FFI Norwegian Defence Research Establishment

Quantum safe cryptography

Martin Strand, PhD, senior researcher

Foto: AdobeStock

About me: Martin Strand

- BS, MS in algebra and cryptography
- PhD in cryptography from Norwegian University of Science and Technology (NTNU), 2018
- Used to be expert in fully homomorphic encryption
- Senior researcher at the Norwegian Defense Research Establishment (FFI)
- Interests: Secure computations, lattice crypto, group key exchange, constrained and unmanned devices

Agenda

1. Classical cryptography
2. Quantum algorithms
3. Post-quantum crypto standardisation
4. Lattice cryptography
5. Setbacks
6. Hybrid cryptography
7. Some benchmarking
8. Other security properties and protocols
9. What about privacy?
10. Bonus content

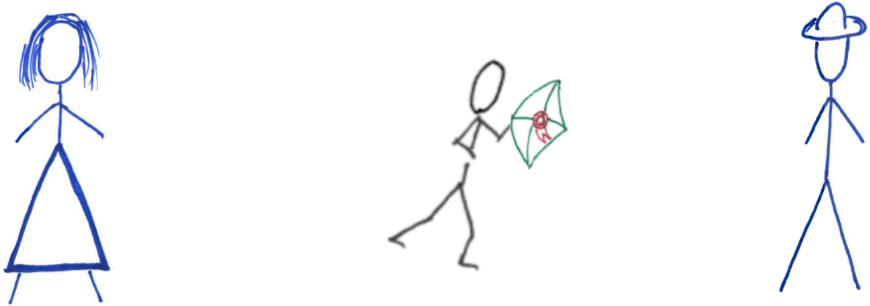
Crypto means cryptography

Security – against whom?

Security – against whom?

Security – against whom?

Security – against whom?



Classical cryptography

Soft start: Who can tell the time?

- Pretend it’s 11 o’clock. What’s the time in 26 hours?

Examples

Let $n = 17$.

- $10 + 12 \equiv \pmod{17}$
- $4 \cdot 5 \equiv \pmod{17}$
- $3^3 = 27 \equiv \pmod{17}$

(These computations work particularly well when n is prime.)

Examples

Let $n = 17$.

- $10 + 12 \equiv 5 \pmod{17}$
- $4 \cdot 5 \equiv \quad \pmod{17}$
- $3^3 = 27 \equiv \quad \pmod{17}$

(These computations work particularly well when n is prime.)

Examples

Let $n = 17$.

- $10 + 12 \equiv 5 \pmod{17}$
- $4 \cdot 5 \equiv 3 \pmod{17}$
- $3^3 = 27 \equiv \pmod{17}$

(These computations work particularly well when n is prime.)

Examples

Let $n = 17$.

- $10 + 12 \equiv 5 \pmod{17}$
- $4 \cdot 5 \equiv 3 \pmod{17}$
- $3^3 = 27 \equiv 10 \pmod{17}$

(These computations work particularly well when n is prime.)

How to agree on a secret

Martin		you
Random $0 < a < 17$		
Set $A \equiv 3^a \pmod{17}$	A $\longrightarrow$	Random $0 < b < 17$
	B $\longleftarrow$	Set $B \equiv 3^b \pmod{17}$
Set $C_1 \equiv B^a \pmod{17}$		Set $C_2 \equiv A^b \pmod{17}$

How to agree on a secret

Martin		you
Random $0 < a < 17$		
Set $A \equiv 3^a \pmod{17}$	$\xrightarrow{A}$	Random $0 < b < 17$
	$\xleftarrow{B}$	Set $B \equiv 3^b \pmod{17}$
Set $C_1 \equiv B^a \pmod{17}$		Set $C_2 \equiv A^b \pmod{17}$

Claim

We have $C = C_1 = C_2$ and nobody else knows C unless they are really good at computing discrete logarithms.

Why does it work?

$$
\begin{aligned}
C_1 &= B^a = \left(3^b\right)^a \\
&= 3^{ba} = 3^{ab} \\
&= \left(3^a\right)^b = A^b \\
&= C_2
\end{aligned}
$$

Let's try!

Martin		you
Random $0 < a < 17$		
Set $A \equiv 3^a \pmod{17}$	A →	Random $0 < b < 17$
	← B	Sett $B \equiv 3^b \pmod{17}$
Set $C_1 \equiv B^a \pmod{17}$		Set $C_2 \equiv A^b \pmod{17}$

$3^0 \equiv 1$	$3^6 \equiv 15$	$3^{12} \equiv 4$
$3^1 \equiv 3$	$3^7 \equiv 11$	$3^{13} \equiv 12$
$3^2 \equiv 9$	$3^8 \equiv 16$	$3^{14} \equiv 2$
$3^3 \equiv 10$	$3^9 \equiv 14$	$3^{15} \equiv 6$
$3^4 \equiv 13$	$3^{10} \equiv 8$	$3^{16} \equiv 1$
$3^5 \equiv 5$	$3^{11} \equiv 7$	$3^{17} \equiv 3$

How big must p be?

On the back of the envelope

The problem

Given $p, g, h = g^a \pmod p$, find a.

Assumptions

- 2^{40} operations per second (Intel Core i9-13900KS: 2^{37})
- About one per person, say 10 000 000 000 CPUs
- 31 536 000 seconds in a year
- In total, 2^{98} operations per year

On the back of the envelope

The problem

Given $p, g, h = g^a \pmod{p}$, find a.

Assumptions

- 2^{40} operations per second (Intel Core i9-13900KS: 2^{37})
- About one per person, say 10 000 000 000 CPUs
- 31 536 000 seconds in a year
- In total, 2^{98} operations per year
- (Bitcoin: $\sim 2^{92}$ operations per year, $1/64$ of this)

Attack: Brute force

Set $p \approx 2^{128}$.

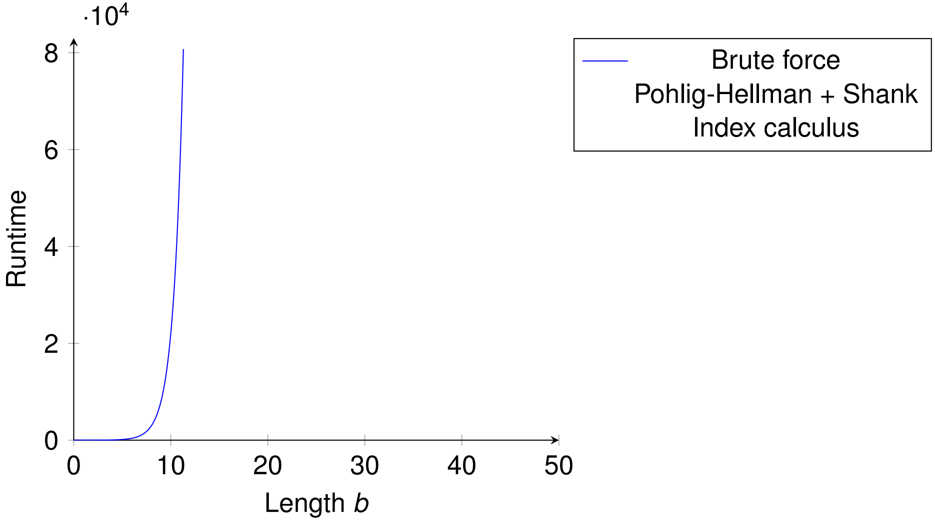
Attack: Brute force

Set $p \approx 2^{128}$. It will then take $2^{128}/2^{98} \approx 1\,000\,000\,000$ years to find a.

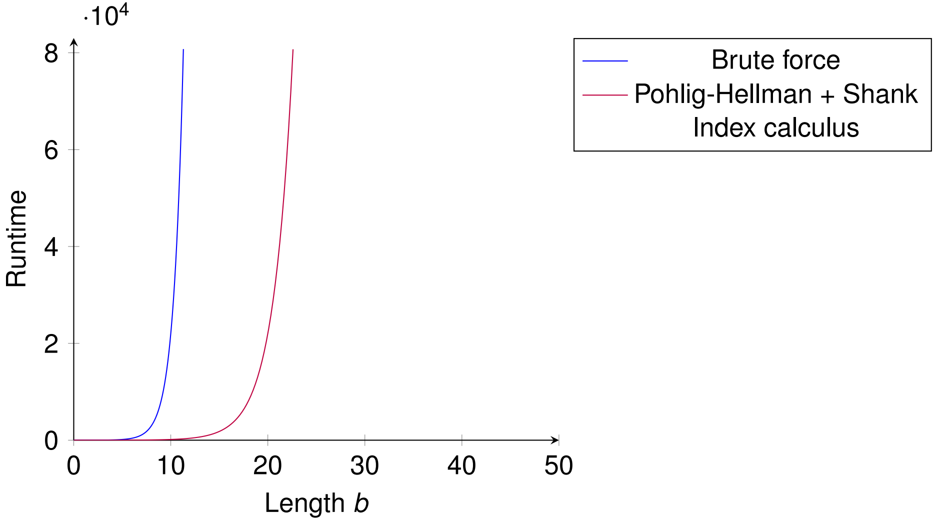
Asymptotic runtime

$\mathcal{O}(p)$

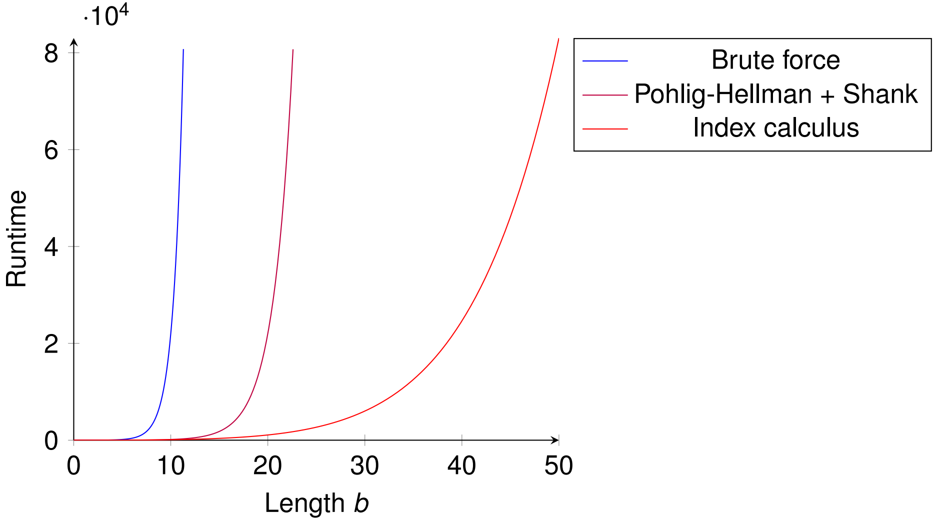
Somewhat smarter algorithms



Somewhat smarter algorithms



Somewhat smarter algorithms



Number of atoms in the universe

100 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000

Number of unique chess games

1 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000

A suitable prime for Diffie-Hellman

```
17 371 633 710 861 526 985 402 756 155 605 996 322 196 257 048 455
   675 141 997 211 607 897 588 436 880 112 664 345 732 402 516 434
   975 116 670 023 472 796 825 233 643 612 395 266 186 808 119 984
   996 372 379 602 426 678 900 493 286 192 039 475 551 678 848 776
   585 415 169 949 664 415 820 483 514 690 301 509 982 058 398 659
   940 050 744 425 005 234 342 360 377 140 221 362 953 519 273 046
   483 446 364 930 471 865 451 176 965 825 059 235 201 349 014 188
   384 323 322 347 988 836 585 004 216 878 741 293 400 993 565 478
   114 200 002 489 905 246 623 078 674 988 568 740 682 222 428 856
   692 842 421 774 076 905 917 061 448 967 466 083 362 856 797 534
   215 180 379 822 041 036 186 832 388 654 983 120 685 889 564 412
   266 789 511 781 064 026 694 452 185 724 178 282 543 463 162 021
   793 730 933 403 449 281 865 751 197 897 543 205 563
```

Can we use a smaller number?

Curve25519

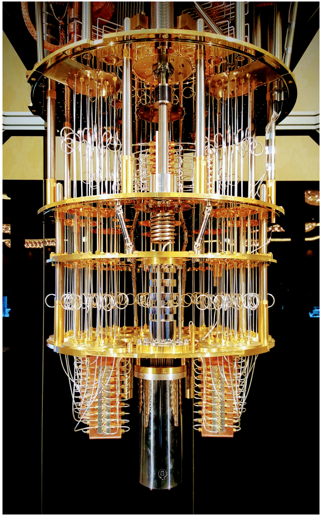
57 896 044 618 658 097 711 785 492 504 343 953 926 634 992 332 820 282 019 728 792 003 956 564 819 949

We often use asymmetric encryption to establish keys for symmetric encryption

Our primitive toolbox

	Symmetric	**Asymmetric**
Confidentiality	AES, ChaCha20, ...	Diffie–Hellman, ElGamal, ...
Integrity	HMAC, KMAC, AES-GCD, ...	RSA signatures, DSA, ECDSA, ...

Quantum algorithms



(Photo: Lars Plougmann (CC BY-SA 2.0))

Shor’s algorithm

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

Peter W. Shor†

Abstract

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

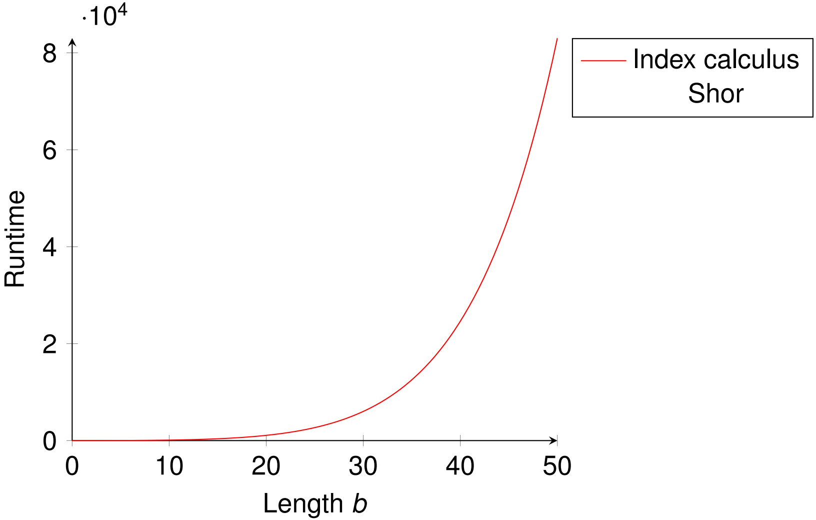
How to factor efficiently

Given $N = pq$, find p, q

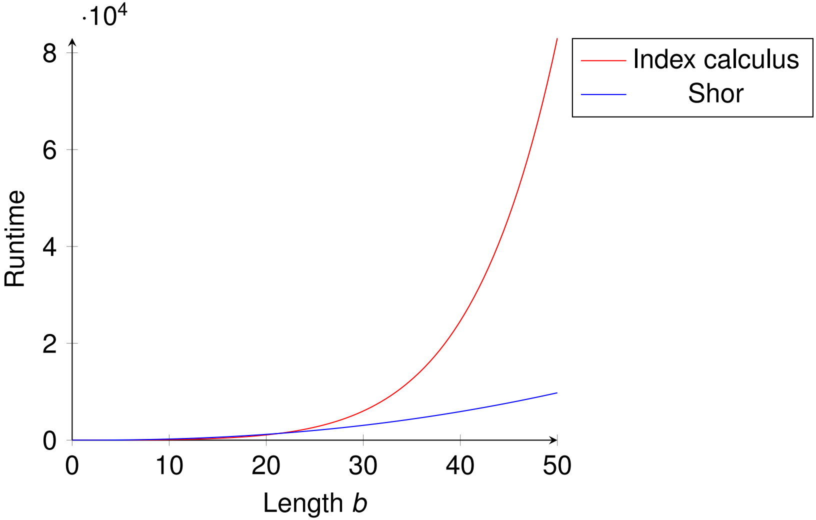
1. Choose $1 < a < N$ such that $\gcd(a, N) = 1$
2. Find smallest $r > 0$ such that $a^r \equiv 1 \pmod{N}$
3. If $2 \nmid r$, go to 1
4. If $a^{r/2} \equiv -1 \pmod{N}$, go to 1
5. Let $d = \gcd(a^{r/2} - 1, N)$

Can show: d is a non-trivial factor of N.

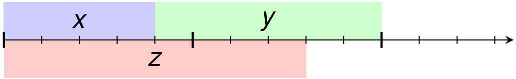
Shor's algorithm vs. dlog and factoring



Shor's algorithm vs. dlog and factoring



Quantum computers are coming (?)



Theorem (Mosca)

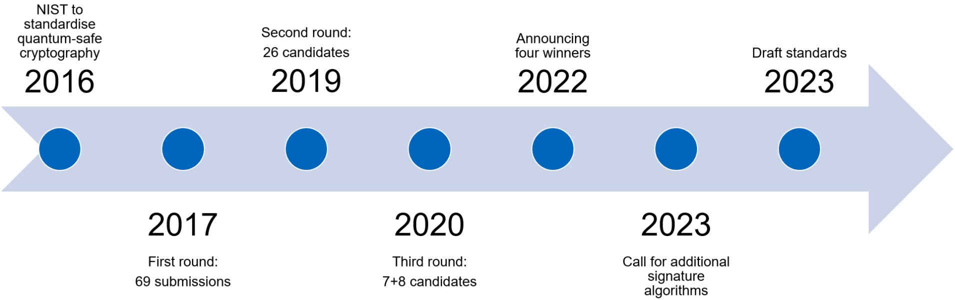
If $x + y > z$, be worried now.

Our primitive toolbox, pt. II

	Symmetric	**Asymmetric**
Confidentiality	AES, ChaCha20, ...	
Integrity	HMAC, KMAC, AES-GCD, ...	

We’ve been here before

Post-quantum crypto standardisation


NIST to standardise quantum-safe cryptography
2016
2017
First round:
69 submissions
Second round:
26 candidates
2019
2020
Third round:
7+8 candidates
Announcing four winners
2022
2023
Call for additional signature algorithms
Draft standards
2023

2nd round candidates

Encryption

- Classic McEliece
- CRYSTALS-KYBER
- NTRU
- SABER

- BIKE
- FrodoKEM
- HQC
- NTRU Prime
- SIKE

Signatures

- CRYSTALS-DILITHIUM
- FALCON
- Rainbow

- GeMSS
- Picnic
- SPHINCS+

3rd round candidates

Encryption

- CRYSTALS-KYBER (lattices)

Signatures

- CRYSTALS-DILITHIUM (lattices)
- FALCON (lattices)
- SPHINCS+ (hash functions)

4th round candidates

- BIKE (error correcting codes)
- HQC (error correcting codes)
- SIKE (isogenies)
- Classic McEliece (error correcting codes)

2022: The four picks

Key encapsulation	CRYSTALS-Kyber	Peter Schwabe, Roberto Avanzi, Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehle, Jintai Ding
Signatures	CRYSTALS-Dilithium	Vadim Lyubashevsky, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehle, Shi Bai
	FALCON	Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang
	SPHINCS+	Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, Ward Beullens

Lattice cryptography

Kyber: Oversimplified

Let k be an integer.

KGen

1. Choose a matrix A from $\mathbb{R}^{k \times k}$
2. Choose a vector $\mathrm{sk} = \boldsymbol{s}$ from $\mathbb{R}^k$
3. Compute $\boldsymbol{t} = A\boldsymbol{s}$, and set $\mathrm{pk} = (\boldsymbol{t}, A)$

Kyber: Oversimplified

Let k be an integer.

KGen
1. Choose a matrix A from $\mathbb{R}^{k \times k}$
2. Choose a vector $\mathrm{sk} = \boldsymbol{s}$ from $\mathbb{R}^k$
3. Compute $\boldsymbol{t} = A\boldsymbol{s}$, and set $\mathrm{pk} = (\boldsymbol{t}, A)$

Enc(pk, m)
1. Choose $\boldsymbol{r}$ from $\mathbb{R}^k$
2. Set $\boldsymbol{u} = A^T \boldsymbol{r}$ and $v = \boldsymbol{t}^T \cdot \boldsymbol{r} + m$
3. Return $c = (\boldsymbol{u}, v)$

Kyber: Oversimplified

Let k be an integer.

KGen

1. Choose a matrix A from $\mathbb{R}^{k \times k}$
2. Choose a vector sk $= \boldsymbol{s}$ from $\mathbb{R}^k$
3. Compute $\boldsymbol{t} = A\boldsymbol{s}$, and set pk $= (\boldsymbol{t}, A)$

Enc(pk, m)

1. Choose $\boldsymbol{r}$ from $\mathbb{R}^k$
2. Set $\boldsymbol{u} = A^T\boldsymbol{r}$ and $v = \boldsymbol{t}^T \cdot \boldsymbol{r} + m$
3. Return $c = (\boldsymbol{u}, v)$

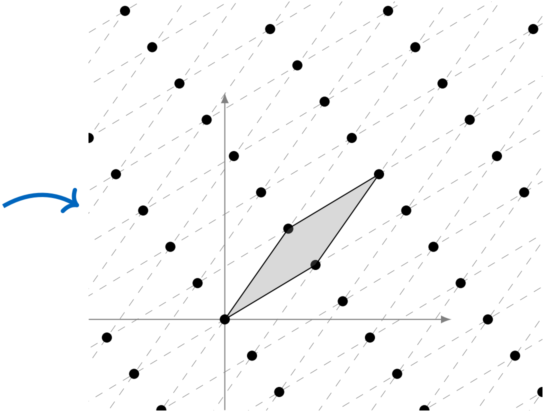
Dec(sk, c) Compute $w = v - \boldsymbol{s}^T \cdot \boldsymbol{u}$ and return w

(For those reading this after the presentation: Be aware that this is wrong by purpose; please use a different source to get the actual algorithms.)

Learning with errors

$$a_{1,1}s_1 + \dots a_{1,n}s_n + e_1 = b_1$$
$$a_{2,1}s_1 + \dots a_{2,n}s_n + e_2 = b_2$$
$$a_{3,1}s_1 + \dots a_{3,n}s_n + e_3 = b_3$$
$$a_{4,1}s_1 + \dots a_{4,n}s_n + e_4 = b_4$$
$$a_{5,1}s_1 + \dots a_{5,n}s_n + e_5 = b_5$$
$$\vdots$$

Given A, b, and if e_i are small, what is s?



Lattices

Let $\mathbb{R}^n \cong V = \mathrm{span}\{\boldsymbol{b}_1, \ldots \boldsymbol{b}_n\}$ be a real vector space.
Then

$$L = \left\{ \sum_{i=1}^{n} a_i \boldsymbol{b}_i \mid a_i \in \mathbb{Z} \right\} \subset \mathbb{R}^n$$

is the lattice generated by $\{\boldsymbol{b}_1, \ldots \boldsymbol{b}_n\}$.

Lattices

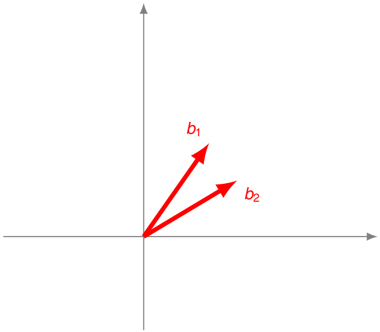
Let $\mathbb{R}^n \cong V = \text{span}\{\boldsymbol{b}_1, \dots \boldsymbol{b}_n\}$ be a real vector space.
Then

$$L = \left\{ \sum_{i=1}^{n} a_i \boldsymbol{b}_i \mid a_i \in \mathbb{Z} \right\} \subset \mathbb{R}^n$$

is the lattice generated by $\{\boldsymbol{b}_1, \dots \boldsymbol{b}_n\}$.

Example

Consider $\mathbb{R}^2 \cong \text{span}\{(2,3),(3,2)\}$



Lattices

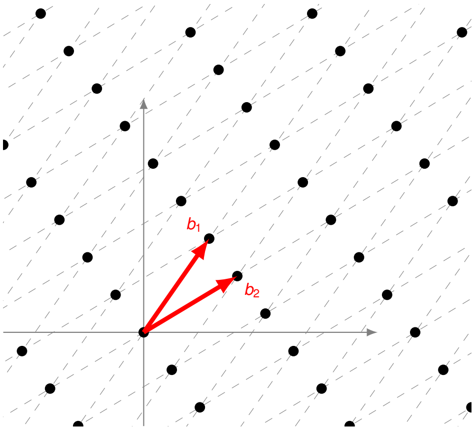
Let $\mathbb{R}^n \cong V = \text{span}\,\{\boldsymbol{b}_1, \ldots \boldsymbol{b}_n\}$ be a real vector space.
Then

$$L = \left\{ \sum_{i=1}^{n} a_i \boldsymbol{b}_i \mid a_i \in \mathbb{Z} \right\} \subset \mathbb{R}^n$$

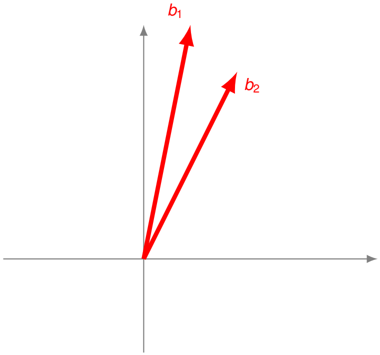
is the lattice generated by $\{\boldsymbol{b}_1, \ldots \boldsymbol{b}_n\}$.

Example

Consider $\mathbb{R}^2 \cong \text{span}\,\{(2,3),(3,2)\}$



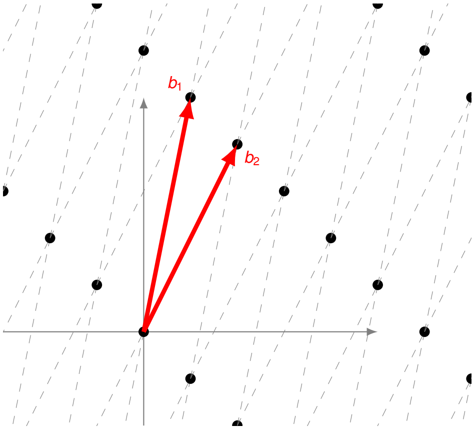
Lattice problems



Lattice problems

Shortest Vector Problem

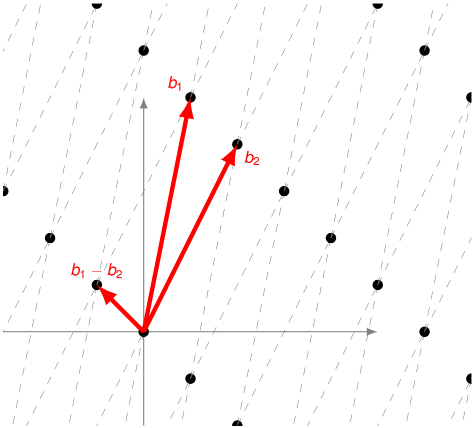
Given a basis for L, find the shortest vector in V that is also a point in L.



Lattice problems

Shortest Vector Problem

Given a basis for L, find the shortest vector in V that is also a point in L.



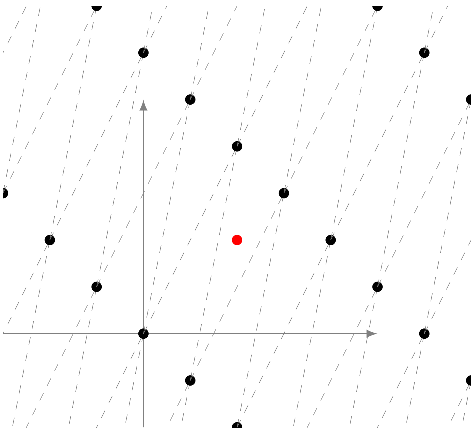
Lattice problems

Shortest Vector Problem

Given a basis for L, find the shortest vector in V that is also a point in L.

Closest Vector Problem

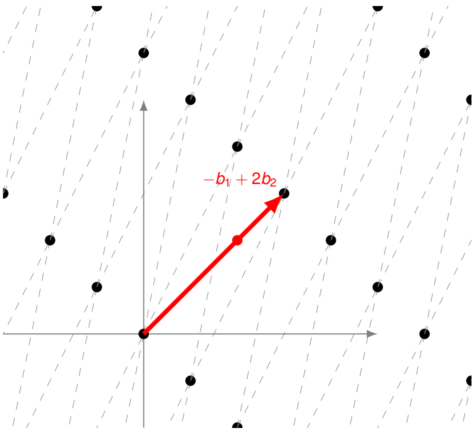
Given a basis for L and a point v in V, find closest lattice point to v in L.



Lattice problems

Shortest Vector Problem
: Given a basis for L, find the shortest vector in V that is also a point in L.

Closest Vector Problem
: Given a basis for L and a point v in V, find closest lattice point to v in L.



Setbacks

Breaking Rainbow Takes a Weekend on a Laptop

Ward Beullens

IBM Research, Zurich, Switzerland
wbe@zurich.ibm.com

Abstract. This work introduces new key recovery attacks against the Rainbow signature scheme, which is one of the three finalist signature schemes still in the NIST Post-Quantum Cryptography standardization project. The new attacks outperform previously known attacks for all the parameter sets submitted to NIST and make a key-recovery practical for the SL 1 parameters. Concretely, given a Rainbow public key for the SL 1 parameters of the second-round submission, our attack returns the corresponding secret key after on average 53 hours (one weekend) of computation time on a standard laptop.

AN EFFICIENT KEY RECOVERY ATTACK ON SIDH (PRELIMINARY VERSION)

WOUTER CASTRYCK AND THOMAS DECRU

imec-COSIC, KU Leuven

Abstract. We present an efficient key recovery attack on the Supersingular Isogeny Diffie–Hellman protocol (SIDH), based on a "glue-and-split" theorem due to Kani. Our attack exploits the existence of a small non-scalar endomorphism on the starting curve, and it also relies on the auxiliary torsion point information that Alice and Bob share during the protocol. Our Magma implementation breaks the instantiation `SIKEp434`, which aims at security level 1 of the Post-Quantum Cryptography standardization process currently ran by NIST, in about one hour on a single core. This is a preliminary version of a longer article in preparation.

#eprint555

Quantum Algorithms for Lattice Problems

Yilei Chen*

April 10, 2024

Abstract

We show a polynomial time quantum algorithm for solving the learning with errors problem (LWE) with certain polynomial modulus-noise ratios. Combining with the reductions from lattice problems to LWE shown by Regev [J.ACM 2009], we obtain polynomial time quantum algorithms for solving the decisional shortest vector problem (GapSVP) and the shortest independent vector problem (SIVP) for all n-dimensional lattices within approximation factors of $\tilde{\Omega}(n^{4.5})$. Previously, no polynomial or even subexponential time quantum algorithms were known for solving GapSVP or SIVP for all lattices within any polynomial approximation factors.

To develop a quantum algorithm for solving LWE, we mainly introduce two new techniques. First, we introduce Gaussian functions with *complex* variances in the design of quantum algorithms. In particular, we exploit the feature of the *Karst wave* in the discrete Fourier transform of complex Gaussian functions. Second, we use *windowed* quantum Fourier transform with complex Gaussian windows, which allows us to combine the information from both time and frequency domains. Using those techniques, we first convert the LWE instance into quantum states with purely imaginary Gaussian amplitudes, then convert purely imaginary Gaussian states into classical linear equations over the LWE secret and error terms, and finally solve the linear system of equations using Gaussian elimination. This gives a polynomial time quantum algorithm for solving LWE.

Nine simple steps

1. Prepare a uniform superposition over $L \cap \mathbb{Z}_{Dq}^n$, and then apply a complex Gaussian window on it. We obtain a classical string $\mathbf{y}' \in \mathbb{Z}_{Dq}^n$ and a quantum state $|\varphi_1\rangle$:

$$|\varphi_1\rangle = \sum_{k \in \mathbb{Z}, k\mathbf{x}-\mathbf{y} \in (r \log n)\mathcal{B}_\infty^n} \exp\left(-\pi\left(\frac{1}{r^2}+\frac{i}{s^2}\right)\|k\mathbf{x}-\mathbf{y}\|^2\right)|k\mathbf{x}-\mathbf{y}\rangle, \tag{13}$$

   where $\mathbf{y} \in \mathbb{Z}^n$ is an unknown vector at this moment but its information is carried in $\mathbf{y}'$.

2. Compute $|\varphi_2\rangle = \mathsf{QFT}_{\mathbb{Z}_P^n}|\varphi_1\rangle$.
3. Apply a complex Gaussian window on $|\varphi_2\rangle$, get $|\varphi_3\rangle$, $\mathbf{z}' \in \mathbb{Z}_P^n$.
4. Compute $|\varphi_4\rangle = \mathsf{QFT}_{\mathbb{Z}_P^n}|\varphi_3\rangle$.
5. Split $|\varphi_4\rangle$ into higher and lower order bits, then measure the lower order bits in $\mathbb{Z}_{t^2+u^2}^n$ and get $\mathbf{h}^* \in \mathbb{Z}_{t^2+u^2}^n$. Denote the residual state (containing the higher order bits in $\mathbb{Z}_M^n$) as $|\varphi_5\rangle$.
6. Compute $|\varphi_6\rangle = \mathsf{QFT}_{\mathbb{Z}_M^n}|\varphi_5\rangle$. (The Karst wave feature is heavily used in the analysis of Step 6.)
7. Extract the centers of the Gaussian ball states in $|\varphi_6\rangle$ using $\mathbf{y}'$, $\mathbf{z}'$, and $\mathbf{h}^*$, get

$$|\varphi_7\rangle = \sum_{\mathbf{k} \in 0|\mathbb{Z}^{n-1}, j \in \mathbb{Z}} e^{-2\pi i \frac{(2Dj)^2}{2M}} e^{2\pi i \frac{\|\mathbf{k}\|^2}{4}} \left|2Dj\mathbf{x} + \mathbf{v}' + \frac{M}{2}\mathbf{k} \bmod M\right\rangle, \tag{14}$$

   where $\mathbf{v}'$ is a vector in L fixed by the previous measurements but unknown at this point.

8. Apply a sequence of small operations to extract $v_1' \bmod D^2 p_1$, without collapsing the state, and get $|\varphi_8\rangle = |\varphi_7\rangle$.
9. From $|\varphi_8\rangle$, use the $p_2, \ldots, p_\kappa$ values planted in the secret vector in the instance of $\mathsf{LWE}^{k \text{ chosen secret}}$, $v_1' \bmod D^2 p_1$ obtained in Step 8, and apply a few operations on $|\varphi_8\rangle$ to get a random vector $\mathbf{u} \in \mathbb{Z}_{\frac{M}{2}}^n$ satisfying

$$u_1 + \left\langle \mathbf{b}_{[2\ldots n]}^*, \mathbf{u}_{[2\ldots n]} \right\rangle \equiv 0 \pmod{\frac{M}{2D^2}}, \tag{15}$$

   where in $\mathbf{b}_{[2\ldots n]}^* = \mathbf{b}_{[2\ldots\kappa]}^* | \mathbf{b}_{[\kappa+1\ldots n]}^*$, $\mathbf{b}_{[2\ldots\kappa]}^*$ is known and fixed, $\mathbf{b}_{[\kappa+1\ldots n]}^* = \mathbf{b}_{[\kappa+1\ldots n]}$, which is exactly the secret term we want to learn.

Note: Update on April 18: Step 9 of the algorithm contains a bug, which I don't know how to fix. See Section 3.5.9 (Page 37) for details. I sincerely thank Hongxun Wu and (independently) Thomas Vidick for finding the bug today. Now the claim of showing a polynomial time quantum algorithm for solving LWE with polynomial modulus-noise ratios does not hold. I leave the rest of the paper as it is (added a clarification of an operation in Step 8) as a hope that ideas like Complex Gaussian and windowed QFT may find other applications in quantum computation, or tackle LWE in other ways.

runs through all $j \in \mathbb{Z}_{p_1 p_2 \ldots p_\kappa}$, but currently the j in the first coordinate only runs through $\mathbb{Z}_{p_1}$. So we apply the domain extension trick (Lemma 2.17) on the first coordinate of $|\varphi_{8.f}\rangle$ to extend the domain of the first coordinate from $D^2 p_1 p_2 \ldots p_\kappa$ to $D^2 p_1 p_2 \ldots p_\kappa \cdot p_2 \ldots p_\kappa$, and get

$$|\varphi_{8.g}\rangle := \sum_{j \in \mathbb{Z}} e^{-2\pi i \frac{(2Dj)^2}{2M}} \left|2D^2 j \mathbf{b}_1^* \bmod D^2 p_1 p_2 \ldots p_\kappa \cdot p_2 \ldots p_\kappa\right\rangle \left|2D^2 j \mathbf{b}_{[2\ldots n]}^* + \mathbf{v}_{[2\ldots n]}^* \bmod D^2 p_1 p_2 \ldots p_\kappa\right\rangle .$$

Yilei (April 18) Here is the bug: the amplitude of $|\varphi_{8.f}\rangle$ does not satisfy $\frac{M}{2}$-periodicity. Another way of explaining the bug is: the support of $|\varphi_{8.f}\rangle$ contains $p_1 \ldots p_\kappa$ vectors. After domain extension, we should have got $p_1 p_2 \ldots p_\kappa \cdot p_2 \ldots p_\kappa$ vectors, but as the way $|\varphi_{8.g}\rangle$ is written, it only contains $p_1 \ldots p_\kappa$ vectors. So the expression of $|\varphi_{8.g}\rangle$ is wrong.

Thomas Vidick Today at 3:17 AM

Unfortunately, the bug by itself does not seem to teach us more about the overall viability of Chen's approach. I think that there is much more to do to understand what parts may still be valid, and if some of the ideas can be extended, either back to a quantum algorithm for lattice problems, or possibly another application in quantum cryptography.

Chris Peikert
@ChrisPeikert

Any serious attempt to attack lattices/LWE that doesn't change the status quo should increase our confidence in their security.

> **Boaz Barak** @boazbaraktcs · Apr 19
>
> Chen's paper has a bug, independently discovered by Hongxun Weng and Thomas Vidick, that he doesn't know how to fix. If I understand correctly, in its current form the paper doesn't yield any improvement on prior algorithms.
>
> eprint.iacr.org/2024/555

1:14 PM · Apr 19, 2024 · **20.1K** Views

2 37 161 9

Post your reply

Reply

Martin R. Albrecht @martinralbrecht · Apr 19

Do you have a sense of how fundamental the bug is? Yilei doesn't know how to fix the algorithm but it's not clear to me, this invalidates the approach? I'm not gunning for some abstract "maybe everything is broken" but for whether we can indeed have more confidence from this?

1 6 1.2K

Martin R. Albrecht @martinralbrecht · Apr 19

Do you have a sense of how fundamental the bug is? Yilei doesn't know how to fix the algorithm but it's not clear to me, this invalidates the approach? I'm not gunning for some abstract "maybe everything is broken" but for whether we can indeed have more confidence from this?

1 · 6 · 1.2K

Jacob Alperin-Sheriff 🐵🍌🖍️
@DemocraticLuntz

The bug is at the end, but seems quasi-fundamentally related to the initial idea (over a standard uSVP instance) of the product of primes use that seems highly fundamental to the approach, so in conclusion, like with all things in this context, "Who knows"

6:11 PM · Apr 19, 2024 · **334** Views

1

Post your reply

Reply

Jacob Alperin-Sheriff 🐵🍌🖍️ @DemocraticLuntz · Apr 19

Also there is NO guarantee that this is the only bug, there's no way anyone really subjected themselves to trying to verify Step 6 yet, for instance.

131

Summary of #555

- How near is the attack to work? Nobody knows.
- Open research works.

Hybrid cryptography

Dilemma: What do we fear the most?

1. A cryptographically relevant quantum computer, it may only be a few decades away
2. These new algorithms have fundamental flaws, just waiting to be found

Two answers

NSA	“The schemes are fine, go fully quantum-safe.”
NOR, UK, GER, FRA, ...	Get k_1 from PQC, k_2 from ECDH, $k \leftarrow \mathrm{KDF}(k_1, k_2)$

Some benchmarking

For discussion: How will lattice crypto compare to elliptic curve crypto?

Some benchmarks

	Public key	Private key	Ciphertext
ECDH	97 B	48 B	
Kyber	1568 B	3168 B	1568 B

	Verification key	Signing key	Signature
ECDSA	48 B	48 B	96 B
Dilithium	2592 B	4864 B	4595 B

Key exchange	Signature	Time
Kyber	Dilithium	69.6 ms
Kyber	FALCON	44.5 ms
Kyber	SPHINCS+	911.0 ms
ECDHE	ECDSA	

(Timings from Table 2 in https://eprint.iacr.org/2023/506)

Some benchmarks

	Public key	Private key	Ciphertext
ECDH	97 B	48 B	
Kyber	1568 B	3168 B	1568 B

	Verification key	Signing key	Signature
ECDSA	48 B	48 B	96 B
Dilithium	2592 B	4864 B	4595 B

Key exchange	Signature	Time
Kyber	Dilithium	69.6 ms
Kyber	FALCON	44.5 ms
Kyber	SPHINCS+	911.0 ms
ECDHE	ECDSA	102.1 ms

(Timings from Table 2 in https://eprint.iacr.org/2023/506)

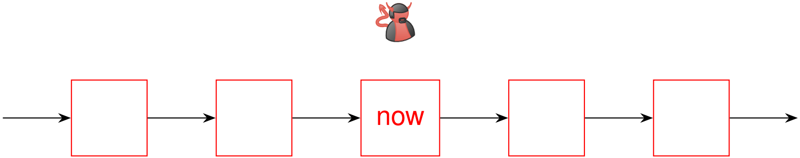
Our primitive toolbox, pt. III

	Symmetric	**Asymmetric**
Confidentiality	AES, ChaCha20, ...	Kyber + Diffie–Hellman
Integrity	HMAC, KMAC, AES-GCD, ...	{Dilithium, SPHINCS+, Falcon} + ECDSA

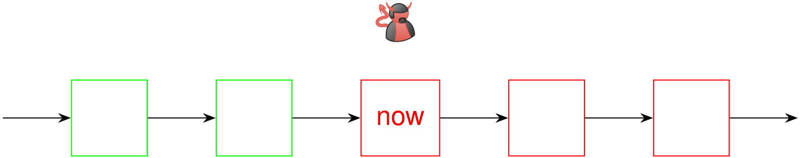
Other security properties and protocols

Security properties

Security properties

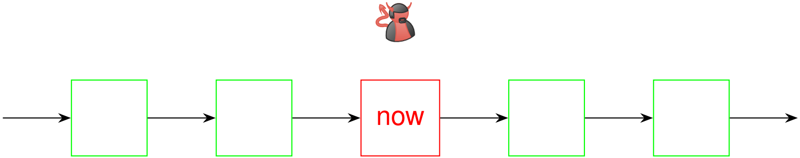


Security properties



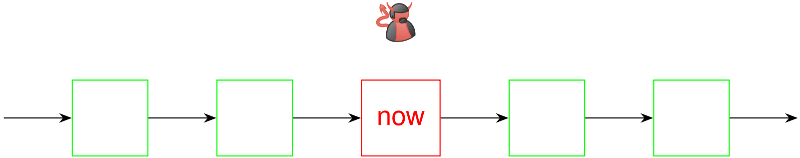
Forward secrecy Compromise today should not affect the past

Security properties



Forward secrecy	Compromise today should not affect the past
Post-compromise security	We should be able to return to a secure state after a full compromise today

Security properties



Forward secrecy	Compromise today should not affect the past
Post-compromise security	We should be able to return to a secure state after a full compromise today
Length of “today”	?

Signal

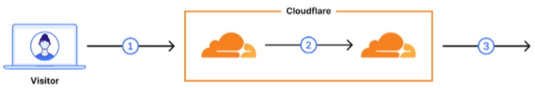
Forward secrecy	Compromise today should not affect the past
Post-compromise security	We should be able to return to a secure state after a full compromise today
Length of “today”	One message

Signal

1. Introduction

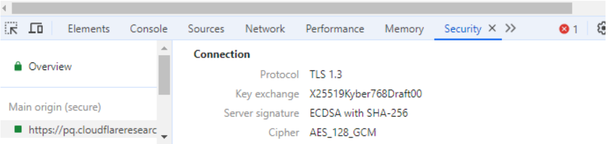
This document describes the "PQXDH" (or "Post-Quantum Extended Diffie-Hellman") key agreement protocol. PQXDH establishes a shared secret key between two parties who mutually authenticate each other based on public keys. PQXDH provides post-quantum forward secrecy and a form of cryptographic deniability but still relies on the hardness of the discrete log problem for mutual authentication in this revision of the protocol.

Cloudflare Research: Post-Quantum Key Agreement



On essentially all domains served (1) through Cloudflare, including this one, we have enabled hybrid post-quantum key agreement. We are also rolling out support for post-quantum key agreement for connection from Cloudflare to origins (3). Check out our blog post the state of the post-quantum Internet for more context.

You are using *X25519Kyber768Draft00* which is **post-quantum secure**.



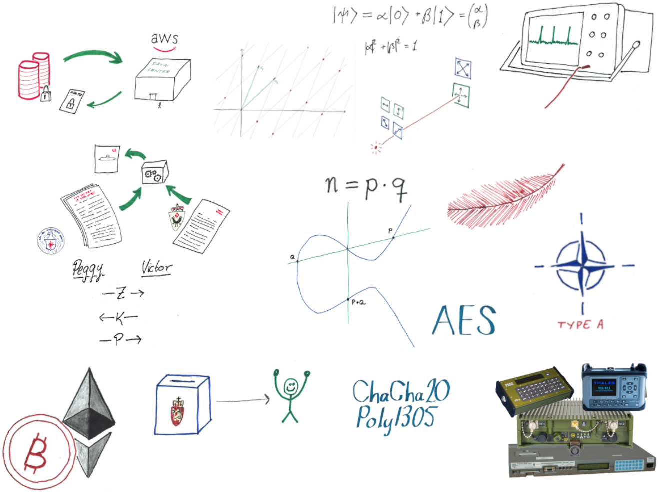
What about privacy?

Consider yourself the adversary

Summary

- Quantum-safe crypto is coming
- Symmetric crypto is already fine
- The algorithms are efficient, but the keys are large
- The protocols will adopt the algorithms
- Fancy crypto still needs loads of work
- Security is not the same as privacy

We made it through!



Bonus content

The Number Theoretic Transform (NTT)

Fact

$$R_q = \mathbb{Z}[X]/(X^{256}+1) \simeq \bigoplus_{k=0}^{127} \mathbb{Z}_q[X]/\left(X^2 - \zeta^{2\mathrm{BitReverse}_7(i)+1}\right) = T_q$$

Let $f \in R_q$. Then $\mathrm{NTT} : R_q \to T_q$ is given by

$$\mathrm{NTT}(f) = \left(f \bmod \left(X^2 - \zeta^{2\mathrm{BitReverse}_7(0)+1}\right), \dots, f \bmod \left(X^2 - \zeta^{2\mathrm{BitReverse}_7(127)+1}\right)\right)$$

and NTT^{-1} is also efficient.

Kyber in the NTT realm

Multiplication in R_q:
256×256 multiplications

Multiplication in T_q:
128×4 multiplications

Sampling algorithms

SampleNTT	Convert a stream of bytes into a polynomial in the NTT domain
SamplePolyCBD$_\eta$	Sample a coefficient array of a polynomial $f \in R_q$, according to a centered binomial distribution specified by η.

Compression using seeds

$\rho \leftarrow \mathsf{ek}_{\mathrm{PKE}}[384k : 384k + 32]$ ▷ extract 32-byte seed from $\mathsf{ek}_{\mathrm{PKE}}$
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ re-generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$
    for $(j \leftarrow 0;\ j < k;\ j{+}{+})$
        $\hat{\mathbf{A}}[i, j] \leftarrow \mathsf{SampleNTT}(\mathsf{XOF}(\rho, i, j))$
    end for
end for

Computer-friendly representation

Algorithm 4 ByteEncode$_d(F)$

Encodes an array of d-bit integers into a byte array, for $1 \le d \le 12$.

Input: integer array $F \in \mathbb{Z}_m^{256}$, where $m = 2^d$ if $d < 12$ and $m = q$ if $d = 12$.
Output: byte array $B \in \mathbb{B}^{32d}$.

for $(i \leftarrow 0;\ i < 256;\ i{+}{+})$
  $a \leftarrow F[i]$ ▷ $a \in \mathbb{Z}_{2^d}$
  for $(j \leftarrow 0;\ j < d;\ j{+}{+})$
    $b[i \cdot d + j] \leftarrow a \bmod 2$ ▷ $b \in \{0,1\}^{256 \cdot d}$
    $a \leftarrow (a - b[i \cdot d + j])/2$ ▷ note $a - b[i \cdot d + j]$ is always even.
  end for
end for
$B \leftarrow$ BitsToBytes(b)
return B

Algorithm 5 ByteDecode$_d(B)$

Decodes a byte array into an array of d-bit integers, for $1 \le d \le 12$.

Input: byte array $B \in \mathbb{B}^{32d}$.
Output: integer array $F \in \mathbb{Z}_m^{256}$, where $m = 2^d$ if $d < 12$ and $m = q$ if $d = 12$.

$b \leftarrow$ BytesToBits(B)
for $(i \leftarrow 0;\ i < 256;\ i{+}{+})$
  $F[i] \leftarrow \sum_{j=0}^{d-1} b[i \cdot d + j] \cdot 2^j \bmod m$
end for
return F

Compression and decompression of numbers

$$\begin{aligned}\text{Compress}_d : \mathbb{Z}_q &\to \mathbb{Z}_{2^d}\\ x &\mapsto \left\lfloor \left(2^d/q\right) \cdot x \right\rceil\\ \text{Decompress}_d : \mathbb{Z}_{2^d} &\to \mathbb{Z}_q\\ y &\mapsto \left\lfloor \left(q/2^d\right) \cdot y \right\rceil\end{aligned}$$

Compression and decompression of numbers

$$\begin{aligned}\text{Compress}_d : \mathbb{Z}_q &\to \mathbb{Z}_{2^d}\\ x &\mapsto \left\lfloor (2^d/q) \cdot x \right\rceil\\ \text{Decompress}_d : \mathbb{Z}_{2^d} &\to \mathbb{Z}_q\\ y &\mapsto \left\lfloor (q/2^d) \cdot y \right\rceil\end{aligned}$$

$\text{Decompress}_d \circ \text{Compress}_d \approx 1$

$$[\text{Decompress}_d(\text{Compress}_d(x)) - x] \bmod {}^{\pm}q \leq \left\lfloor q/2^{d+1} \right\rceil$$

The finished K-PKE algorithm

Algorithm 12 K-PKE.KeyGen()

Generates an encryption key and a corresponding decryption key.

Output: encryption key $\mathsf{ek}_{\mathrm{PKE}} \in \mathbb{B}^{384k+32}$.
Output: decryption key $\mathsf{dk}_{\mathrm{PKE}} \in \mathbb{B}^{384k}$.

$d \xleftarrow{\$} \mathbb{B}^{32}$ ▷ d is 32 random bytes (see Section 3.3)
$(\rho, \sigma) \leftarrow G(d)$ ▷ expand to two pseudorandom 32-byte seeds
$N \leftarrow 0$
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$
  for $(j \leftarrow 0;\ j < k;\ j{+}{+})$
    $\hat{\mathbf{A}}[i,j] \leftarrow \mathsf{SampleNTT}(\mathsf{XOF}(\rho, i, j))$ ▷ each entry of $\hat{\mathbf{A}}$ uniform in NTT domain
  end for
end for
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ generate $\mathbf{s} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{s}[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_1}(\mathsf{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{s}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ generate $\mathbf{e} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{e}[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_1}(\mathsf{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{e}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
$\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$ ▷ NTT is run k times (once for each coordinate of $\mathbf{s}$)
$\hat{\mathbf{e}} \leftarrow \mathsf{NTT}(\mathbf{e})$ ▷ NTT is run k times
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$ ▷ noisy linear system in NTT domain
$\mathsf{ek}_{\mathrm{PKE}} \leftarrow \mathsf{ByteEncode}_{12}(\hat{\mathbf{t}}) \| \rho$ ▷ $\mathsf{ByteEncode}_{12}$ is run k times; include seed for $\hat{\mathbf{A}}$
$\mathsf{dk}_{\mathrm{PKE}} \leftarrow \mathsf{ByteEncode}_{12}(\hat{\mathbf{s}})$ ▷ $\mathsf{ByteEncode}_{12}$ is run k times
return $(\mathsf{ek}_{\mathrm{PKE}}, \mathsf{dk}_{\mathrm{PKE}})$

The finished K-PKE algorithm

Algorithm 13 K-PKE.Encrypt$(\mathsf{ek}_{\mathrm{PKE}}, m, r)$

Uses the encryption key to encrypt a plaintext message using the randomness r.

Input: encryption key $\mathsf{ek}_{\mathrm{PKE}} \in \mathbb{B}^{384k+32}$.
Input: message $m \in \mathbb{B}^{32}$.
Input: encryption randomness $r \in \mathbb{B}^{32}$.
Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

$N \leftarrow 0$
$\hat{\mathbf{t}} \leftarrow \mathsf{ByteDecode}_{12}(\mathsf{ek}_{\mathrm{PKE}}[0:384k])$
$\rho \leftarrow \mathsf{ek}_{\mathrm{PKE}}[384k : 384k+32]$ ▷ extract 32-byte seed from $\mathsf{ek}_{\mathrm{PKE}}$
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ re-generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$
    for $(j \leftarrow 0;\ j < k;\ j{+}{+})$
        $\hat{\mathbf{A}}[i,j] \leftarrow \mathsf{SampleNTT}(\mathsf{XOF}(\rho, i, j))$
    end for
end for
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ generate $\mathbf{r} \in (\mathbb{Z}_q^{256})^k$
    $\mathbf{r}[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_1}(\mathsf{PRF}_{\eta_1}(r, N))$ ▷ $\mathbf{r}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
    $N \leftarrow N + 1$
end for
for $(i \leftarrow 0;\ i < k;\ i{+}{+})$ ▷ generate $\mathbf{e_1} \in (\mathbb{Z}_q^{256})^k$
    $\mathbf{e}_1[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$ ▷ $\mathbf{e}_1[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
    $N \leftarrow N + 1$
end for
$e_2 \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$ ▷ sample $e_2 \in \mathbb{Z}_q^{256}$ from CBD
$\hat{\mathbf{r}} \leftarrow \mathsf{NTT}(\mathbf{r})$ ▷ NTT is run k times
$\mathbf{u} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^{\intercal} \circ \hat{\mathbf{r}}) + \mathbf{e_1}$ ▷ NTT^{-1} is run k times
$\mu \leftarrow \mathsf{Decompress}_1(\mathsf{ByteDecode}_1(m))$
$v \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^{\intercal} \circ \hat{\mathbf{r}}) + e_2 + \mu$ ▷ encode plaintext m into polynomial v.
$c_1 \leftarrow \mathsf{ByteEncode}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u}))$ ▷ $\mathsf{ByteEncode}_{d_u}$ is run k times
$c_2 \leftarrow \mathsf{ByteEncode}_{d_v}(\mathsf{Compress}_{d_v}(v))$
return $c \leftarrow (c_1 \| c_2)$

The finished K-PKE algorithm

Algorithm 14 K-PKE.Decrypt($\mathsf{dk}_{\mathsf{PKE}}, c$)

Uses the decryption key to decrypt a ciphertext.

Input: decryption key $\mathsf{dk}_{\mathsf{PKE}} \in \mathbb{B}^{384k}$.
Input: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.
Output: message $m \in \mathbb{B}^{32}$.

```
c_1 ← c[0 : 32 d_u k]
c_2 ← c[32 d_u k : 32(d_u k + d_v)]
u ← Decompress_{d_u}(ByteDecode_{d_u}(c_1))      ▷ ByteDecode_{d_u} invoked k times
v ← Decompress_{d_v}(ByteDecode_{d_v}(c_2))
ŝ ← ByteDecode_12(dk_PKE)
w ← v − NTT^{-1}(ŝ^T ∘ NTT(u))                   ▷ NTT^{-1} and NTT invoked k times
m ← ByteEncode_1(Compress_1(w))                 ▷ decode plaintext m from polynomial v
return m
```

Security amplification: The Fujisaki-Okamoto transformation

Theorem (Fujisaki-Okamoto (informal))

If $\mathcal{E}$ is an IND-CPA secure public-key cryptosystem, then $\mathsf{FO}(\mathcal{E})$ *is an IND-CCA secure key encapsulation mechanism.*

Security amplification: The Fujisaki-Okamoto transformation

Theorem (Fujisaki-Okamoto (informal))

If $\mathcal{E}$ is an IND-CPA secure public-key cryptosystem, then $\mathsf{FO}(\mathcal{E})$ *is an IND-CCA secure key encapsulation mechanism.*

Theorem 3.5 (PKE_1 **det.,** $\mathsf{OW\text{-}VA} \overset{\text{ROM}}{\Rightarrow} \mathsf{KEM}_m^{\perp}$ $\mathsf{IND\text{-}CCA}$**).** *If* PKE_1 *is* δ_1*-correct, then so is* $\mathsf{KEM}_m^{\perp}$*. Furthermore, assume* PKE_1 *to be rigid. Let* G *denote the random oracle that* PKE_1 *uses (if any), and let* $q_{\mathsf{Enc}_1,\mathsf{G}}$ *and* $q_{\mathsf{Dec}_1,\mathsf{G}}$ *denote an upper bound on the number of* G*-queries that* Enc_1*, resp.* Dec_1 *makes upon a single invocation. If* Enc_1 *is deterministic then, for any* $\mathsf{IND\text{-}CCA}$ *adversary* B *against* $\mathsf{KEM}_m^{\perp}$*, issuing at most* q_D *queries to the decapsulation oracle* $\textsc{Decaps}_m^{\perp}$ *and at most* q_{G}*, resp.* q_{H} *queries to its random oracles* G *and* H*, there exists an* $\mathsf{OW\text{-}VA}$ *adversary* A *against* PKE_1 *that makes at most* q_D *queries to the* $\textsc{Cvo}$ *oracle such that*

$$\mathrm{Adv}_{\mathsf{KEM}_m^{\perp}}^{\mathsf{IND\text{-}CCA}}(\mathsf{B}) \leq \mathrm{Adv}_{\mathsf{PKE}_1}^{\mathsf{OW\text{-}VA}}(\mathsf{A}) + \delta_1(q_{\mathsf{G}} + (q_{\mathsf{H}} + q_D)(q_{\mathsf{Enc}_1,\mathsf{G}} + q_{\mathsf{Dec}_1,\mathsf{G}}))$$

and the running time of A *is about that of* B*.*

(Hofheinz, Hövelmanns, Kiltz: “A Modular Analysis of the Fujisaki-Okamoto Transformation” (2017))

ML-KEM

Algorithm 15 ML-KEM.KeyGen()

Generates an encapsulation key and a corresponding decapsulation key.

Output: Encapsulation key $\mathsf{ek} \in \mathbb{B}^{384k+32}$.
Output: Decapsulation key $\mathsf{dk} \in \mathbb{B}^{768k+96}$.

$z \xleftarrow{\$} \mathbb{B}^{32}$ ▷ z is 32 random bytes (see Section 3.3)
$(\mathsf{ek}_{\text{PKE}}, \mathsf{dk}_{\text{PKE}}) \leftarrow$ K-PKE.KeyGen() ▷ run key generation for K-PKE
$\mathsf{ek} \leftarrow \mathsf{ek}_{\text{PKE}}$ ▷ KEM encaps key is just the PKE encryption key
$\mathsf{dk} \leftarrow (\mathsf{dk}_{\text{PKE}} \| \mathsf{ek} \| H(\mathsf{ek}) \| z)$ ▷ KEM decaps key includes PKE decryption key
return $(\mathsf{ek}, \mathsf{dk})$

ML-KEM

Algorithm 16 ML-KEM.Encaps(ek)

Uses the encapsulation key to generate a shared key and an associated ciphertext.

Validated input: encapsulation key $\mathsf{ek} \in \mathbb{B}^{384k+32}$.
Output: shared key $K \in \mathbb{B}^{32}$.
Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

1: $m \xleftarrow{\$} \mathbb{B}^{32}$ ▷ m is 32 random bytes (see Section 3.3)
2: $(K, r) \leftarrow G(m \| H(\mathsf{ek}))$ ▷ derive shared secret key K and randomness r
3: $c \leftarrow$ K-PKE.Encrypt(ek, m, r) ▷ encrypt m using K-PKE with randomness r
4: **return** (K, c)

ML-KEM

Algorithm 17 ML-KEM.Decaps(c, dk)

Uses the decapsulation key to produce a shared key from a ciphertext.

Validated input: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.
Validated input: decapsulation key $\mathsf{dk} \in \mathbb{B}^{768k+96}$.
Output: shared key $K \in \mathbb{B}^{32}$.

$\mathsf{dk}_{\mathrm{PKE}} \leftarrow \mathsf{dk}[0 : 384k]$ ▷ extract (from KEM decaps key) the PKE decryption key
$\mathsf{ek}_{\mathrm{PKE}} \leftarrow \mathsf{dk}[384k : 768k + 32]$ ▷ extract PKE encryption key
$h \leftarrow \mathsf{dk}[768k + 32 : 768k + 64]$ ▷ extract hash of PKE encryption key
$z \leftarrow \mathsf{dk}[768k + 64 : 768k + 96]$ ▷ extract implicit rejection value
$m' \leftarrow$ K-PKE.Decrypt$(\mathsf{dk}_{\mathrm{PKE}}, c)$ ▷ decrypt ciphertext
$(K', r') \leftarrow G(m' \| h)$
$\bar{K} \leftarrow J(z \| c, 32)$
$c' \leftarrow$ K-PKE.Encrypt$(\mathsf{ek}_{\mathrm{PKE}}, m', r')$ ▷ re-encrypt using the derived randomness r'
if $c \neq c'$ **then**
    $K' \leftarrow \bar{K}$ ▷ if ciphertexts do not match, "implicitly reject"
end if
return K'

Parameter sets and key sizes

	n	q	k	η_1	η_2	d_u	d_v	required RBG strength (bits)
ML-KEM-512	256	3329	2	3	2	10	4	128
ML-KEM-768	256	3329	3	2	2	10	4	192
ML-KEM-1024	256	3329	4	2	2	11	5	256

Table 2. Approved parameter sets for ML-KEM

	encapsulation key	decapsulation key	ciphertext	shared secret key
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

Table 3. Sizes (in bytes) of keys and ciphertexts of ML-KEM

NIST security levels

NIST cat.	As strong as	Kyber
I	AES-128	ML-KEM-512
II	SHA-256	
III	AES-192	ML-KEM-768
IV	SHA-384	
V	AES-256	ML-KEM-1024