Lambda$. Compute $m := (c - e)B'^{-1}$.

This scheme can really only encrypt random messages, and great care must be taken when sampling B' and e, else this is totally broken.

A signature scheme (à la GGH '97)

- <u>KeyGen():</u> Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.

A signature scheme (à la GGH '97)

- <u>KeyGen():</u> Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.
- <u>Sign(m, B):</u> Let $t := H(m) \in \mathbb{R}^m$ and compute the signature $s \in \Lambda$ as a lattice vector close to the hash t. <u>Example:</u> If $m = n$ one could set $s := \lfloor tB^{-1} \rceil B$, but this is very broken. ☺

A signature scheme (à la GGH '97)

- KeyGen(): Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.

- Sign(m, B): Let $t := H(m) \in \mathbb{R}^m$ and compute the signature $s \in \Lambda$ as a lattice vector close to the hash t. Example: If $m = n$ one could set $s := \lfloor tB^{-1} \rceil B$, but this is very broken. :)

- Verify(m, s, B'): Ensure $s \in \Lambda$. Let $t := H(m)$ and check that $\|s - t\|$ is small.

A signature scheme (à la GGH '97)

- KeyGen(): Sample a "good" basis B, defining a lattice Λ, and compute a "bad" basis B' of the same lattice. The private key is B, the public key is B'.
- Sign(m, B): Let $t := H(m) \in \mathbb{R}^m$ and compute the signature $s \in \Lambda$ as a lattice vector close to the hash t. Example: If $m = n$ one could set $s := \lfloor tB^{-1} \rceil B$, but this is very broken. ☺
- Verify(m, s, B'): Ensure $s \in \Lambda$. Let $t := H(m)$ and check that $\|s - t\|$ is small.

⚠ Great care must be taken when sampling s, else this is totally broken.

Real-world lattice-based cryptography

...works with lattices defined by linear systems of equations over $\mathbb{Z}/q$.

Real-world lattice-based cryptography

...works with lattices defined by linear systems of equations over $\mathbb{Z}/q$.

They are a convenient choice for cryptography since they are easy to generate and allow us to work with integers of bounded size.

Big picture: Cryptography

Important public-key systems

The impending(?) quantum apocalypse

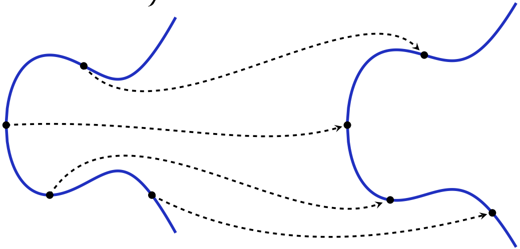
Post-quantum cryptography (PQC)

Cryptography from lattices

Post-quantum elliptic-curve cryptography

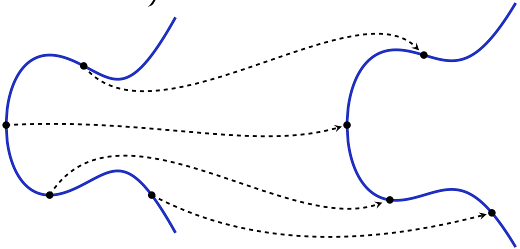
Isogenies of elliptic curves

- ...are essentially just *nice maps* between elliptic curves.

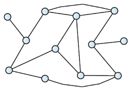


Isogenies of elliptic curves

- ...are essentially just *nice maps* between elliptic curves.

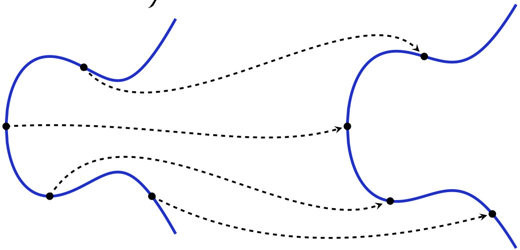


- They are a source of *exponentially large graphs*.

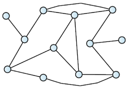


Isogenies of elliptic curves

- ...are essentially just *nice maps* between elliptic curves.

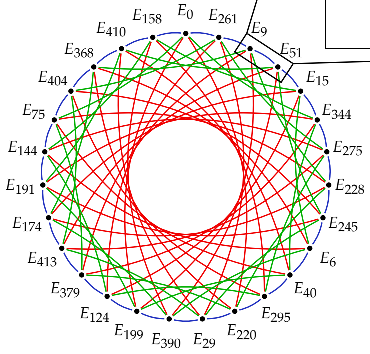


- They are a source of *exponentially large graphs*.

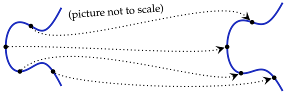


- ...with enough structure to *navigate meaningfully*!

Graphs of elliptic curves



A 3-isogeny



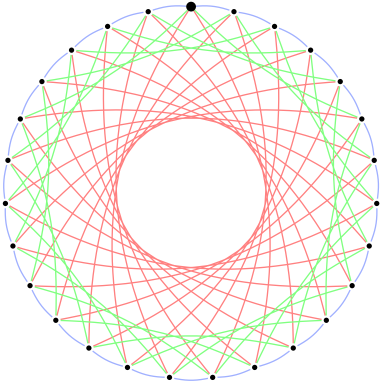
$$E_{51}\colon y^2 = x^3 + 51x^2 + x \longrightarrow E_9\colon y^2 = x^3 + 9x^2 + x$$

$$(x, y) \longmapsto \left(\frac{97x^3 - 183x^2 + x}{x^2 - 183x + 97},\; y \cdot \frac{133x^3 + 154x^2 - 5x + 97}{-x^3 + 65x^2 + 128x - 133} \right)$$

CSIDH [ˈsiːˌsaɪd] key exchange

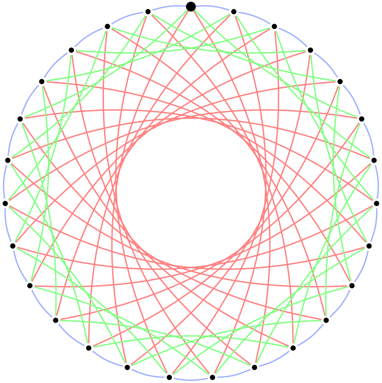
Alice

[+, +, −, −]



Bob

[−, +, −, −]



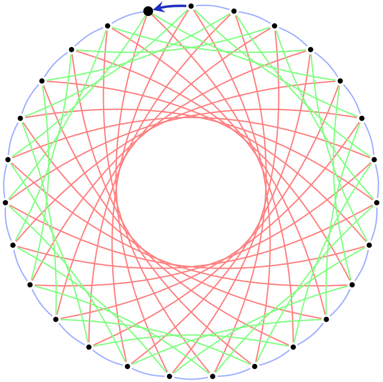
CSIDH [ˈsiːˌsaɪd] key exchange

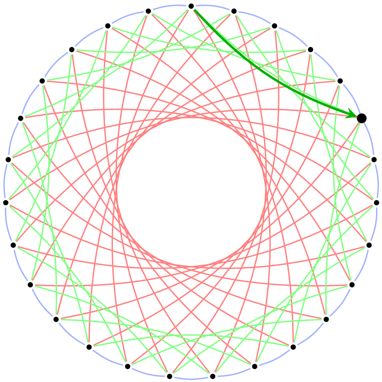
Alice

[+, +, −, −]
↑

Bob

[−, +, −, −]
↑





CSIDH [ˈsiːˌsaɪd] key exchange

Alice

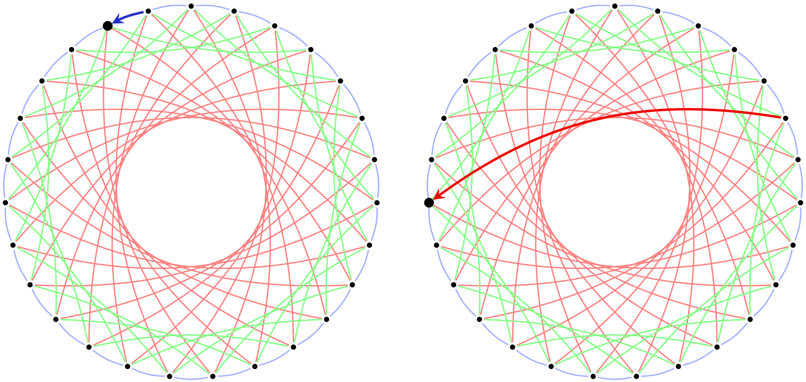
[+, +, −, −]

↑

Bob

[−, +, −, −]

↑

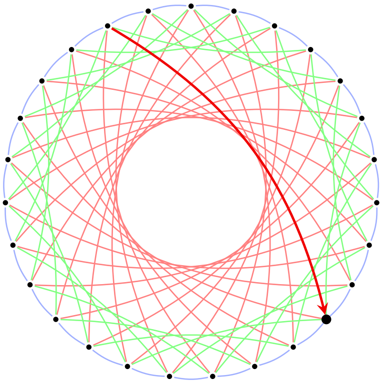


CSIDH [ˈsiːˌsaɪd] key exchange

Alice

[+, +, −, −]

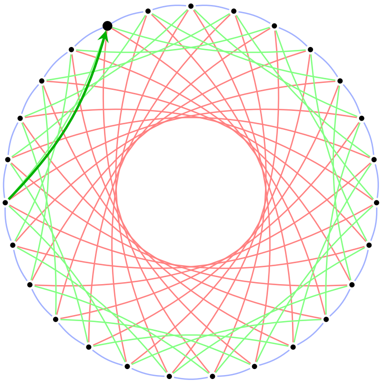
↑



Bob

[−, +, −, −]

↑

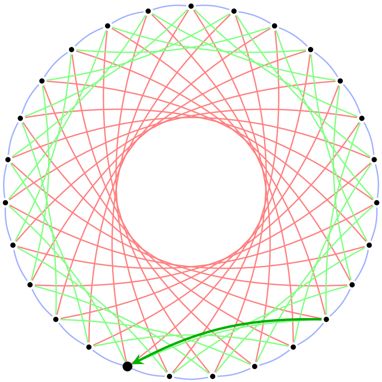


CSIDH [ˈsiːˌsaɪd] key exchange

Alice

[+, +, −, −]

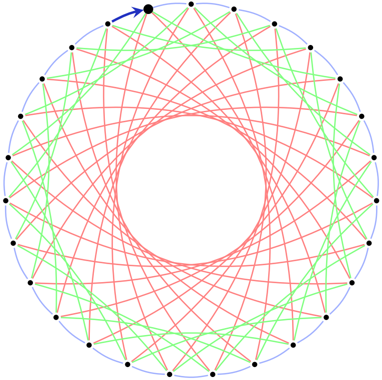
↑



Bob

[−, +, −, −]

↑



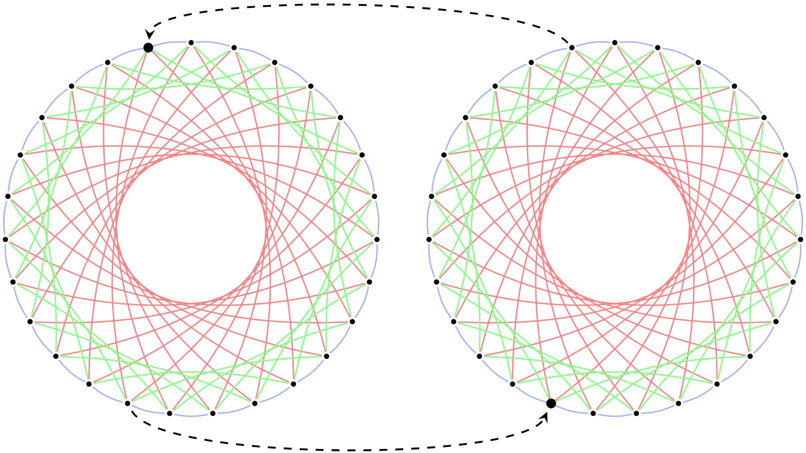
CSIDH [ˈsiːˌsaɪd] key exchange

Alice

[+, +, −, −]

Bob

[−, +, −, −]



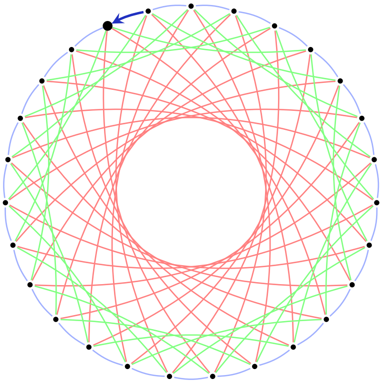
CSIDH [ˈsiːˌsaɪd] key exchange

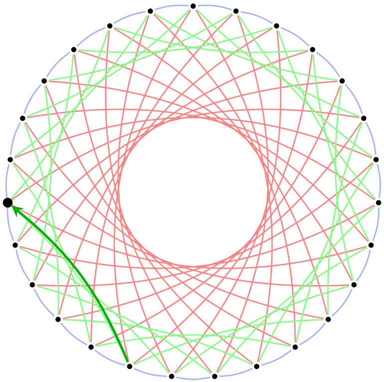
Alice

[+, +, −, −]
↑

Bob

[−, +, −, −]
↑





CSIDH ['siː,saɪd] key exchange

Alice

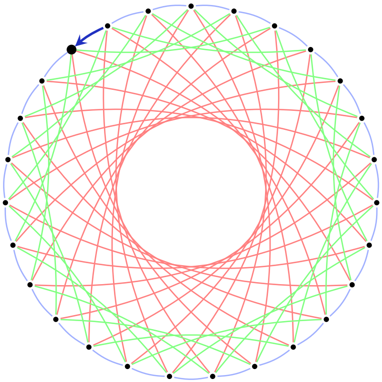
[+, +, −, −]

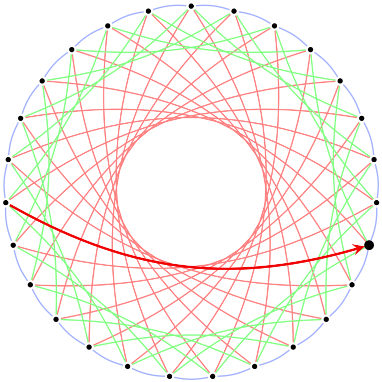
↑

Bob

[−, +, −, −]

↑



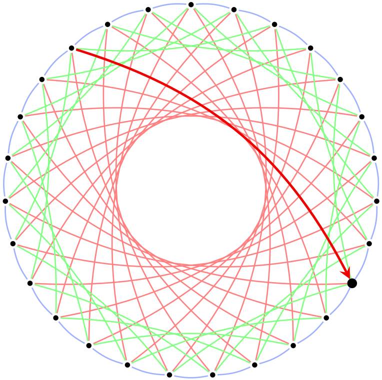


CSIDH [ˈsiːˌsaɪd] key exchange

Alice

[+, +, −, −]

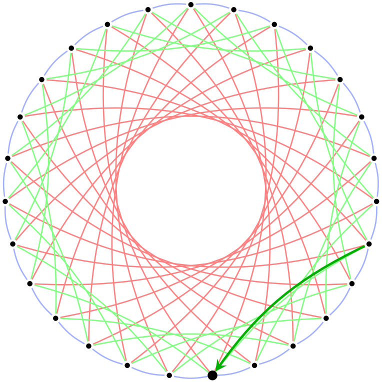
↑



Bob

[−, +, −, −]

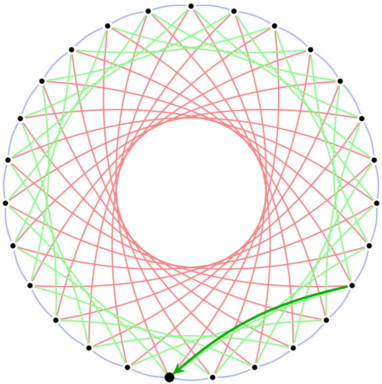
↑



CSIDH [ˈsiːˌsaɪd] key exchange

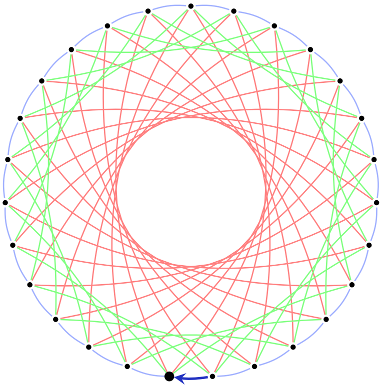
Alice

[+, +, −, −]
↑



Bob

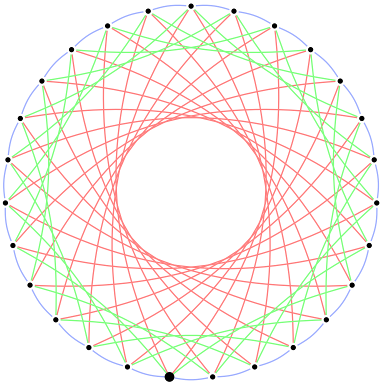
[−, +, −, −]
↑



CSIDH [ˈsiːˌsaɪd] key exchange

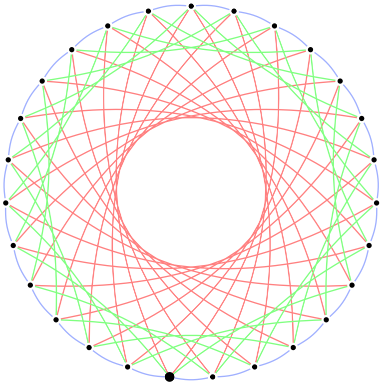
Alice

[+, +, −, −]

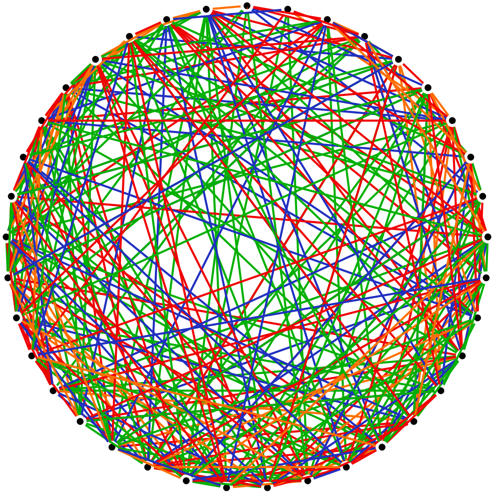


Bob

[−, +, −, −]



A much more random-looking isogeny graph



The Deuring correspondence

Isogeny graphs are not random graphs.
Lots of useful structure looming in the background.

The Deuring correspondence

Isogeny graphs are not random graphs.
Lots of useful structure looming in the background.

Deuring *correspondence*:

Almost exact equivalence between the worlds of maximal orders in certain quaternion algebras and of supersingular elliptic curves.

The Deuring correspondence

Isogeny graphs are not random graphs.
Lots of useful structure looming in the background.

Deuring *correspondence*:

> Almost exact equivalence between the worlds of maximal orders in certain quaternion algebras and of supersingular elliptic curves.

The correspondence is polynomial-time in the $\Longrightarrow$ direction,
but exponential-time in the $\Longleftarrow$ direction. $\rightsquigarrow$ ***Cryptography!***

SQIsign

...is a signature scheme based on this one-wayness.

SQIsign

...is a signature scheme based on this one-wayness.

SQIsign

...is a signature scheme based on this one-wayness.

https://sqisign.org

SQIsign: Numbers

core properties

- \+ Very compact keys and signatures.
- \+ Confident tuning of security parameters.
- \+ No longer slow!
- \- A complex signing procedure.
- ☃ The coolest team!

-- sizes --

parameter set	public keys	signatures
NIST - **I**	**65** bytes	**148** bytes
NIST - **III**	**97** bytes	**224** bytes
NIST - **V**	**129** bytes	**292** bytes

-- performance --

Cycle counts for an optimized implementation using platform-specific assembly running on an Intel *Raptor Lake* CPU:

parameter set	keygen	signing	verifying
NIST - **I**	**43.3** megacycles	**101.6** megacycles	**5.1** megacycles
NIST - **III**	**134.0** megacycles	**309.2** megacycles	**18.6** megacycles
NIST - **V**	**212.0** megacycles	**507.5** megacycles	**35.7** megacycles

Source: https://sqisign.org

Questions?

(Also feel free to email me: `lorenz@yx7.cc`)