Post-Quantum Security

(TU Wien, Austria, December 13th 2023)

o.Prof. Hon.-Prof. Dr. Dr. h.c. Frank Leymann

Institut für Architektur von Anwendungssystemen (IAAS)
Universität Stuttgart

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

Summary

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

Summary

RSA

Key Generation

1. Choose two "large" prime numbers $p, q \in \mathbb{P}$
2. Compute $n = pq$
3. Compute $\varphi(n) = (p-1)(q-1)$ (*Euler-function* φ)
4. Choose "small" $g \in \mathbb{N}$ such that $\varphi(n)$ and g coprime (i.e. $\gcd(\varphi(n), g) = 1$)
5. Determine solution d of equation $dg \equiv 1 \bmod \varphi(n)$ (Euklid $\rightarrow O\left((\log n)^3\right)$)
6. (d,n) is *public* key
7. g is *private* key
8. Destroy $p, q, \varphi(n)$

Cracking the secrete key g means to (i) determine $\varphi(n)$ (note: d is known) and then (ii) solving $dg \equiv 1 \bmod \varphi(n)$ (this is simple). Determining $\varphi(n)$ is simple if $n = pq$ can be determined. This requires factorization. But factorization can be reduced to compute discrete logarithms

En-/Decryption

A. Let μ be message to be encrypted
   - μ is mapped to a binary representation
   - μ must be split into blocks of size < n i.e. $\mu = \mu_1 \parallel \mu_2 \parallel \ldots \parallel \mu_k$ with $\mu_j < n$

B. $\bar{\mu} = \mu^d \bmod n$ is the *encrypted* message

C. $\mu = \bar{\mu}^g \bmod n$ is the *decrypted* original message

Module Exponential Function

Chose an arbitrary $0 < a < n \in \mathbb{N}$

Define $\exp_a : \mathbb{N}_0 \to \mathbb{N}_0$ with $\exp_a(x) = a^x \bmod n$

$\exp_a$ is called *module exponential function* with basis a

$\exp_a$ has *period* p $:\Leftrightarrow \exp_a(p) = \exp_a(0)$ (with minimal p)

$\Leftrightarrow \; a^p \bmod n = a^0 \bmod n = 1 \bmod n$

$\overset{def}{\Leftrightarrow} \; a^p \equiv 1 \bmod n \quad \Leftrightarrow \quad n \,|\, (a^p - 1) \quad \Leftrightarrow \quad \exists k : kn = (a^p - 1)$

$$\boxed{\exp_a \text{ has period p} \Leftrightarrow a^p \equiv 1 \bmod n}$$

Cracking Keys

Assume we can determine the period of $\exp_a$

Then we know $a^p \equiv 1 \mod n \;\Leftrightarrow\; \exists k : a^p - 1 = kn$

Assume $p \in \mathbb{N}$ is even (otherwise: chose another a)

Then: $kn = a^p - 1 = \left(a^{p/2} - 1\right)\left(a^{p/2} + 1\right)$

Cracking the secrete key g means to (i) determine $\varphi(n)$ (note: d is known) and then (ii) solving $dg \equiv 1 \mod \varphi(n)$ (this is simple). Determining $\varphi(n)$ is simple if $n = rs$ can be determined.

$\Leftrightarrow$ $\left(a^{p/2} - 1\right)$ and n have a common divisor, or $\left(a^{p/2} + 1\right)$ and n have a common divisor

$\Leftrightarrow$ $\gcd\left(\left(a^{p/2} - 1\right), n\right)$ or $\gcd\left(\left(a^{p/2} + 1\right), n\right)$ is a divisor of n

I.e. if we can determine $p \in \mathbb{N}$ with $a^p \equiv 1 \mod n$ we can determine a divisor of n and, thus, we can determine the prime factors of n !

Factorization means to determine period of $\exp_a$

Discrete Logarithm

It is $\exp_a(x) = a^x \bmod n$

Smallest number y with $a^y \equiv z \bmod n$ is called *discrete logarithm* of z with basis a : $\log_a z = y$

This defines a map $\log_a : \mathbb{N}_0 \to \mathbb{N}_0$ with $z \mapsto y$ (as usual $\exp_a(\log_a z) = z$ and $\log(\exp_a y) = y$)

<u>Example</u>: $2^4 = 16 \equiv 5 \bmod 11 \Rightarrow \log_2 5 = 4$

We know: $\exp_a$ has period p $\Leftrightarrow a^p \equiv 1 \bmod n$

Computing the period p of $\exp_a$ means to compute $p = \log_a 1$

Thus, factorization can be reduced to compute discrete logarithms

Elliptic Curves

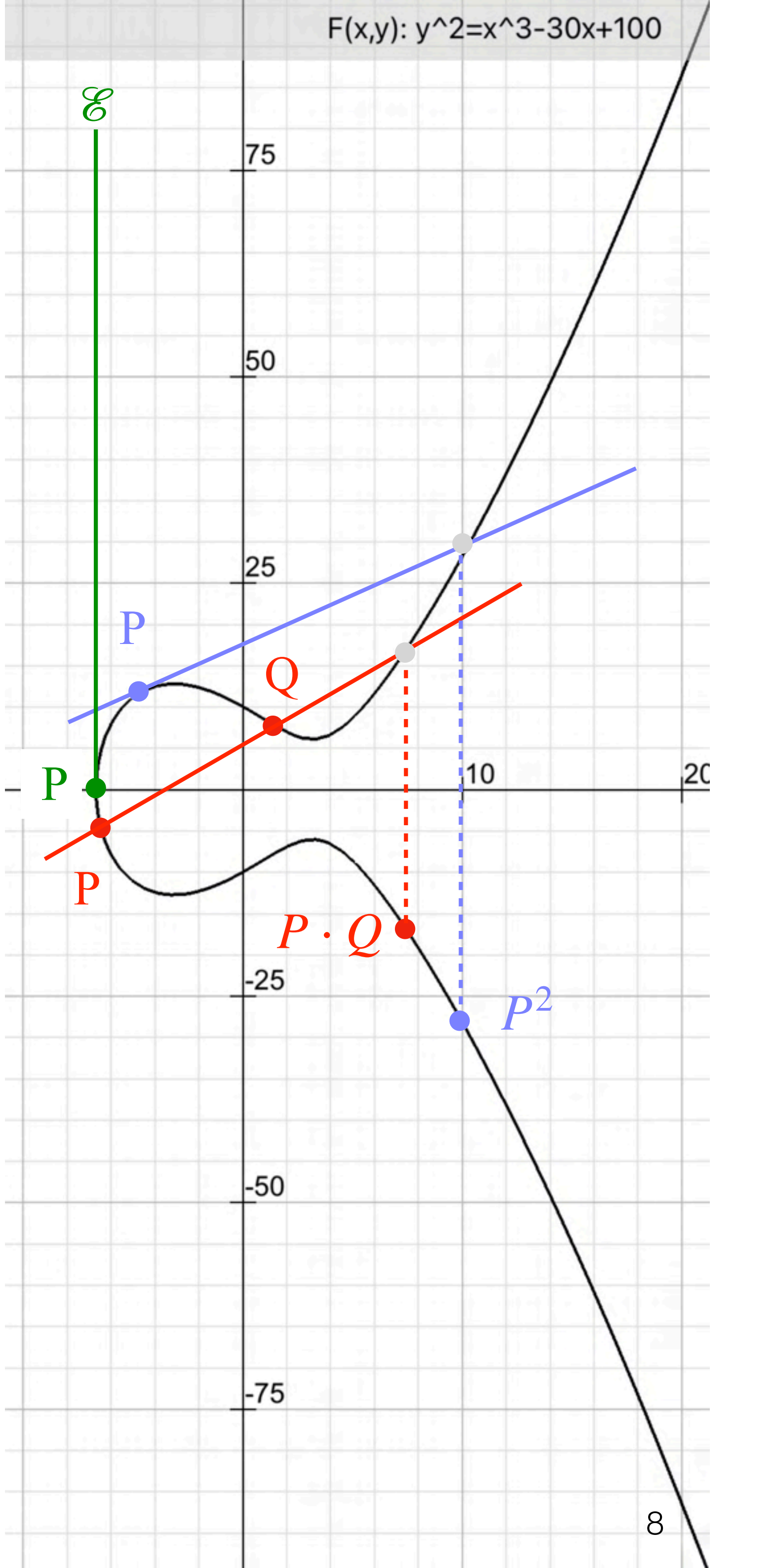
An *elliptic curve* is defined by $y^2 = x^3 + ax + b$
with $4a^3 + 27b^2 \neq 0$

- The curve C is symmetric to the x-axis: $P \in C \Rightarrow -P \in C$
- An infinitely far point $\mathscr{E}$ is added to the curve C

Definition of multiplication $\cdot : C \times C \to C$

- $P, Q \in C$ and let $\overline{PQ} \cap C = R \Rightarrow -R \to P \cdot Q$
- $P \in C$, let g_p be the tangent at C in P, $g_p \cap C = R$
  $\Rightarrow -R \to P^2 := P \cdot P$
- $P \in C$, let g_p be the tangent at C in P, $g_p \cap C = \varnothing$
  $\Rightarrow \mathscr{E} \to P^2$
  - $\mathscr{E}$ becomes the neutral element, in symbol: $P \cdot \mathscr{E} = P$

$(C, \cdot\,)$ is an abelian group



Elliptic Curve Cryptography

(very informal…the whole truth has to introduce finite fields, elliptic curves over finite fields, order and cofactor of subgroups etc*)

- The curve's parameter a and b are published, i.e. $y^2 = x^3 + ax + b$
- A point $G \in C$ (*generator*) is chosen and published
- Each participant has a *private key* $s \in \mathbb{N}$ and $G^s =: P \in C$ as corresponding *public key*

$\Rightarrow$ Cracking the private key s means to compute $s = \log P$

Cracking Elliptic Curve Cryptography can be reduced to compute discrete logarithms

- Shor's algorithm can brake ECC (because it computes discrete logarithms)
  - Braking ECC requires less qubits than factorization, and orders of magnitudes fewer gates $\Rightarrow$ cracking ECC with quantum computers is easier than cracking RSA

(https://arxiv.org/abs/1706.06752)

(* nice overview in https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/)

Elliptic Curve Encryption

- Let $G \in C$ be the generator
- Then, $G^s =: P \in C$ is the *public key*, with $s \in \mathbb{N}$ the *private key*

Encryption

- Message m is mapped to a point $M \in C$
  - $m \in \mathbb{N}$ as a bit string $\Rightarrow$ then compute y with $y^2 = m^3 + am + b \;\Rightarrow\; M = (m, y)$
- Choose a random $k \in \mathbb{N}$
- Compute $E_1 = G^k$ and $E_2 = M \cdot P^k$
- Encrypted message is $\chi(m) = (E_1, E_2)$

Decryption

- Compute $M = E_2 \div (E_1)^s = E_2 \cdot (E_1)^{-s}$
  - It is $(E_1)^s = (G^k)^s = (G^s)^k = P^k$
  - $\Rightarrow E_2 \cdot (E_1)^{-s} = M \cdot P^k \cdot (E_1)^{-s} = M \cdot P^k \cdot P^{-k} = M$
- Then $m = \pi_1(M)$

In practice, ECC is not used for encryption, but for key exchange and signatures

ECC Key Exchange
(Elliptic Curve Diffie-Hellman)

- Let $G \in C$ be the generator
- Alice has $G^{s_A} =: P_A \in C$ as public key, with $s_A \in \mathbb{N}$ as private key
- Bob has $G^{s_B} =: P_B \in C$ as public key, with $s_B \in \mathbb{N}$ as private key

- Alice and Bob exchange P_A, P_B

- Alice computes $K_A = P_B^{s_A}$
- Bob computes $K_B = P_A^{s_B}$
- In fact, $K_A = P_B^{s_A} = \left(G^{s_B}\right)^{s_A} = \left(G^{s_A}\right)^{s_B} = P_A^{s_B} = K_B$

$\Rightarrow$ Alice and Bob share the same secrete key $K \left(= K_A = K_B \right)$

(<u>Note</u>: Also, signatures can be computed based on elliptic curves too)

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

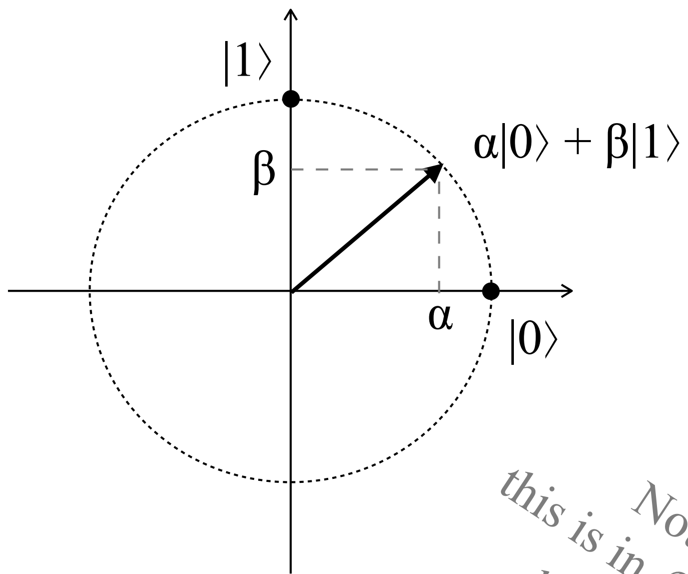
Summary

Qbit & Superposition

Quantum bit (***Qbit***) is in the two classical states $|0\rangle$ or $|1\rangle$ <u>at the same time</u> (!):
Superposition

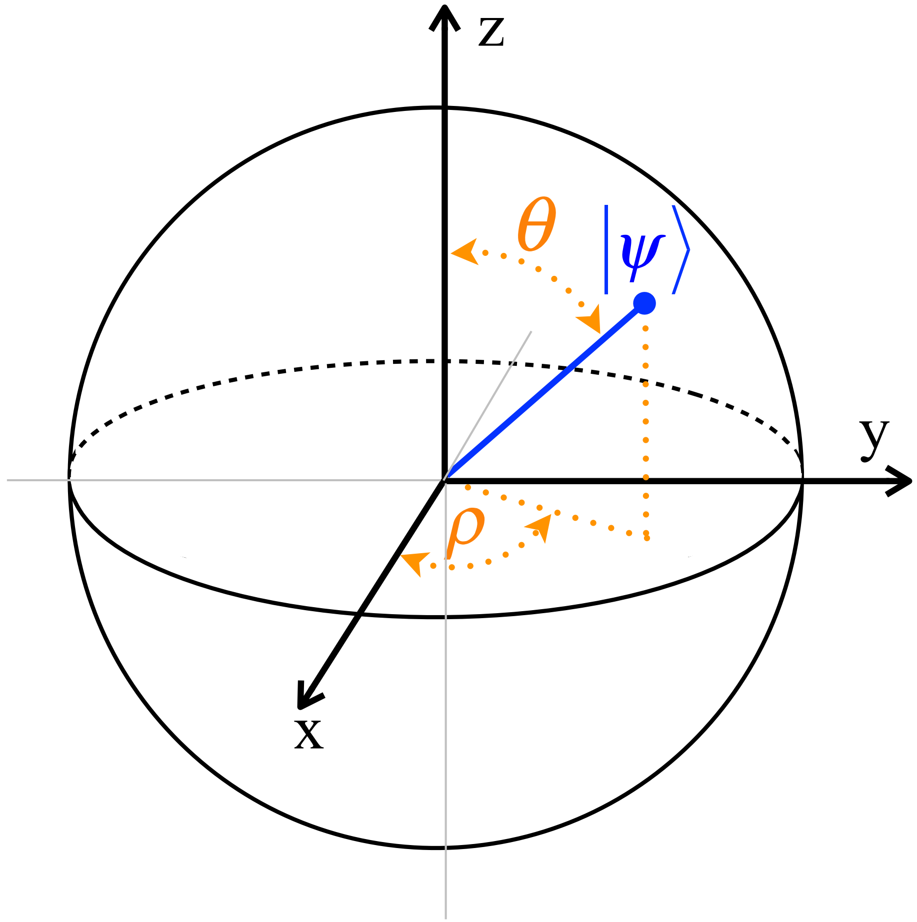
State of a qbit is $\alpha|0\rangle + \beta|1\rangle$

$\alpha, \beta \in \mathbb{C}$ and $\mathbf{|\alpha|^2 + |\beta|^2 = 1}$.

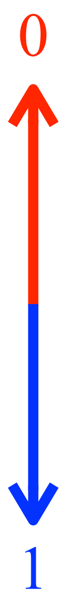


Note: this is in $\mathbb{C}^2 \simeq \mathbb{R}^4$, <u>not</u> in $\mathbb{R}^2$! I.e. this is S^3, <u>not</u> S^1!

Bloch Sphere

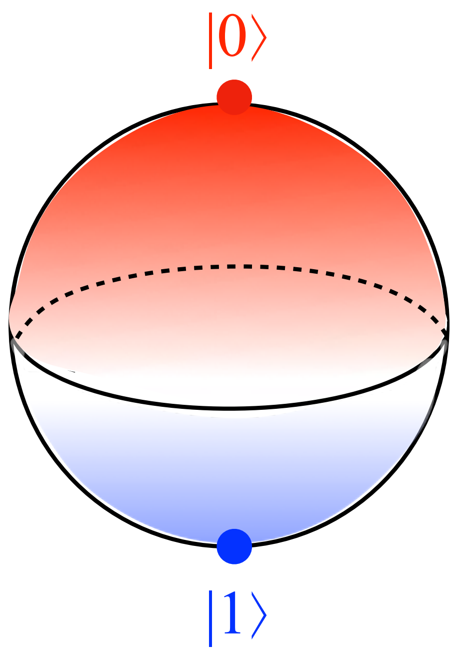


Intuition of a Qbit



Bit

A bit is either "0" or "1"
→ Two possible values



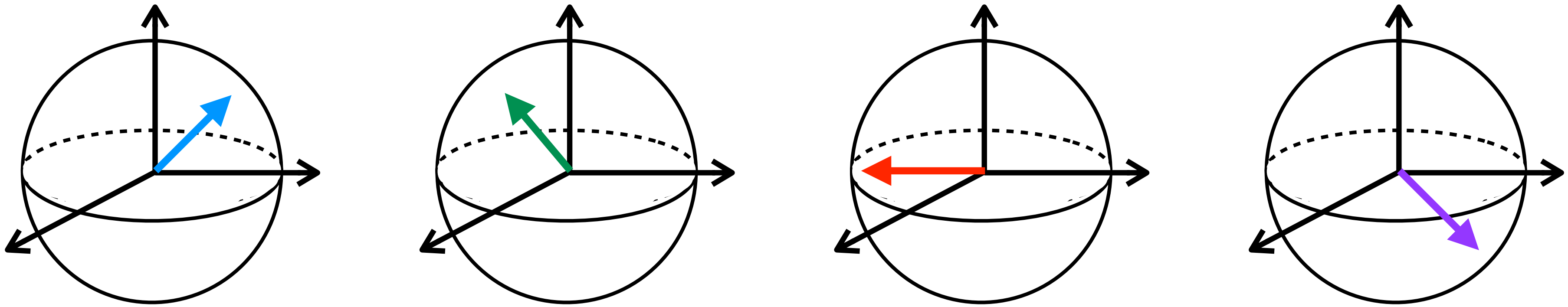
Qbit

A qbit is an arbitrary point on the Bloch Sphere
→ Uncountably infinit possible values

Quantum Register

Classical register is a series of n bits

Quantum ***register*** is a series of n qbits



Quantum register with n qbits is the superposition of the corresponding 2^n states

$$|00...00\rangle, |00...01\rangle, |00...10\rangle, ..., |11...11\rangle$$

Quantum Parallelism

Classical register with n bit $\rightarrow$ 1 value at a time

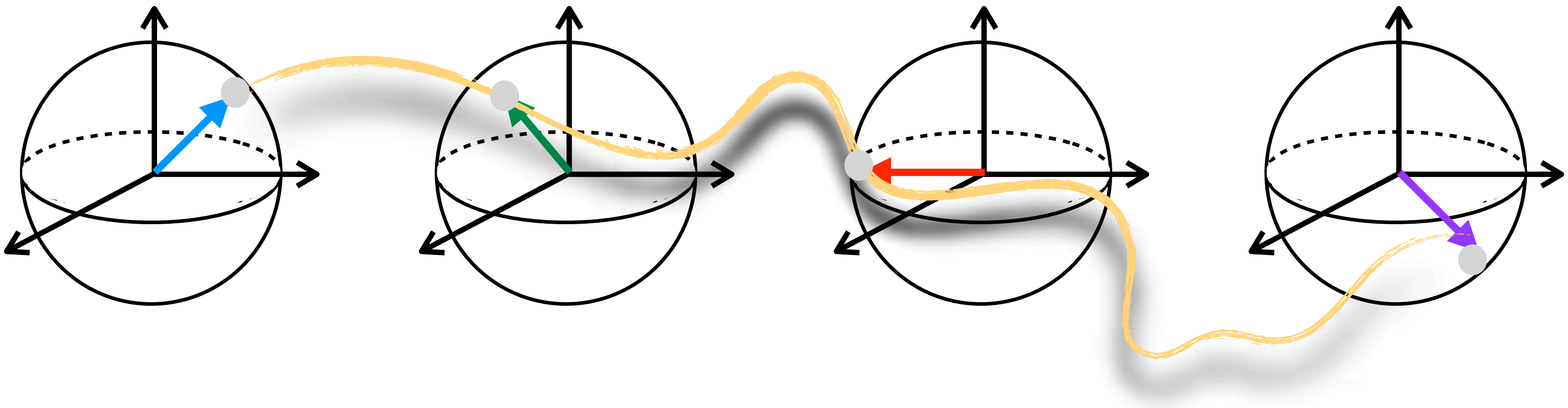
Quantum register with n qbit $\rightarrow$ 2^n values at the **<u>same</u>** time

50 Qbits $\mapsto$ $2^{50} = (2^{10})^5 > (10^3)^5 = 10^{15}$ ($\triangleq$Peta…)

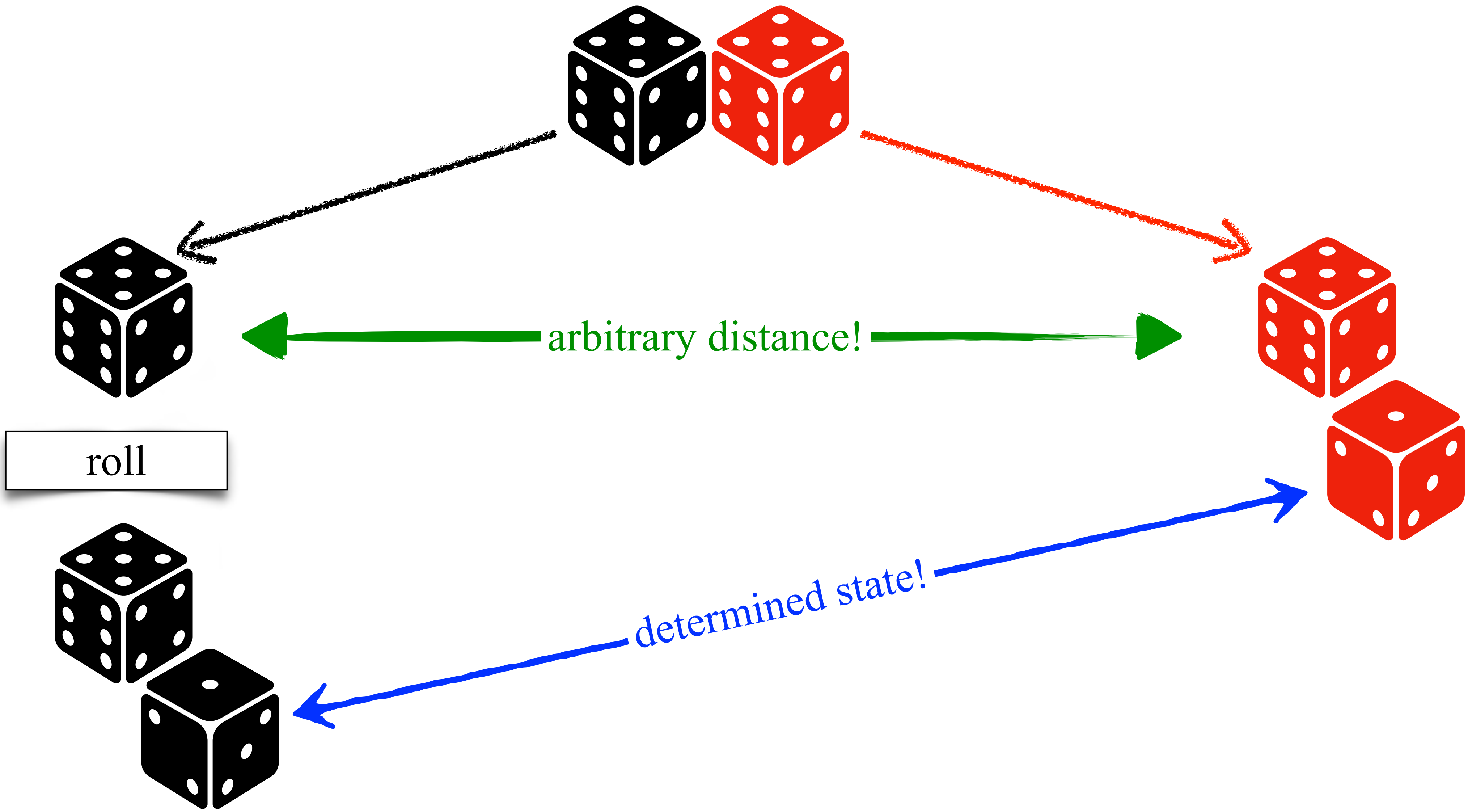
#Atoms in universe $\leq 10^{90} = (10^3)^{30} \leq (2^{10})^{30} = 2^{300} \mapsto$ 300 Qbits

Quantum computer manipulates 2^n values at the same time
(*Quantum Parallelism*)

Entanglement



Intuition: Entanglement



Entanglement: Importance

Entanglement is unique for quantum computing!

Each computation <u>not</u> involving entangled qubits,
can be realized classically and **in principle** with the same efficiency than a quantum computation

(**but**: n qubits $\Rightarrow 2^n$ classical storage | quantum parallelism | …)

Every quantum algorithm showing exponential speedup compared to a classical algorithm, must exploit entanglement.

(R. Jozsa, N. Linden: On the role of entanglement in quantum computational speed-up. (2003) arXiv:quant-ph/0201143v2)

Shor Algorithm: Impression

Step 1: Initialize Qbit-Register

$R = |a\rangle|b\rangle \leftarrow |0...0\rangle|0...0\rangle$

Step 2: Apply Hadamard Transformation to $|a\rangle$

$$R \leftarrow H^{\otimes n} \otimes I\left(|0\rangle|0\rangle\right) = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|0\rangle$$

Superposition

Step 3: Apply Oracle Function

$$R \leftarrow U_f\left(R\right) = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|f(x)\rangle$$

Entanglement & Parallelism

Step 4: Apply Quanten Fourier Transformation to $|a\rangle$

$|a\rangle \leftarrow \mathrm{QFT_N}\left(|a\rangle\right)$

Step 5: y ← Measure $|a\rangle$

Determine Period

Quantum Part

Step 6: Expand y as "Continued Fraction" $[y_0;y_1,\ldots,y_n]$

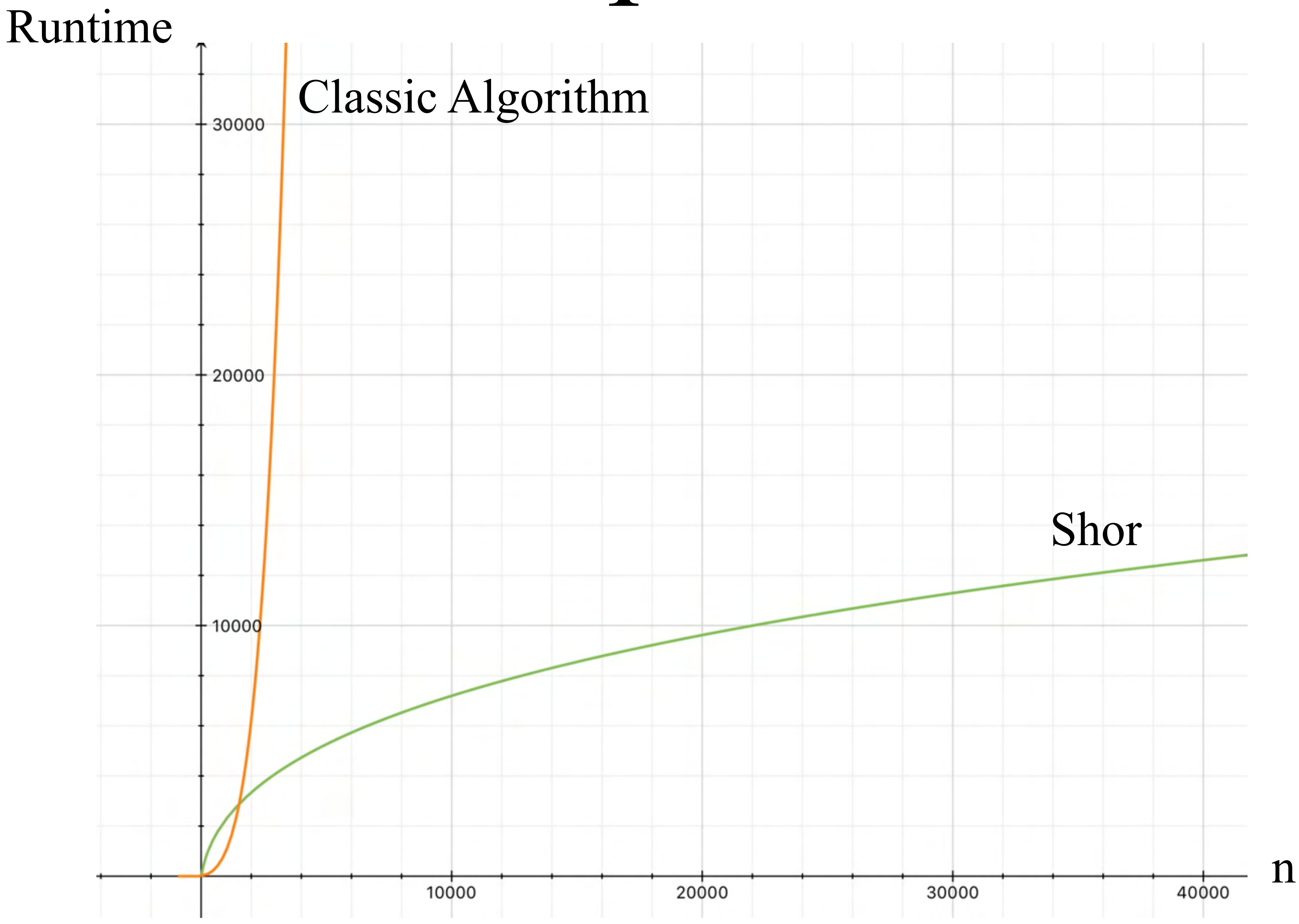
Step 7: p ← Determine p from convergents $[y_0;y_1,\ldots,y_k]$, $k \le n$

Step 8: Output of p

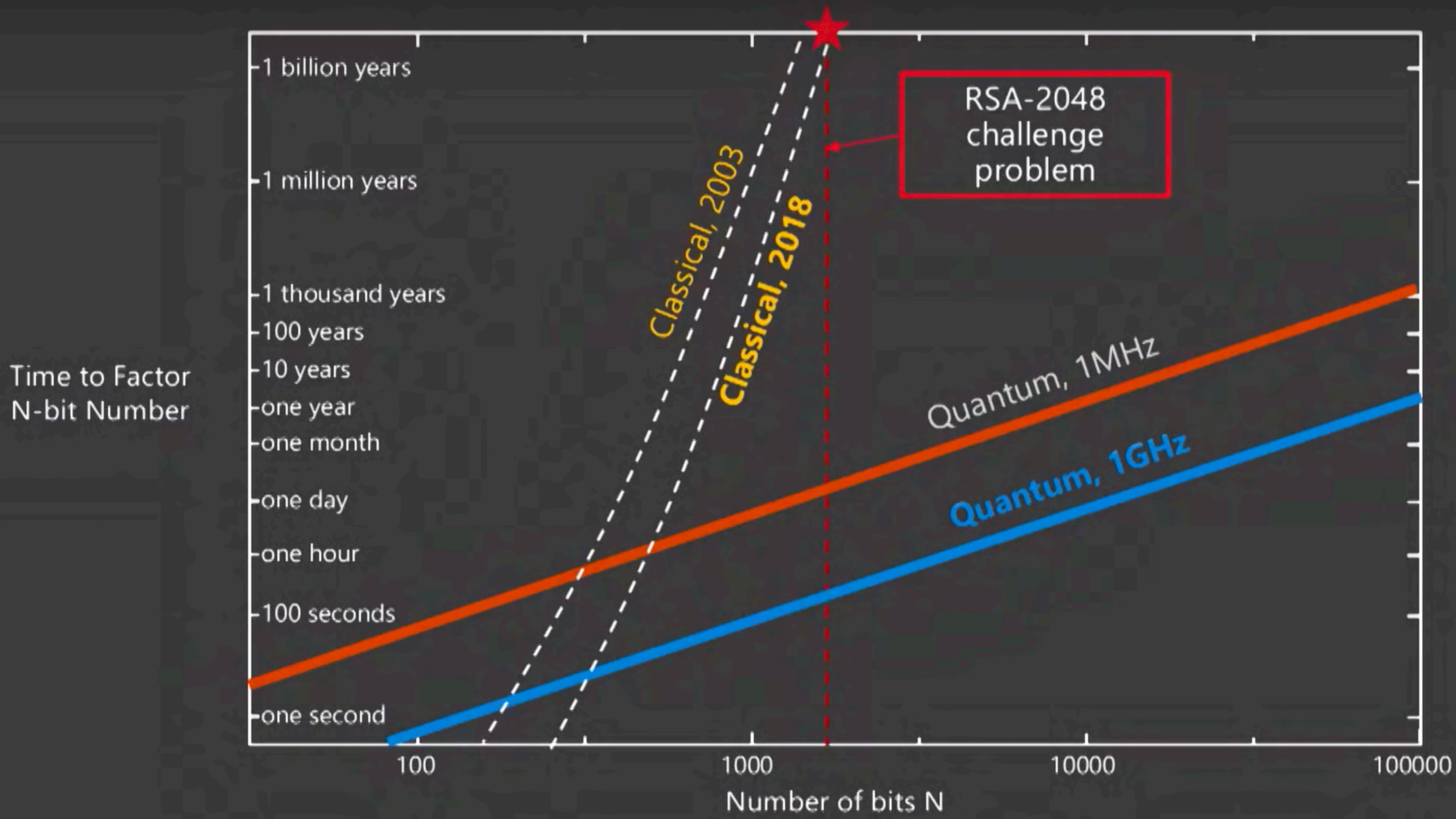
Classical Part

J. Barzen, F. Leymann: Continued Fractions and Probability Estimations in Shor's Algorithm: A Detailed and Self-Contained Treatise. In: AppliedMath. Vol. 2(3), MDPI, 2022

Comparison


Runtime
Classic Algorithm
Shor
n
30000
20000
10000
10000
20000
30000
40000

Assumption: ***error-corrected*** QPU!



(In 2019 , frequency was already at ≈5GHz
https://www.ibm.com/blogs/research/2019/12/qiskit-openpulse/)

Required Qbits & Fidelity

To factorize a number Z that has n Bits in binary representation,

in order to run the Shor Algorithm you need:

- $5n+1$ Qbits
- $4n^3$ Operations (Gates)

CRQC

Cryptographically Relevant Quantum Computer

RSA-2048:

- $n = 2048$
- $\approx$ 10.000 qbits
- $\approx 34 \cdot 10^9$ operations

Today:

- $\approx$ 400 qbits
- $\approx 10^3$ operations

Seems to be impossible for quite some years!

Recent Advancements (1/2)

...we show that 12 logical qubits can be preserved for ten million syndrome cycles using 288 physical qubits in total, assuming the physical error rate of 0.1% (*)

$\Rightarrow \sim$ 24 physical qubits ($\varnothing$) required for realizing 1 logical qubit

...achieving the same level of error suppression on 12 logical qubits with the surface code would require more than 4000 physical qubits (*)

$\Rightarrow \sim$ 10…100 $\times$ improvement from former expectations!

$\Rightarrow$ 10.000 logical qubits $\widehat{=}$ 10.000 $\cdot$ 24 = 240.000 physical qubits

Expected in a few years!

(*) Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, Theodore J. Yoder: High-threshold and low-overhead fault-tolerant quantum memory. (2023) https://arxiv.org/abs/2308.07915

Recent Advancements (2/2)

…we have successfully factorized … 261980999226229 (48-bit), with …10 qubits in a superconducting quantum processor…

We estimate that a quantum circuit with 372 physical qubits and a depth of thousands is necessary to challenge RSA-2048 using our algorithm.

https://arxiv.org/pdf/2212.12372.pdf (2022)

⇒ NISQ machine!

TABLE I. Resource estimation for RSA numbers. The main quantum resources mentioned are the number of qubits, the quantum circuit depth of QAOA with a single iteration in three typical topologies, including all connected system (Kn), 2D-lattice system (2DSL) and 1D-chain system (LNN). The results are obtained without considering the native compilation of the ZZ-basic module (or ZZ-SWAP basic module) in a specific physical system.

RSA number	Qubits	Kn-depth	2DSL-depth	LNN-depth
RSA-128	37	113	121	150
RSA-256	64	194	204	258
RSA-512	114	344	357	458
RSA-1024	205	617	633	822
RSA-2048	372	1118	1139	1490

…it's getting close!

…that's still a problem!

IBM's Osprey QPU already has 433 Qubits (2023)

…But What About Symmetric Encryption?

- Variational Quantum Attack Algorithm (VQAA) (*)
  https://link.springer.com/content/pdf/10.1007/s11432-022-3511-5.pdf
  - …suitable for todays NISQ machines (!)
  - …can crack symmetric keys
    (standard (AES)-like symmetric cryptography)

- This invalidates two assumptions (which have been consensus):
  - Symmetric encryption is quantum safe
  - Error corrected QPUs are needed to break encryption

(*) Wang, Z., Wei, S., Long, GL. et al.: Variational quantum attacks threaten advanced encryption standard based symmetric cryptography. Sci. China Inf. Sci. 65, 200503 (2022)

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

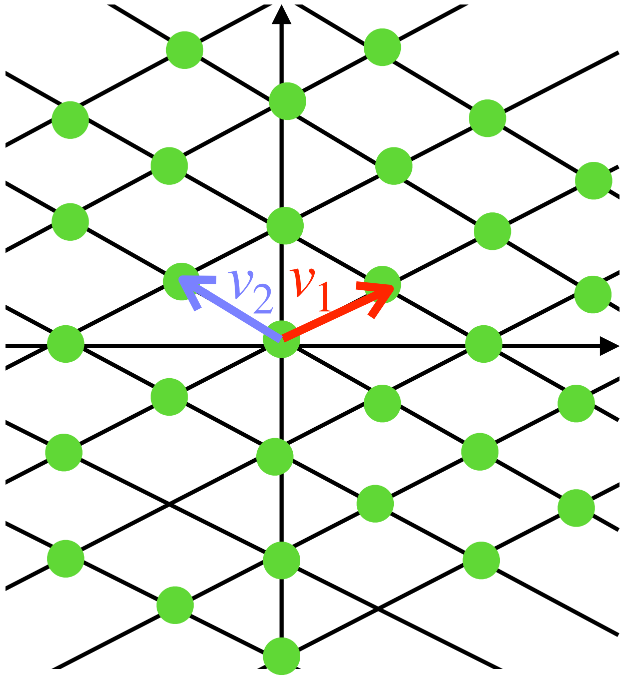
Summary

Basics about Lattices

Lattices

Let $v_1, \ldots, v_k \in \mathbb{R}^n$ be linear independent

$\Lambda_{\mathbb{Z}} < v_1, \ldots, v_k > := \left\{ \sum_{i=1}^{k} g_i v_i \mid g_i \in \mathbb{Z} \right\}$ is called the *lattice* of rank k with basis $\{v_1, \ldots, v_k\}$



If k = n the lattice is called to have *full rank*

Λ is an abelian subgroup of $\mathbb{R}^n$

An Equivalent Definition

A *lattice* Λ is a discrete additive subgroup of $\mathbb{R}^n$, i.e.

- Λ is closed under addition
- there is an $\varepsilon > 0$ such that for all $x \neq y \in \Lambda$ it is $\| x - y \| \geq \varepsilon$

From a high level, the equivalence of both definitions is proven as follows:

- Obviously, $\mathbb{Z}^n$ is a lattice (discrete and additive)
- If $M \in \mathrm{GL}(n, \mathbb{R})$, then $\{Mx \mid x \in \mathbb{Z}^n\}$ is a lattice (additivity is clear, discreteness is hard)
- $\{Mx \mid x \in \mathbb{Z}^n\} = \left\{ \sum x_i m_i \mid x_i \in \mathbb{Z} \right\} = \Lambda_{\mathbb{Z}} < m_1, \ldots, m_n >$ is a lattice in the former sense

  …where $M = (m_1 \ldots m_n)$ is the matrix with column $m_1, \ldots, m_n$

Based on the above definition:

$$\boxed{M \in \mathrm{GL}(n, \mathbb{R}) \;\Rightarrow\; \{x \in \mathbb{Z}^n \mid Mx = 0\} \text{ is a lattice}}$$

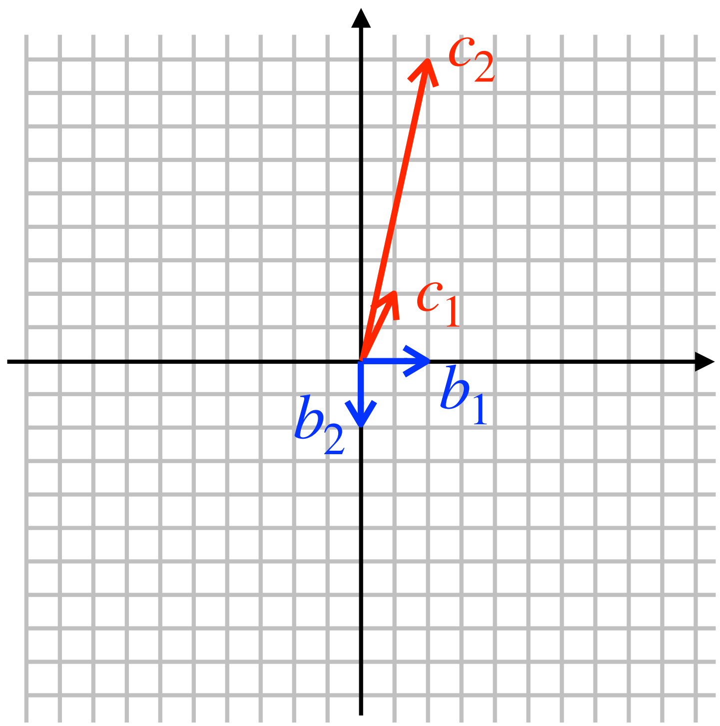
(this is an important observation for post-quantum cryptography! See LWE later!)

Bases of a Lattice

A lattice Λ has many different bases:

- Let $\mathscr{U} \in \mathbb{Z}^{n \times n}$ with $\det \mathscr{U} = \pm 1$ (*unimodular* matrix)
- Let $\mathscr{B} = \{b_1, \ldots, b_n\}$ be a basis of Λ and B the matrix with columns $b_1, \ldots, b_n$
- Then, the columns $c_1, \ldots, c_n$ of the matrix $C = B \cdot \mathscr{U}$ are a basis $\mathscr{C} = \{c_1, \ldots, c_n\}$ of Λ

Computations with the basis b_1, b_2 are much simpler than computations with the basis c_1, c_2 especially in high dimensions ($n > 500$) that are used in practice



$\Rightarrow$ Question is: what are good bases for lattice-based computations and how to find them?

Lattice Problems

Nearly Orthogonal Basis

Let $\mathscr{B} = \{v_1, \ldots, v_n\}$ be a basis of the lattice Λ of full rank

Let $B = (v_1 \ldots v_n)$ be the matrix with columns v_i

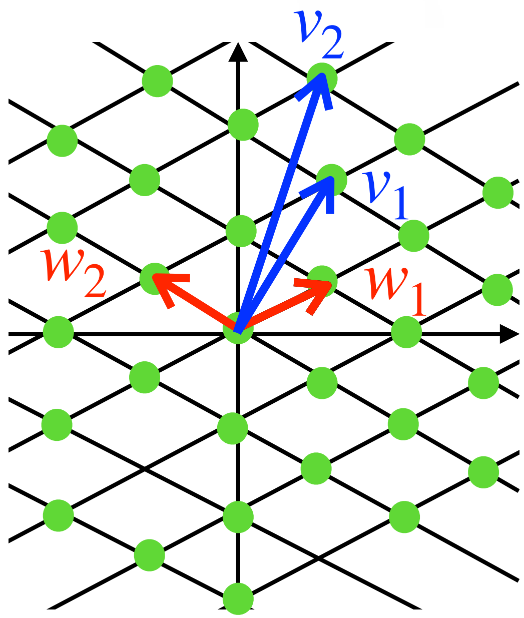
Then, $|\det B|$ is the volume of the parallelotop spanned by $\{v_1, \ldots, v_n\}$

Hadamard's Inequality

$$\det B \leq \prod_{i=1}^{n} \| v_i \|_2$$

$$\delta(\mathscr{B}) := \frac{\prod_{i=1}^{n} \| v_i \|}{|\det B|}$$ is called (*orthogonality*) *defect* of $\mathscr{B}$

($\mathscr{B}$ orthogonal $\Rightarrow \delta(\mathscr{B}) = 1$, i.e. it is $\delta(\mathscr{B}) \geq 1$)

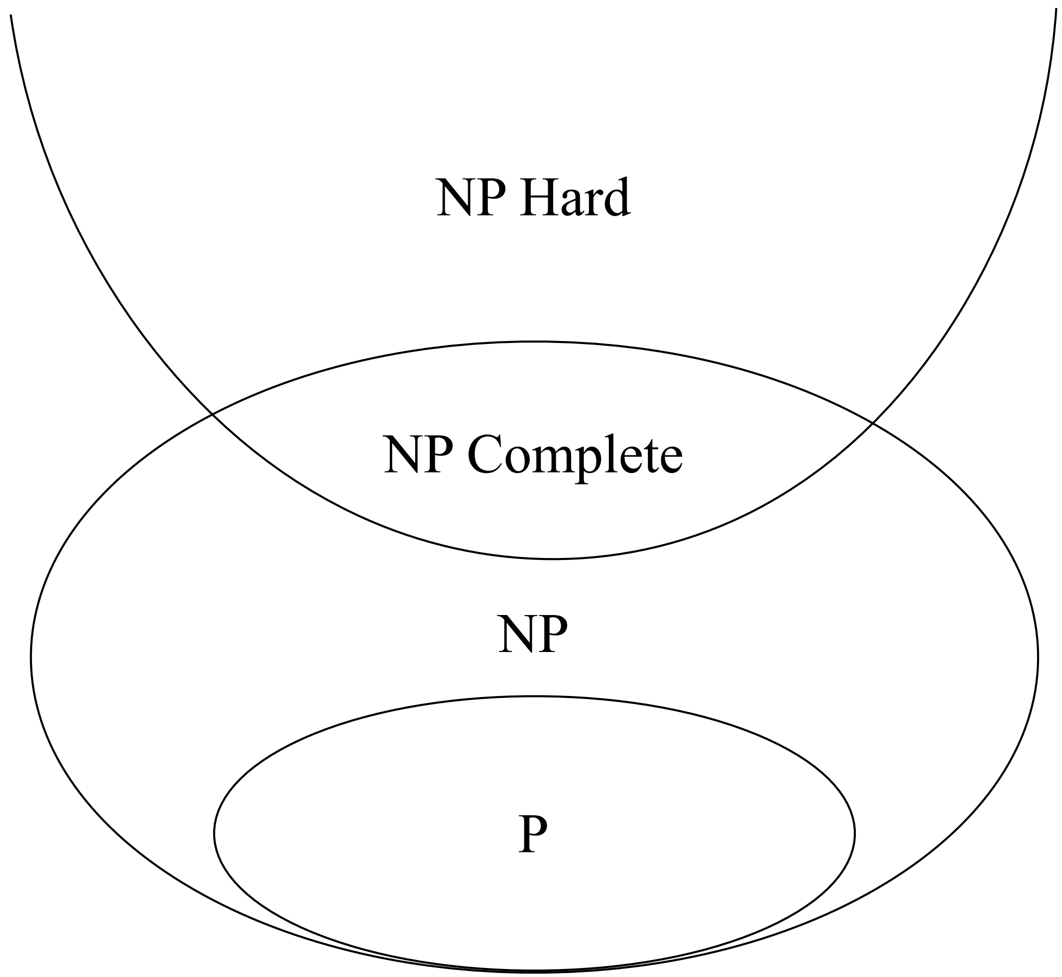


$\{v_1, v_2\}$ original basis,
$\{w_1, w_2\}$ basis with minimal defect

Lattice-Reduction-Problem:
determine a basis $\mathscr{B}$ for Λ with minimal $\delta(\mathscr{B})$

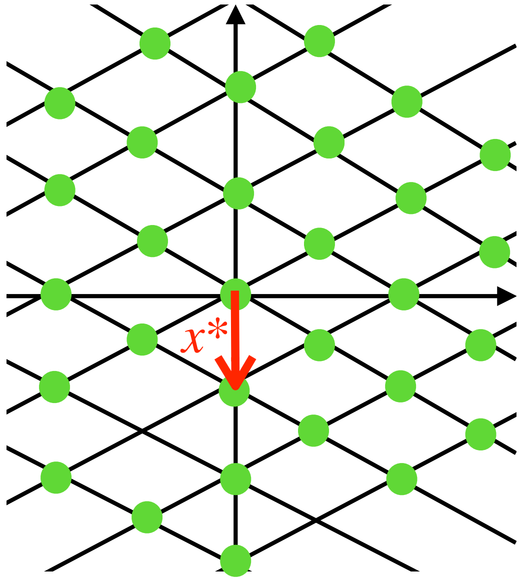
The Lattice-Reduction-Problem is NP-hard

Reminder: Complexity



Shortest-Vector-Problem

Let $\Lambda_{\mathbb{Z}} < v_1, \ldots, v_n >$ be a lattice of full rank,
$\|\cdot\|$ a norm on $\mathbb{R}^n$, $\lambda_1(\Lambda) := \min\limits_{x \in \Lambda \setminus \{0\}} \| x \|$



Shortest-Vector-Problem (SVP):
Determine $x^* \in \Lambda : \| x^* \| = \lambda_1(\Lambda)$

SVP is NP-hard for $\|\cdot\|_2$

SVP is NP-complete for $\|\cdot\|_\infty$

Algorithm see: https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch18.pdf

Shortest-Independent-Vectors

Let $\Lambda_{\mathbb{Z}} < v_1, \ldots, v_n >$ be a lattice of full rank,

let $\mathfrak{B}$ be the set of all bases of Λ and $M(\Lambda) := \min\limits_{\{b_1,\ldots,b_n\}\in\mathfrak{B}} \max\limits_{1\leq i\leq n} \| b_i \|$

- $M(\Lambda)$ is the minimal length of the longest vector in any basis

Shortest-Independent-Vectors-Problem (SIVP):
Find a basis $\{s_1, \ldots, s_n\}$ of Λ with $\max\limits_{1\leq i\leq n} \| s_i \| = M(\Lambda)$

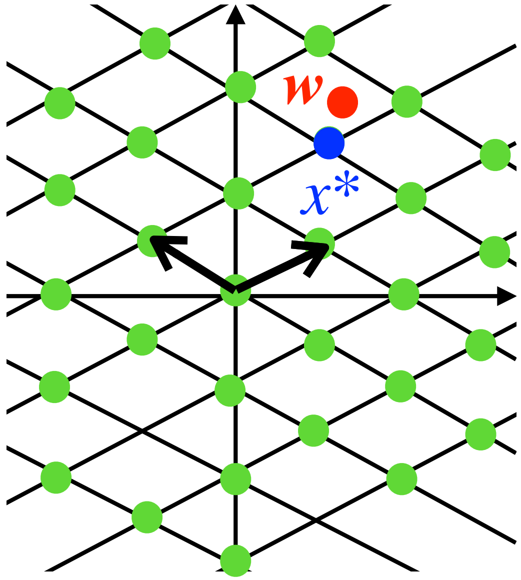
(Determine a basis with a minimal length of the longest vector in the basis)

SIVP is NP-complete

Closest-Vector-Problem

Let $\Lambda_{\mathbb{Z}} < v_1, \ldots, v_n >$ be a lattice of full rank,

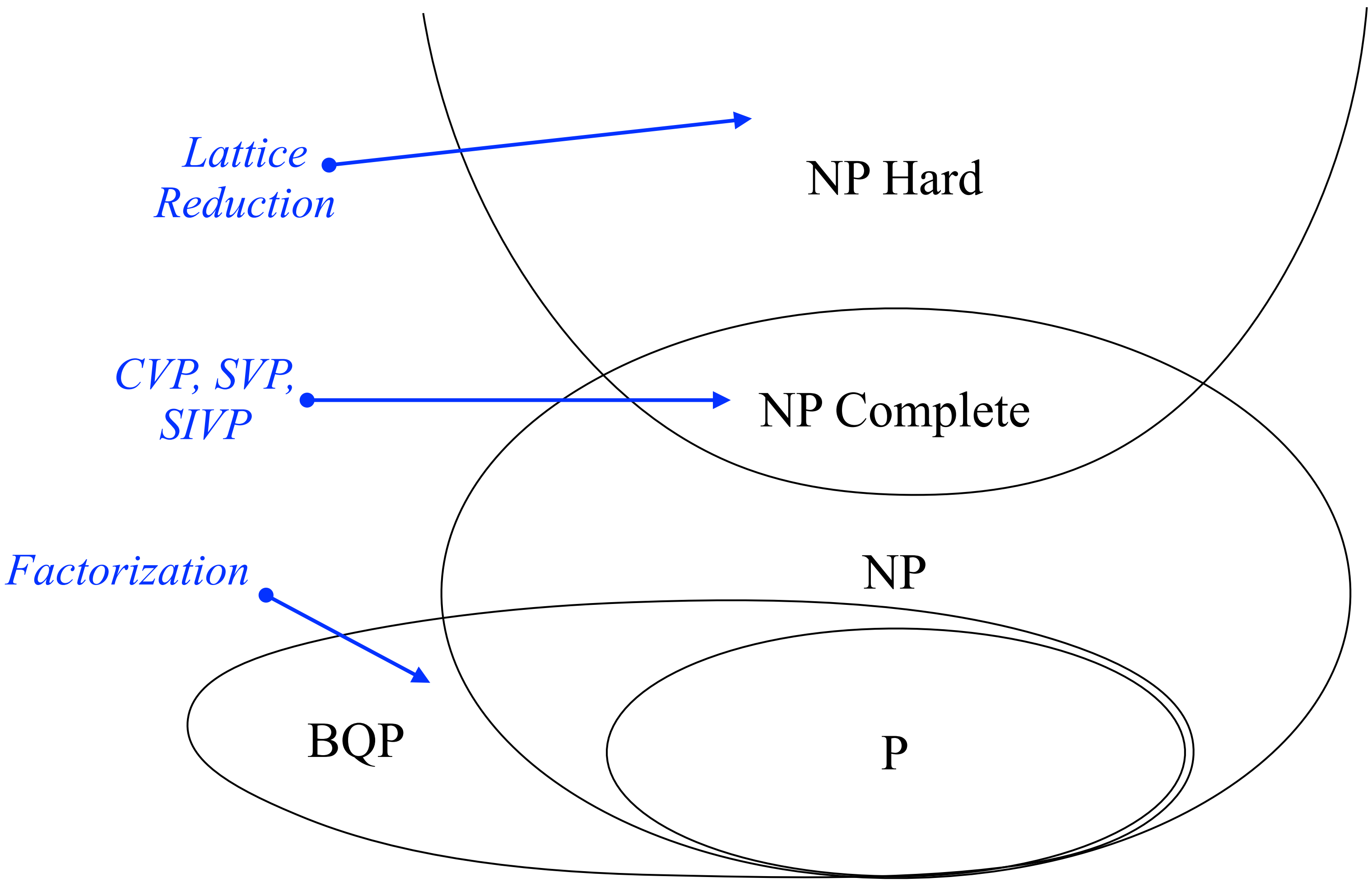
$\| \cdot \|$ a norm on $\mathbb{R}^n$, $w \in \mathbb{R}^n$ and $\operatorname{dist}(\Lambda, w) := \min_{x \in \Lambda} \| x - w \|$



Closest-Vector-Problem (CVP):
Determine $x^* \in \Lambda : \| x^* - w \| = \operatorname{dist}(\Lambda, w)$

CVP is NP-complete

Complexity: Quantum Computing & Lattices



<u>Short Integer Solution</u>:
(see later)
is average-case hard

<u>Prime Factorization</u>:
50% probability that a randomly chosen number can easily be factorized:

$$N = 2 \cdot \frac{N}{2}$$

$\Rightarrow$ worst-case hardness

Worst-Case
vs.
Average-Case

Why Lattice-Based Cryptography

Worst-case hardness

- There is a (!) problem instance ("worst case") on which each efficient algorithm fails

Average-case hardness

- A randomly chosen problem instance is as hard to solve as the worst problem instance

Average-case hardness $\Rightarrow$ Worst-case hardness (the reverse is not true!)

Cryptography requires average-case hardness, i.e. worst-case hardness does not suffice

- E.g. encryption must be hard on a random instance, not only on the worst-case instance

For factoring (or discrete logarithm or elliptic curves as used today) it is not known…

- …whether it is at the end in P or whether it is NP-hard
- …whether it is average-case hard (i.e. really suitable for cryptography)

Using lattice problems it is possible to construct average-case-hard problems which are just as difficult as certain well-known worst-case-hard problem

Miklo´s Ajtai: Generating Hard Instances of the Short Basis Problem. (2001) https://people.csail.mit.edu/vinodv/CS294/ajtai99.pdf

Short-Integer-Solution

Let $n, m, q \in \mathbb{N}$ and $\beta \in \mathbb{R}_{>0}$

Let $A \in \mathbb{Z}_q^{n \times m}$ with $a_{ij} \in \mathbb{Z}_q$ uniformly random

β-*Short-Integer-Solution-Problem* (β-SISP):
Determine $z \in \mathbb{Z}_q^m \setminus \{0\} : Az = 0 \quad \wedge \quad 0 < \ \| z \| \ \leq \beta$

- For $\beta \geq \sqrt{m} q^{n/m}$ the β-SIS-Problem has a solution.

There is a polynomial reduction from the
Shortest Independent Vector Problem to the Short Integer Solution Problem

A randomly selected SIVP is as hard as the worst SISP!

In lattice-based cryptography, one-way-functions are constructed based on SISP

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

Summary

Polynomial Rings

For a commutative ring R, die *polynomial ring* $R[X]$ is defined as follows:

- A *polynomial* in X over R is an expression $p = a_0 + a_1X + \ldots + a_nX^n$ with $a_i \in R$
- For $p = \sum_{i=0}^{n} a_iX^i$ and $q = \sum_{i=0}^{m} b_iX^i$ it is
  - $p + q = \sum_{i=0}^{\max(m,n)} r_iX^i$ and $r_i = a_i + b_i$
  - $p \cdot q = \sum_{i=0}^{m+n} s_iX^i$ with $s_i = a_0b_i + a_1b_{i-1} + \cdots + a_ib_0$

An equivalent definition is:

- A *polynomial* over R is a sequence p : $\mathbb{N} \rightarrow R$ with $\mathrm{card}\{i \mid p_i \neq 0\} < \infty$
- $p + q = (a_i + b_i)_{i\in\mathbb{N}}$ and $p \cdot q = (\sum_{i+j=k} a_ib_j)_{k\in\mathbb{N}}$
  - With $X := (0,1,0,\ldots)$ it is $a_0 + a_1X + \ldots + a_nX^n \cong (a_0, a_1, \ldots, a_n, 0, 0, \ldots)$

Signatures

Dilithium

- Substituting the field $\mathbb{K}$ of a $\mathbb{K}$-vector space V by a ring R results in a *module* over R
- Let $\mathbb{Z}_q$ be the ring of integers modulo q
- Then, $R_q := \mathbb{Z}_q[X] \big/ (X^n + 1)$ is a ring (consisting of polynomials)
- Dilithium is using $n = 256$ and the prime $q = 2^{23} - 2^{13} + 1 = 8380417$
- $B_h \subset R_q$: set of polynomials having exactly h coefficients that are $+1$ or -1
- It is card $B_h = 2^h \begin{pmatrix} n \\ h \end{pmatrix}$; Dilithium is using $h = 60$, resulting in card $B_h \geq 2^{256}$
  - B_{60} is the range of the hash function $\Psi : \{0,1\}^* \to B_{60}$ constructed in Dilithium
- The matrices $R_q^{m \times k}$ are then an R_q module
- Given $A \in R_q^{m \times k}$ and $t \in R_q^m$ (both uniformly random), following *module-SISP* is solved:
  - Given message M, find $y \in R_q^{k+1+m}$ with $0 < \| y \| \leq \beta$ such that $\Psi\left(M \,\|\, \underbrace{\left(A \,|\, t \,|\, E_m \right)} \cdot y \right) = c \in B_{60}$
    where $y = \begin{pmatrix} r_1 \\ c \\ r_2 \end{pmatrix}$ (with $r_1 \in R_q^k$, $c \in B_{60}$, $r_2 \in R_q^m$)

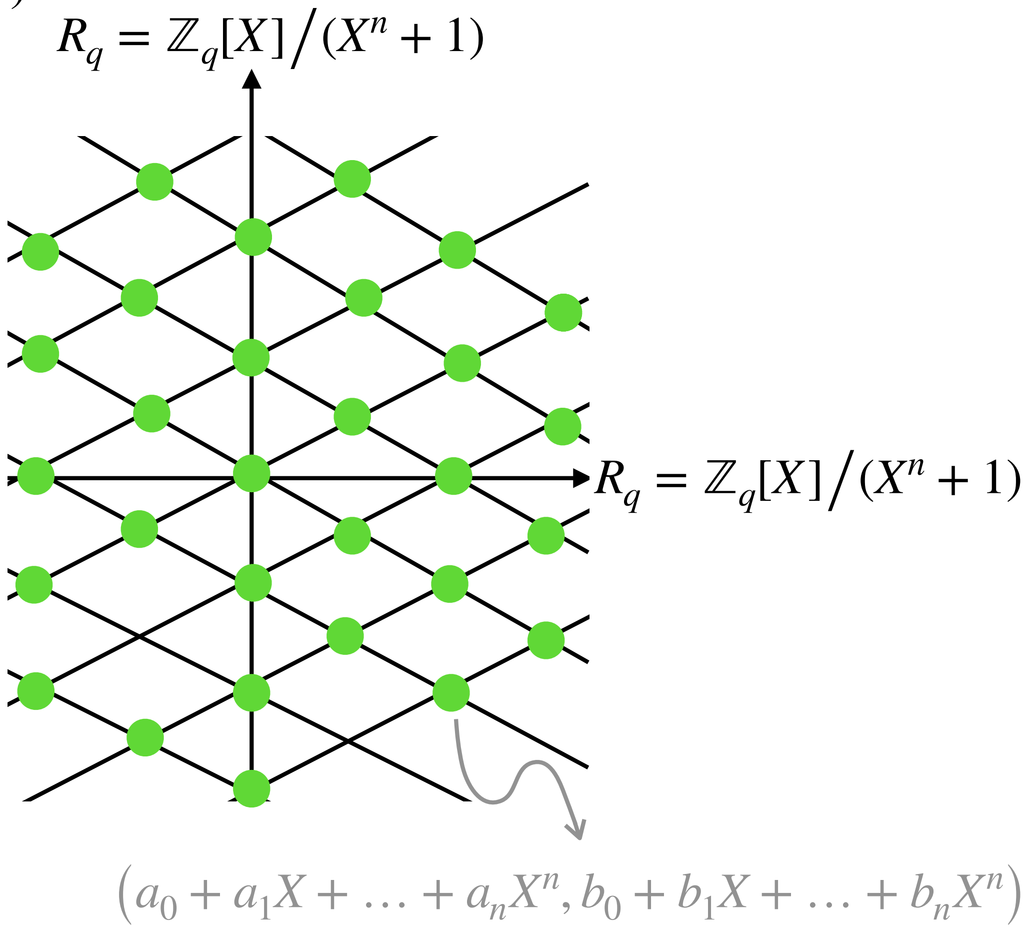
message

concatenation

(matrix with columns of A,
a single column t,
and the $m \times m$-identity matrix)

Situation

- We are dealing with "vector spaces" (modules) the points of which are tuples of polynomials
  - …of degree $n = 256$
- All computations are done $\bmod q$, with $q = 8380417$
  - Kyber is using $q = 3329$
- Multiplication of polynomials based on number theoretic transform (NTT)



Dilithium (approximately)

Signature on M

Secrete Key

$$\Psi\left(M \,\|\, \left(A \mid t \mid E_m\right) \cdot \begin{pmatrix} r_1 \\ c \\ r_2 \end{pmatrix}\right) = c$$

Public Key

(For details see:
https://uwspace.uwaterloo.ca/bitstream/handle/10012/14832/BakosLang_Elena.pdf
and
https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

Dilithium and LWE

$$\Psi\left(M \,\|\, \left(A\,|\,t\,|\,E_m\right) \cdot \begin{pmatrix} r_1 \\ c \\ r_2 \end{pmatrix}\right) = c$$

$$\Downarrow$$

$$\left(A\,|\,t\,|\,E_m\right) \cdot \begin{pmatrix} r_1 \\ c \\ r_2 \end{pmatrix} = Ar_1 + \underbrace{ct}_{\text{"error"}} + r_2$$

Dilithium is an LWE problem

(see next)

Encryption

Learning With Errors

Let $f : V \rightarrow W$ be a linear function

Given samples $\{(x, y)\}$, $x \in V$ and $y \in W$ with $f(x) = y$,

linear algebra makes is easy to determine ("learn") f (Gaussian elimination)

Assume that the input $\{(x, y)\}$ to the "learning algorithm" has errors,

i.e. it is $f(x) \neq y$ for each (x, y) with some small probability

Example: instead of solving $Ax = y$ for each y, a secret random vector e is chosen (e is "small", i.e. an error) and $Ax + e = y$ is to be solved

- A and y are public, x and e are private

All known algorithms that solve such *Learning-with-Error* problems (LWE) are exponential

LWE hardness

- Lattice-based problems are believed to be intractable assuming LWE is hard
- Under this assumption, SVP can be reduced from worst-case hard to average-case hard
- **Most important**: LWE is proven hard assuming worst-case hardness of SIVP

LWE as Lattice Problem

Reminder: For a matrix M the integer solutions $\left\{x \in \mathbb{Z}^n \mid Mx = 0\right\}$ build a lattice

Let $A \in R_q^{n \times m}$ and $b \in R_q^n$

(A,b) is the public key

$R_q = \mathbb{Z}_q[X]/(X^n + 1)$

And let $s \in R_q^m$ and $e \in R_q^n$ be "small" vectors

$\Rightarrow$ The LWE problem $As + e = b$ is equivalent to $\left(A \,|\, E_n \,|\, -b\right) \cdot \begin{pmatrix} s \\ e \\ 1 \end{pmatrix} = 0$

The solutions of the LWE problem $As + e = b$ build a lattice $\Lambda := \left\{x \in \mathbb{Z}^{n+m+1} \mid \left(A \,|\, E_n \,|\, -b\right) \cdot x = 0\right\}$

$\begin{pmatrix} s \\ e \\ 1 \end{pmatrix}$ solves the Shortest Vector Problem for Λ

Especially:
LWE is a lattice problem!

s is the private key

Kyber

Public key and private key are computed as before

Kyber is using $N = 256$ and $q = 3329$

Encryption is based on another LWE problem:

- $r \in R_q^n$, $e_1 \in R_q^m$ and $e_2 \in R_q$ are randomly sampled
- First, $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A^T \\ b^T \end{pmatrix} r + \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$, i.e. $u = A^T r + e_1 \in R_q^m$ and $v = \langle b, r \rangle + e_2 \in R_q$
- Next, the message μ to be encrypted resulting in the modified $v = \langle b, r \rangle + e_2 + \rho(\mu)$
  - ρ is a fancy "rounding" mechanism, $\rho : \{0,1\}^{256} \rightarrow R_q$
    - ρ transforms the message into a polynomial
  - But the principle is important: the message μ is hidden in the value v

$\begin{pmatrix} u \\ v \end{pmatrix}$ *is the encrypted message*

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

NIST & Industry

Summary

Industry Awareness

Why Should You Care Today?

Many data that are secret and have to be kept for many years are encrypted with current methods

⇒ These data will be cracked in the foreseeable future!

Harvest now, decrypt later!

Encryption based on quantum technology requires new (hardware) infrastructure

- E.g. for (symmetric) key exchange

Quantum-safe procedures require changes to the (software) infrastructure

- E.g. new encryption algorithms

⇒ Building this infrastructure takes many years!

Post-Quantum Cryptography (PQC)

(aka: Quantum-Safe, Quantum-Proof or Quantum-Resistant)

...this refers to cryptographic algorithms which, according to todays knowledge, are secure against attacks with a quantum computer

- I.e. no quantum algorithm is known to crack such an algorithm
- Most of the algorithms "common" today are <u>not</u> quantum-safe
  - Factorization, discrete logarithms, elliptical curves

Corresponding procedures are e.g.

- Hash-based cryptography, lattice-based cryptography.

But:
There is no guarantee that these methods cannot be cracked at some point by quantum computers (or even classic computers)!

(see later!)

Quantum Computing Cybersecurity Preparedness Act

US Law

Public Law 117 - 260 - Quantum Computing Cybersecurity Preparedness Act

Summary | Related Documents

Category	Bills and Statutes
Collection	Public and Private Laws
SuDoc Class Number	AE 2.110:117-260
Law Number	Public Law 117-260
Date Approved	December 21, 2022
Full Title	An act to encourage the migration of Federal Government information technology systems to quantum-resistant cryptography, and for other purposes.
Bill Number	H.R. 7535
Report Number	S Rept. 117-251
Statutes at Large Citations	136 Stat. 2389, 2390, 2391 and 2392
United States Code Citations	44 U.S.C. 3502, 3552 and 3553 Chapter 35
Legislative History	LEGISLATIVE HISTORY–H.R. 7535 (S. 4592):

https://www.govinfo.gov/app/details/PLAW-117publ260

THE WHITE HOUSE

Administration Priorities The Record Briefing Room Español MENU

MAY 04, 2022

Home

National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems

…the United States begins the multi-year process of migrating vulnerable computer systems to quantum-resistant cryptography

https://www.whitehouse.gov/briefing-room/statements-releases/2022/05/04/national-security-memorandum-on-promoting-united-states-leadership-in-quantum-computing-while-mitigating-risks-to-vulnerable-cryptographic-systems/

A Quote from the NSA

TECHNOLOGY VENDOR RESPONSIBILITIES

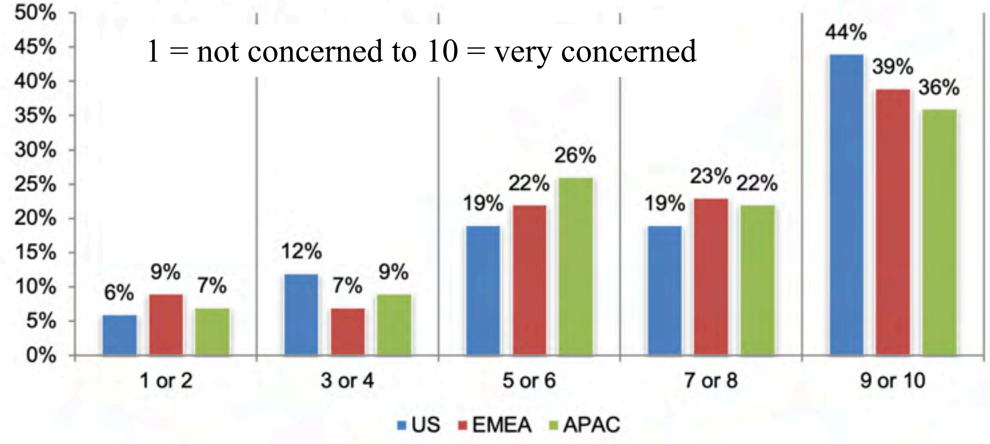
Technology manufacturers and vendors whose products support the use of quantum-vulnerable cryptography should begin planning and testing for integration. CISA, NSA, and NIST encourage vendors to review the NIST-published draft PQC standards, which contain algorithms, with the understanding that final implementation specifics for these algorithms are incomplete. Ensuring that products use post-quantum cryptographic algorithms is emblematic of Secure by Design principles. Vendors should prepare themselves to support PQC as soon as possible after NIST finalizes its standards.

Awareness in Industry

digicert

Ponemon INSTITUTE

Figure 19. How concerned are you that your organization will not be prepared to address the security implications of PQC?



Preparing for a Safe Post Quantum Computing Future: A Global Study

Sponsored by DigiCert

Independently conducted by Ponemon Institute LLC

Publication Date: October 2023

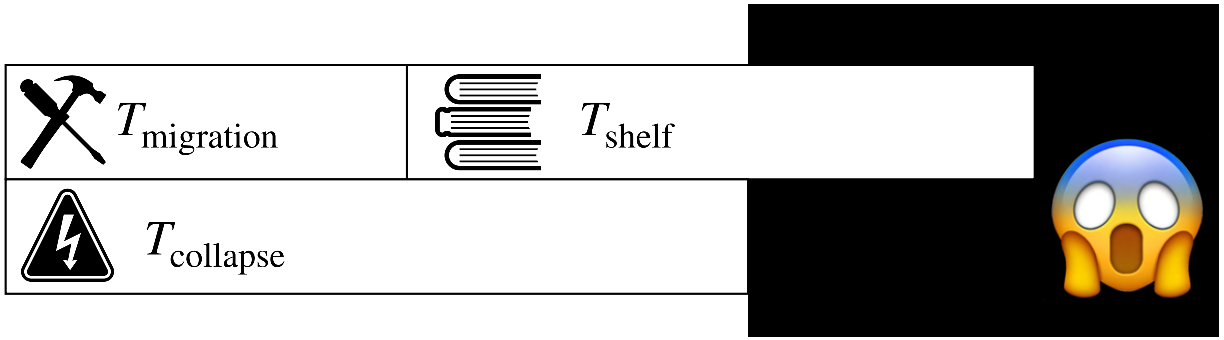
https://www.digicert.com/content/dam/digicert/pdfs/report/ponemon-preparing-safe-post-quantum-future-report-en-v1.pdf

How Much Time Do You Have?

(Mosca's Inequality(*))

- Collapse Time T_{collapse} : number of years until a CRQC is available
- Migration Time $T_{\text{migration}}$: number of years needed to realize a quantum-safe solution
- Shelf-Life Time T_{shelf} : number of years information must be kept secrete

You are in trouble if: $T_{\text{shelf}} + T_{\text{migration}} > T_{\text{collapse}}$



The time you have: $T^{\text{max}}_{\text{migration}} := T_{\text{collapse}} - T_{\text{shelf}}$

- Assume $T_{\text{collapse}} = 10$ y , $T_{\text{shelf}} = 10$ y $\Rightarrow$ $T^{\text{max}}_{\text{migration}} = 0$ y. **You must begin now!**

(*) https://globalriskinstitute.org/publication/2022-quantum-threat-timeline-report/

Secure Communication: Basics

TLS

Protocol for secure data transport based on dynamically established symmetric key

Two phases: (1) TLS Handshake and (2) TLS Record

Handshake performs secure key exchange (and authentication)

- Agree on cypher suite
- Server authenticates itself via certificate; client may do the same
- Client sends secret random number encrypted with server's public key
- …or Diffie-Hellman is used to derive a shared secret

⇒ symmetric session key

Record uses the negotiated symmetric key to transport data securely

- Data is encrypted by means of the symmetric key
- …and protected with a message authentication code (MAC)
  - …symmetric key-based hash of the data

Diffie-Hellman

Algorithm for computing a symmetric key via private keys

- Alice and Bob agree on prime number $p \in \mathbb{P}$ and $g \in \mathbb{N}$ with $g < p$ (p and g may be pubic!)
- Alice and Bob generate their secrete keys $a, b \in \{1, \ldots, p-1\}$
- Alice computes her public key $A = g^a \bmod p$ and sends it to Bob
- Bob computes his public key $B = g^b \bmod p$ and sends it to Alice
- Alice computes $K_1 = B^a \bmod p$
- Bob computes $K_2 = A^b \bmod p$

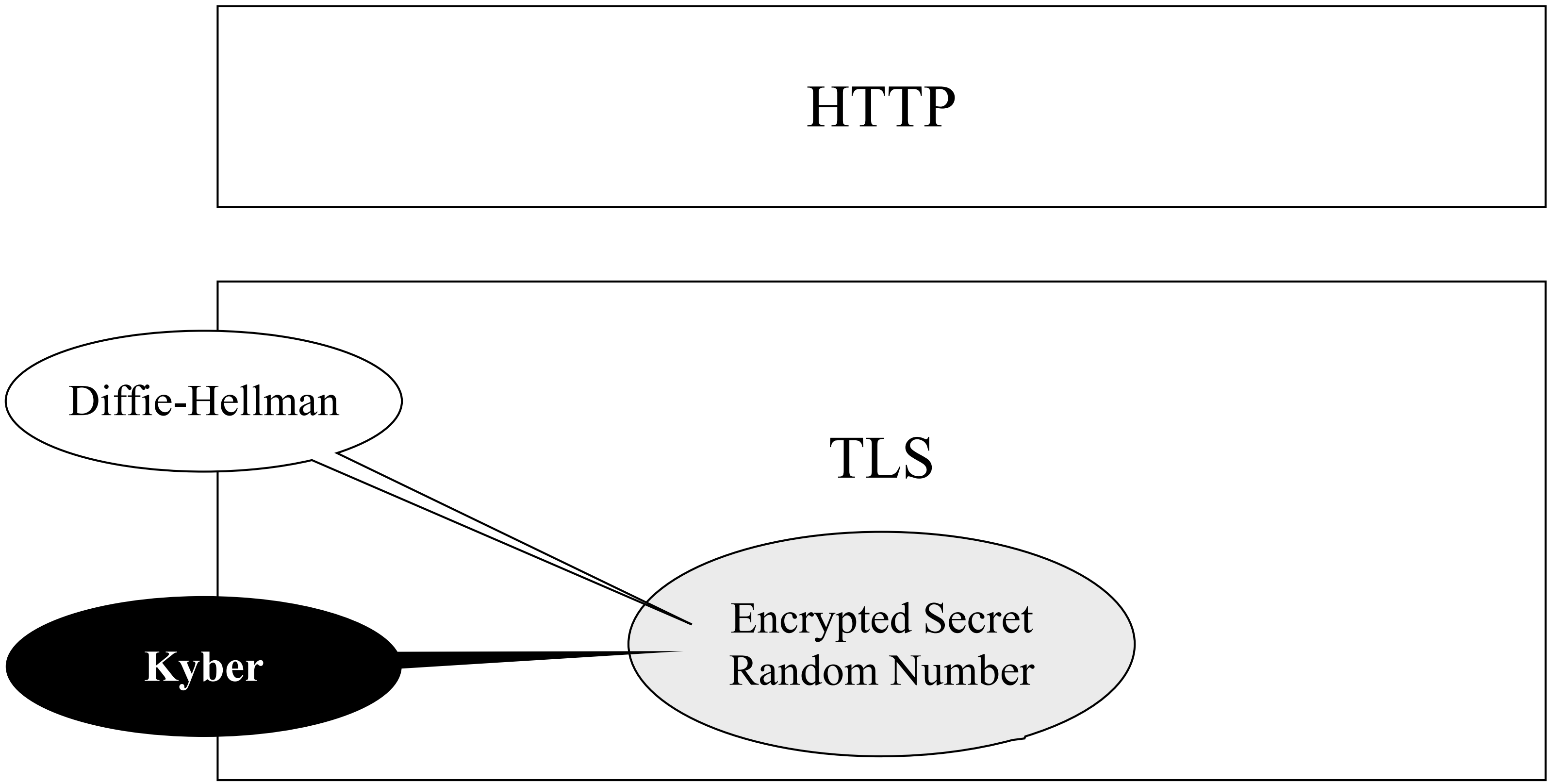
$K_1 = K_2 \quad \Rightarrow \quad$ shared secrete key !

$$K_1 = B^a \bmod p = (g^b \bmod p)^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p = g^{ab} \bmod p$$

$$K_2 = A^b \bmod p = (g^a \bmod p)^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$$

HTTPS

- TLS by default uses build in *encrypted secret random number* mechanism
- Is often substituted by Diffie-Hellmann, e.g.
- Post-quantum resistance may be achieved by using Kyber (e.g.) instead



Ways to Become Quantum Resistant

Let ***C*** be a classical security algorithm, and let ***Q*** be a corresponding post-quantum algorithm

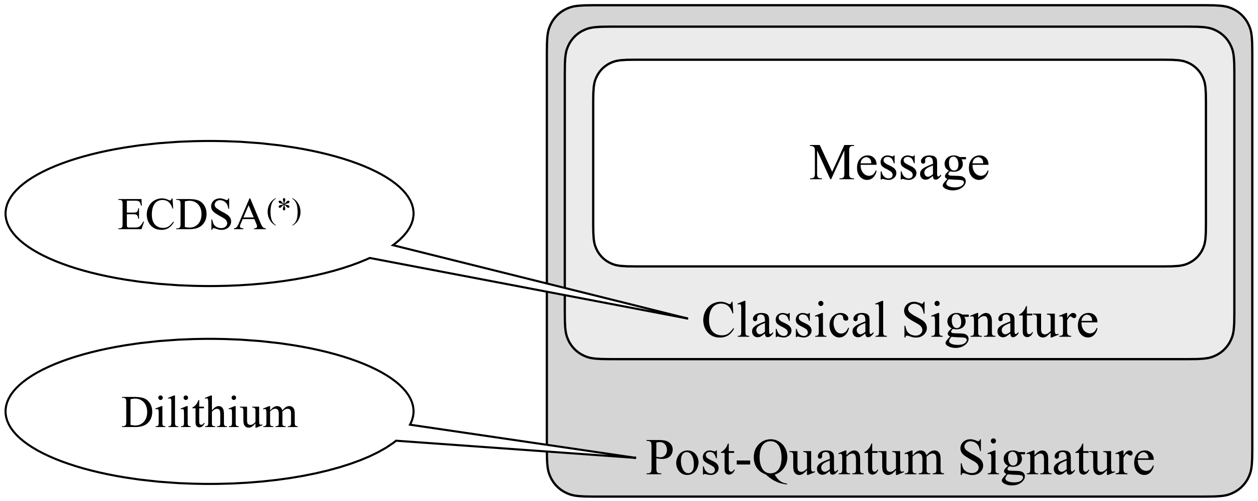
1. Replace C with Q
- Instead of calling the APIs implementing C, use APIs implementing Q
  - E.g.: instead of using Diffie-Hellman (C) for public-key encryption, use Kyber (Q)

2. Replace C with $C + Q$
- Use both, C as well as Q in "parallel"
  - E.g.: sign a document both, classically (C) as well as quantum (Q)
- Advantage: this is at least as secure as C only, and allows to gain trust in Q

…and some variants, e.g. concat then hash,…

3. Replace C with $C \circ Q$ (or with $Q \circ C$)
- Use both, C as well as Q one after the other
  - E.g.: sign a document first classically (C) and them with quantum (Q)

Example



$C \circ Q$ approach

(*) Elliptic Curve Digital Signature Algorithm

But This is Not Thus Simple

Some post-quantum algorithms have…

- excessively large signature sizes
- involve excessive processing
- require very large public and/or private keys
- require operations that are asymmetric between sending and receiving parties
- require the responder to generate a message based on the initiator's public value

Thus, no simple "drop-in" of the new algorithms

Secure implementation may need to address issues such as…

- public-key validation
- public-key reuse
- decryption failure even when all parameters are correctly implemented
- select new auxiliary functions (e.g., hash functions used with public-key algorithms for digital signature).

Performance and scalability issues…

- may demand significant modifications to protocols and infrastructures

Sample Discussion of Such Issues

NIST Cybersecurity White Paper csrc.nist.gov

Getting Ready for Post-Quantum Cryptography:

Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms

https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04282021.pdf

NIST | NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

MIGRATION TO POST-QUANTUM CRYPTOGRAPHY (PQC)

https://www.nccoe.nist.gov/sites/default/files/2023-08/mpqc-fact-sheet.pdf

QUANTUM-READINESS: MIGRATION TO POST-QUANTUM CRYPTOGRAPHY

NIST | NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY U.S. DEPARTMENT OF COMMERCE

https://www.nccoe.nist.gov/sites/default/files/2023-08/quantum-readiness-fact-sheet.pdf

NIST SPECIAL PUBLICATION 1800-38A

Migration to Post-Quantum Cryptography: Preparation for Considering the Implementation and Adoption of Quantum Safe Cryptography

https://www.nccoe.nist.gov/sites/default/files/2023-04/pqc-migration-nist-sp-1800-38a-preliminary-draft.pdf

Standardization

NIST PQC

(Post Quantum Cryptography)

Website: https://csrc.nist.gov/projects/post-quantum-cryptography

Process to submit, evaluate, recommend, and standardize quantum-safe algorithms

- Started already in 2016

CRYSTALS =
Cryptographic Suite for Algebraic Lattices

What is about to be standardized:

- CRYSTALS–KYBER: public-key encryption and key-establishment algorithm
  - https://pq-crystals.org/kyber/index.shtml
  - https://datatracker.ietf.org/doc/html/draft-cfrg-schwabe-kyber-02
- CRYSTALS–Dilithium, FALCON, and SPHINCS+: digital signatures
  - https://pq-crystals.org/dilithium/index.shtml

To be evaluated in future:

- BIKE, Classic McEliece, HQC, and ~~SIKE~~

https://arstechnica.com/information-technology/2022/08/sike-once-a-post-quantum-encryption-contender-is-koed-in-nist-smackdown/


NIST IR 8413-upd1

Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process

https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf

Published Standards

(August 2023)

Variant of CRYSTALS–KYBER: public-key encryption and key-establishment algorithm

- Module-Lattice-based Key-Encapsulation Mechanism
  - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf

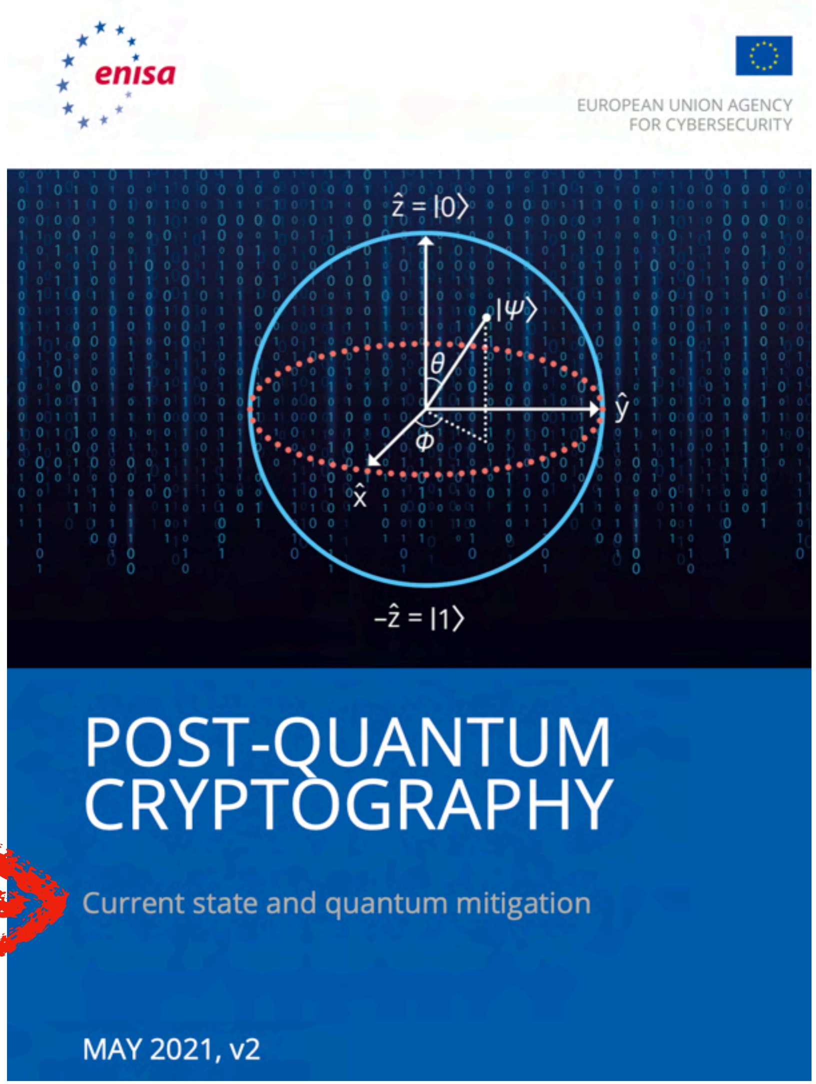
Variant of CRYSTALS-DILITHIUM: digital signature algorithm

- Module-Lattice-Based Digital Signature Standard
  - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf

Variant of SPHINCS+: digital signature algorithm

- Stateless Hash-Based Digital Signature Standard
  - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.ipd.pdf

High-Level Discussions: Standards & Implementations



What others are doing…
(selection only)

AWS Support

(https://aws.amazon.com/de/security/post-quantum-cryptography/)

AWS supports post-quantum cryptography in Key Management Service (KMS), Certificate Manager (ACM), Secrete Manager (ASM)

- Offering TLS endpoints supporting Diffie-Hellman and Kyber $\Rightarrow$ $C + Q$ approach

For example: Secure File Transfer (SFTP)

- Offering SSH (elliptic curve) Diffie-Hellman and Kyber $\Rightarrow$ $C + Q$ approach
- Based on OpenSSH using liboqs (https://github.com/open-quantum-safe/openssh)
- …and WolfSSL (https://www.wolfssl.com/)

```
Workgroup: Network Working Group
Internet-Draft:
draft-kampanakis-curdle-ssh-pq-ke-01
Published: 10 April 2023
Intended Status: Experimental
Expires: 12 October 2023
Authors: P. Kampanakis   D. Stebila              T. Hansen
         AWS             University of Waterloo  AWS

              Post-quantum Hybrid Key Exchange in SSH
```

https://datatracker.ietf.org/doc/draft-kampanakis-curdle-ssh-pq-ke/

For example: post-quantum crypto TLS with KMS API (https://docs.aws.amazon.com/kms/latest/developerguide/pqtls.html)

- Based on open source s2n-tls (https://github.com/aws/s2n-tls)
- Used for key-exchange only, no encryption
- Code samples: https://github.com/aws-samples/aws-kms-pq-tls-example

```
Network Working Group                              D. Stebila
Internet-Draft                         University of Waterloo
Intended status: Informational                     S. Fluhrer
Expires: 10 March 2024                          Cisco Systems
                                                    S. Gueron
                                                      U. Haifa
                                             7 September 2023

                  Hybrid key exchange in TLS 1.3
```

https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/

Microsoft Support

(https://blogs.microsoft.com/blog/2023/05/31/building-a-quantum-safe-future/)

Towards Quantum-Safe VPNs and Internet

Maran van Heesch* Niels van Adrichem* Thomas Attema*† Thijs Veugen*†

https://eprint.iacr.org/2019/1277.pdf

"...*trade-off between network and CPU overhead and the security levels defined by NIST.*"

Post-quantum crypto VPN

- Based on OpenVPN
  (https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/)
- Used between Redmond und MSFT Underwater Datacenter
  https://www.microsoft.com/en-us/research/project/post-quantum-crypto-tunnel-to-the-underwater-datacenter/

Post-quantum crypto TLS

- Based on OpenSSL (https://github.com/aws/s2n-tls)

Post-quantum crypto SSH

- Based on OpenSSH
  (https://www.microsoft.com/en-us/research/project/post-quantum-ssh/)

```
Workgroup: CURDLE
Internet-Draft:
draft-kampanakis-curdle-pq-ssh-00
Published: 21 October 2020
Intended Status: Experimental
Expires: 24 April 2021
Authors: P. Kampanakis   D. Stebila               M. Friedl
         Cisco Systems   University of Waterloo   OpenSSH
         T. Hansen   D. Sikeridis
         AWS         University of New Mexico

 Post-quantum public key algorithms for the Secure Shell (SSH) protocol
```

https://datatracker.ietf.org/doc/draft-kampanakis-curdle-pq-ssh/

Google Support

C + Q approach

Google protects internal communications from quantum threats

https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms

They address "store now, decrypt later" attacks, as these affect our data today

- Signature algorithm threats are not immediate
- Use of NTRU-HRSS (because of lack of some clarification from NIST about Kyber's IP status)
  - In 2016, a NewHope-based implementation of Chrome was released
  - IP issues forced Google to remove the implementation

C + Q approach

Support in Chrome 116

- Establishing symmetric keys based on Kyber and Elliptic Curves (X25519)
  - …over TLS
  - For Google servers over TCP and QUIC

Collaboration with Cloudflare (see next)

Cloudflare

https://blog.cloudflare.com/post-quantum-crypto-should-be-free/

Open source experimental cryptography suite called CIRCL https://github.com/cloudflare/circl

- Tool for experimental deployment of post-quantum cryptographic
- Support of Kyber, Dilithium,…
- Under BSD-3-Clause License.

Application to TLS 1.3 (https://blog.cloudflare.com/post-quantum-for-all/)
https://blog.cloudflare.com/the-tls-post-quantum-experiment/

- …nice reading :-)
- Based on BoringSSL and Go

  https://blog.cloudflare.com/experiment-with-pq/#boringssl

C + Q approach

Application to Cloudflare Tunnel (https://blog.cloudflare.com/post-quantum-tunnel/)

Implementing and Measuring KEMTLS

Sofía Celi[1], Armando Faz-Hernández[1], Nick Sullivan[1], Goutam Tamvada[2], Luke Valenta[1], Thom Wiggers[3], Bas Westerbaan[4], and Christopher A. Wood[1]

https://eprint.iacr.org/2021/1019.pdf

IBM

IBM … z16 is … quantum-safe system … to protect data against future threats … [of] quantum computing.

- *…help businesses tackle threats such as "harvest now, decrypt later" attacks*

Quantum-safe cryptography support for key management and transactions in IBM Cloud

- Data exchange between cloud secured by using a quantum-safe algorithm

IBM Key Protect provides lifecycle management for encryption keys

- Use a quantum-safe cryptography enabled Transport Layer Security (TLS) connection

OpenShift

- Quantum-safe cryptography secured TLS connections to protect data-in-transit

WSO2

https://wso2.com/

- Quantum resistant communication with/in products
  - API Manager, Identity Server, Choreo
- Protect identities against quantum attacks
  - Identity Server, Asgardeo
- *Ballerina as quantum resistant programming language*

  https://ballerina.io/
  - …a language for building cloud-native applications
  - …a language for realizing integration solutions
- Based on liboqs

WolfSSL

Quantum resistant cURL

- TLS 1.3 using WolfSSL

(https://www.wolfssl.com/post-quantum-curl/)

Quantum resistant MQTT

(https://www.wolfssl.com/wolfmqtt-post-quantum-kyber-falcon/)

X9

https://x9.org/new-update-to-cryptographic-key-management-standard/

Financial Industry standards

X9.69 Framework For Key Management Extensions

- ≈ Methods for generation and control of keys used in symmetric cryptographic algorithms
- Includes methods for quantum computing protection
- Framework supporting an algorithm at any key length
- Support compliance with HIPAA, Europe’s GDPR and other privacy regulations

Open Source that is available

All implementations have the caveat that they are experimental prototypes and that they don’t guarantee production readiness!

OQS: Open Quantum Safe

(https://openquantumsafe.org/)

Open-source project to support development and prototyping of quantum-resistant cryptography

Two main lines of work

- liboqs — open source C library for quantum-resistant cryptographic algorithms
- Prototype integrations into protocols and applications

Everything is in Github repositories

Lots of contributors
(from companies like AWS, IBM, Microsoft, Cloudflare, Intel, Cisco as well as universities)

liboqs

(https://openquantumsafe.org/liboqs/) (https://github.com/open-quantum-safe/liboqs)

Open source C library for quantum-safe cryptographic algorithms (MIT license)
[But it uses some third-party libraries with different licenses: list is provided!]

- Implementations of key encapsulation mechanisms (KEM) and digital signature algorithms
- Common API for these algorithms
  https://openquantumsafe.org/liboqs/api/
- Test and benchmarking routines

Wrappers for C++, Go, Rust, Java, .Net, Python

https://openquantumsafe.org/liboqs/wrappers

Multi-platform

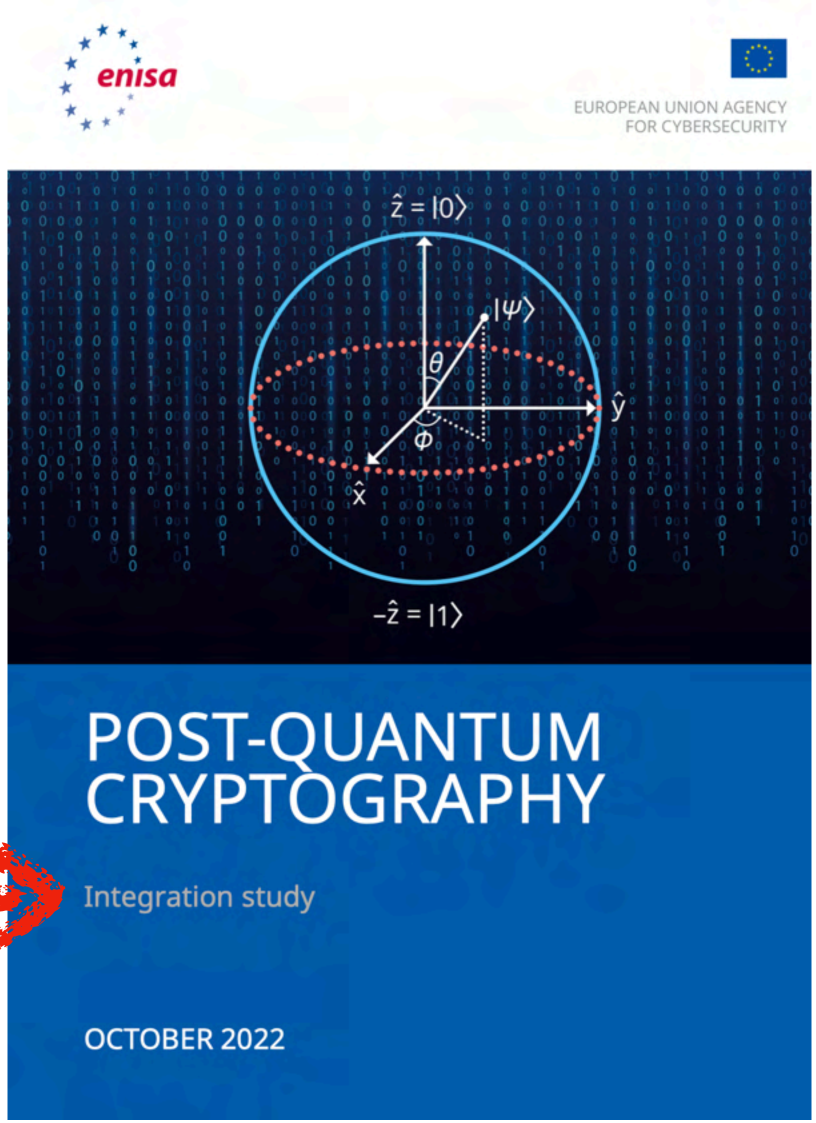
- Linux, macOS, Windows
- x86_64 and ARM (no ARM for Windows)
- clang, gcc, MSFT compilers

Algorithms include Kyber and Dilithium (https://openquantumsafe.org/liboqs/algorithms/)

Applications to several protocols available

- TLS, SSH, X.509, S/MIME (https://openquantumsafe.org/applications/)
- External usages (https://openquantumsafe.org/applications/external.html)

How to Build Protocols & Applications With These Standards



OQS-OpenSSH

(https://github.com/open-quantum-safe/openssh)

OQS-OpenSSH is a fork of OpenSSH that adds quantum-safe cryptography to enable its use and evaluation in the SSH protocol.

- Part of the Open Quantum Safe (OQS) project,
- Uses liboqs

Key-Exchange: based on Kyber and others

Digital Signature: based on Dilithium and others

OQS-OpenSSL

(https://github.com/open-quantum-safe/openssl)

OQS-OpenSSL is a fork of OpenSSL that adds quantum-safe cryptography in TLS 1.3

- Part of the Open Quantum Safe (OQS) project,
- Uses liboqs

Key-Exchange: based on Kyber and others

Digital Signature: based on Dilithium and others

OQS Demos

(https://github.com/open-quantum-safe/oqs-demos)

These demos show how to enable quantum-safe cryptography in various applications

- Part of the Open Quantum Safe (OQS) project,
- Uses liboqs
- Include pre-build Docker files und build instructions

Supported are, e.g.:

- curl — https://github.com/open-quantum-safe/oqs-demos/tree/main/curl
- vpn — https://github.com/open-quantum-safe/oqs-demos/tree/main/openvpn
- httpd — https://github.com/open-quantum-safe/oqs-demos/tree/main/httpd
- nginx — https://github.com/open-quantum-safe/oqs-demos/tree/main/nginx
- envoy — https://github.com/open-quantum-safe/oqs-demos/tree/main/envoy
- mosquitto — https://github.com/open-quantum-safe/oqs-demos/tree/main/mosquitto
- haproxy — https://github.com/open-quantum-safe/oqs-demos/tree/main/haproxy

Dilithium

Reference implementation: https://github.com/pq-crystals/dilithium
But also: https://github.com/itzmeanjan/dilithium

- Zero-dependency, header-only C++ library
- offering key generation, signing & verification API
- for three NIST security level (i.e. 2, 3, 5) parameters

Algorithm	What does it do ?
KeyGen	It takes a 32 -bytes seed, which is used for deterministically computing both public key and secret key i.e. keypair.
Sign	It takes a secret key and a N (>0) -bytes message as input, which is used for deterministically (default)/ randomly (in this case, you must supply 64 uniform random sampled bytes as seed) signing message, producing signature bytes.
Verify	It takes a public key, N (>0) -bytes message and signature, returning boolean value, denoting status of successful signature verification operation.

NIST Security Level	2	3	5
Output Size			
public key size (bytes)	1312	1952	2592
signature size (bytes)	2420	3293	4595

Key Generation

```
#include "dilithium2.hpp"
#include "prng.hpp"

int main() {
    uint8_t seed[32];
    uint8_t pubkey[dilithium2::PubKeyLen];
    uint8_t seckey[dilithium2::SecKeyLen];

    // Sample seed bytes from PRNG
    prng::prng_t prng;
    prng.read(seed, sizeof(seed));

    dilithium2::keygen(seed, pubkey, seckey);

    // ...

    return 0;
}
```

Computing Signature

```
int main() {

  uint8_t msg[32];
  uint8_t sig[dilithium2::SigLen];

  // Sample psuedo-random message, to be signed
  prng.read(msg, sizeof(msg));

  // Default behaviour is deterministic signing and
  // you can safely pass null pointer for last parameter
  // i.e. random seed. It won't be access, in case you adopt
  // default deterministic signing.
  dilithium2::sign(seckey, msg, mlen, sig, nullptr);

  // ...

  return 0;
}
```

Verifying Signature

```
int main() {

  bool flg = dilithium2::verify(pubkey, msg, mlen, sig);
  assert(flg);

  return 0;
}
```

Kyber

Reference implementation: https://github.com/pq-crystals/kyber

But also: https://github.com/itzmeanjan/kyber

- Zero-dependency, header-only C++ library
- offering public key encryption and key encapsulation mechanism API
- only works with 32 byte messages

Algorithm	Input	Output
PKE KeyGen	-	Public Key and Secret Key
Encryption	Public Key, 32 -bytes message and 32 -bytes random coin	Cipher Text
Decryption	Secret Key and Cipher Text	32 -bytes message

KYBER512

Sizes (in Bytes)	
sk:	1632 (or 32)
pk:	800
ct:	768

KYBER768

Sizes (in Bytes)	
sk:	2400 (or 32)
pk:	1184
ct:	1088

KYBER1024

Sizes (in Bytes)	
sk:	3168 (or 32)
pk:	1568
ct:	1568

Usage

```
main()
{
  ...

  prng::prng_t prng;

  prng.read(d, sizeof(d));
  prng.read(z, sizeof(z));
  prng.read(m, sizeof(m));

  kyber512_kem::keygen(d, z, pkey, skey);
  auto skdf = kyber512_kem::encapsulate(m, pkey, cipher);
  auto rkdf = kyber512_kem::decapsulate(skey, cipher);

  uint8_t sender_key[32]{};
  skdf.squeeze(sender_key, sizeof(sender_key));

  uint8_t receiver_key[32]{};
  rkdf.squeeze(receiver_key, sizeof(receiver_key));

  assert(std::ranges::equal(sender_key, receiver_key));
  return 0;
}
```

On 12th Gen Intel(R) Core(TM) i7-1260P [Compiled with GCC]

```
2023-06-08T17:00:31+04:00
Running ./bench/a.out
Run on (16 X 3562.43 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x8)
  L1 Instruction 32 KiB (x8)
  L2 Unified 1280 KiB (x8)
  L3 Unified 18432 KiB (x1)
Load Average: 0.43, 0.40, 0.39
--------------------------------------------------------------------------------
Benchmark                     Time             CPU   Iterations items_per_second
--------------------------------------------------------------------------------
dilithium2_keygen          59.5 us         59.4 us        11768       16.8301k/s
dilithium2_sign/32          189 us          189 us         3696       5.28747k/s
dilithium2_verify/32       65.9 us         65.9 us        10567       15.1858k/s
dilithium3_keygen          98.2 us         98.2 us         7140       10.1804k/s
dilithium3_sign/32          933 us          933 us         2529        1071.29/s
dilithium3_verify/32        105 us          105 us         6653        9.4921k/s
dilithium5_keygen           164 us          164 us         4374       6.11558k/s
dilithium5_sign/32          273 us          273 us         2560       3.65763k/s
dilithium5_verify/32        173 us          173 us         4052       5.78499k/s
```

Message Signing Algorithm (Deterministic)	Min. Exec. Time	Max. Exec. Time	Median Exec. Time	Mean Exec. Time
Dilithium2	122 us	987 us	**256 us**	351 us
Dilithium3	182 us	1309 us	**417 us**	526 us
Dilithium5	274 us	2603 us	**533 us**	610 us

On 12th Gen Intel(R) Core(TM) i7-1260P [Compiled with Clang]

```
2023-06-08T17:15:22+04:00
Running ./bench/a.out
Run on (16 X 3436.72 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x8)
  L1 Instruction 32 KiB (x8)
  L2 Unified 1280 KiB (x8)
  L3 Unified 18432 KiB (x1)
Load Average: 0.60, 0.66, 0.58
--------------------------------------------------------------------------------
Benchmark                    Time             CPU   Iterations items_per_second
--------------------------------------------------------------------------------
dilithium2_keygen         45.4 us         45.3 us        15245       22.0851k/s
dilithium2_sign/32         277 us          277 us         3828       3.60687k/s
dilithium2_verify/32      50.7 us         50.7 us        13791       19.7402k/s
dilithium3_keygen         77.8 us         77.7 us         9010       12.8624k/s
dilithium3_sign/32         331 us          331 us         5199       3.02039k/s
dilithium3_verify/32      81.2 us         81.2 us         8599       12.3189k/s
dilithium5_keygen          127 us          127 us         5537       7.87793k/s
dilithium5_sign/32         576 us          575 us         1000       1.73773k/s
dilithium5_verify/32       134 us          134 us         5199        7.4441k/s
```

Message Signing Algorithm (Deterministic)	Min. Exec. Time	Max. Exec. Time	Median Exec. Time	Mean Exec. Time
Dilithium2	88.5 us	749 us	**230 us**	258 us
Dilithium3	135 us	1509 us	**299 us**	425 us
Dilithium5	210 us	1313 us	**302 us**	467 us

Crypto Agility

Crypto-Agility

...meant to protect against the failure of encryption procedures

- Since (some) post-quantum procedures may also be cracked, the new infrastructure must make it as easy as possible to exchange such procedures

- Instead of a simple exchange, there are also approaches to combine procedures in such a way that the overall procedure remains secure, even if a procedure involved is cracked

https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms

Crypto-Agility: Open Source

SANDWICH

What is Sandwich?

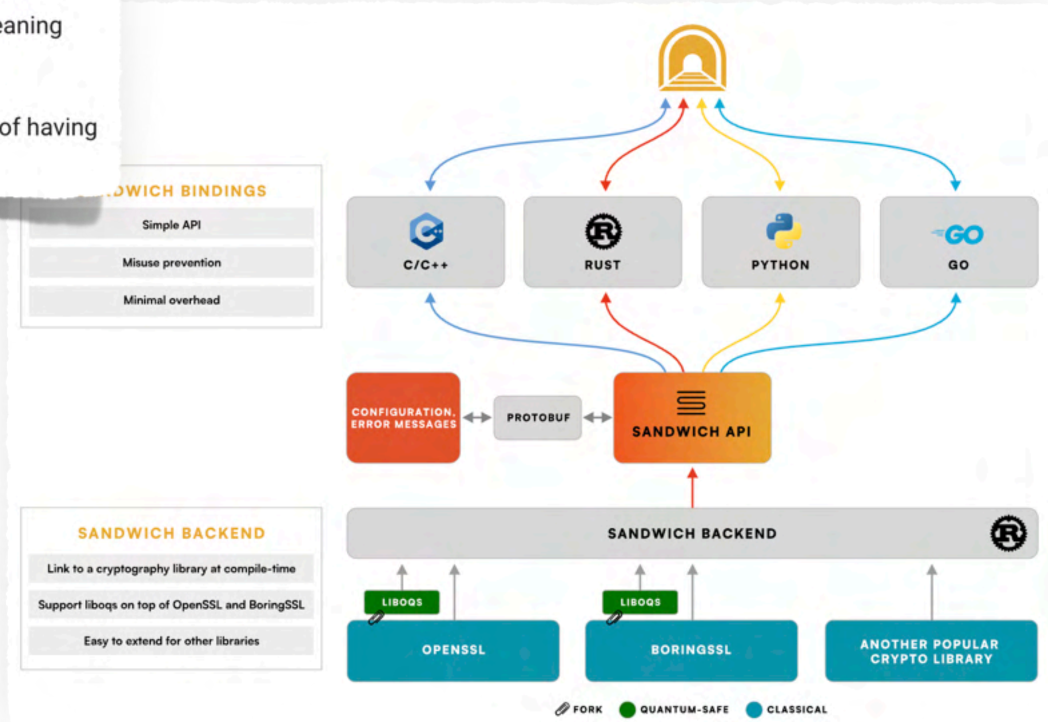
Sandwich provides a simple, unified, and hard to misuse API for developers to use cryptographic algorithms and protocols of their choice in their applications. Sandwich is written in Rust, and provides a C API with bindings for Python and Go. This API is implemented through various cryptographic libraries (OpenSSL and BoringSSL), and in particular supports libOQS, meaning **Sandwich enables post-quantum cryptography**.

One goal of the library is to enable dynamic cryptographic agility, without the necessity of having to recompile or redeploy updated software.

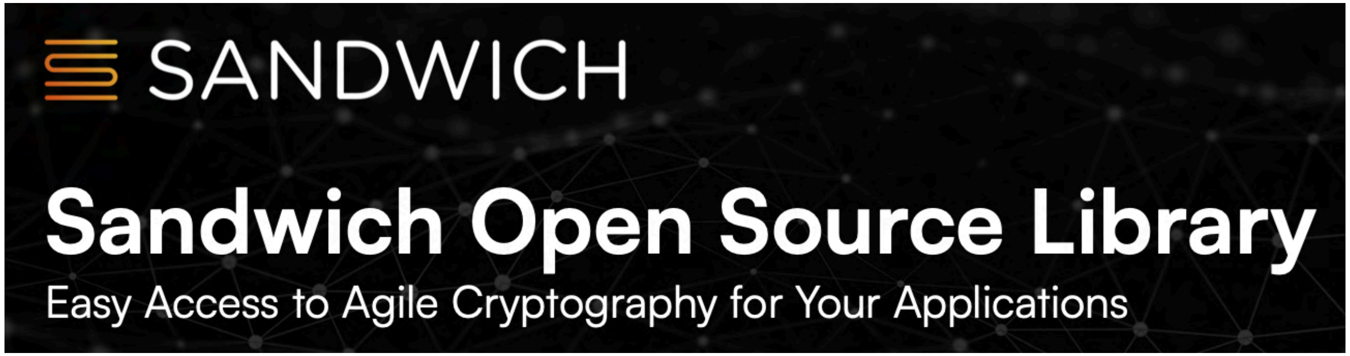
https://sandbox-quantum.github.io/sandwich/

SANDWICH
Agile Cryptography for Developers

https://www.sandboxaq.com/solutions/sandwich



Sandwich



https://go.sandboxaq.com/rs/175-UKR-711/images/Sandwich-datasheet.pdf

v0.1.0: initial public version `Latest`

Initial public version

▾ Assets 6

sandwich-linux-amd64-v0.1.0.tar.bz2	28.4 MB	Aug 8
sandwich-linux-arm64-v0.1.0.tar.bz2	25.3 MB	Aug 8
sandwich-macos-arm64-v0.1.0.tar.bz2	21.4 MB	Aug 8
sandwich-macos-x86_64-v0.1.0.tar.bz2	25.3 MB	Aug 8
Source code (zip)		Aug 8
Source code (tar.gz)		Aug 8

https://github.com/sandbox-quantum/sandwich/releases
https://github.com/sandbox-quantum/sandwich

Agenda

Classical Encryption & Discrete Logarithms

Quantum Computing & Discrete Logarithms (Shor)

Lattice-Based Cryptography

Dilithium & Kyber

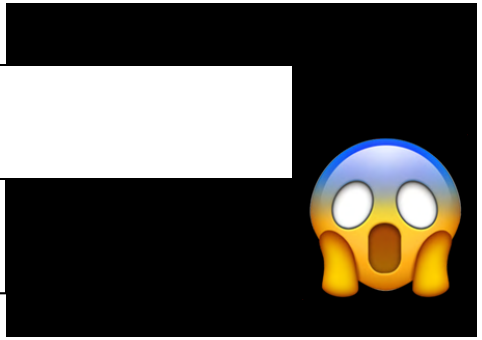
NIST & Industry

Summary

Summary

- Today's crypto infrastructure relies on hardness of discrete logarithm problem
- Quantum computers will be able to crack this infrastructure mid-term
- Lattice-based cryptography seems to be a rescue
- Standards for post-quantum security are under way
- Except for "big players", industry awareness for the problem is lacking
- Open source prototypes are available
- Quantum-safe infrastructure must be agile

$T_{migration}$	T_{shelf}
$T_{collapse}$	



The End