MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Kyber – Design

July 31, 2024

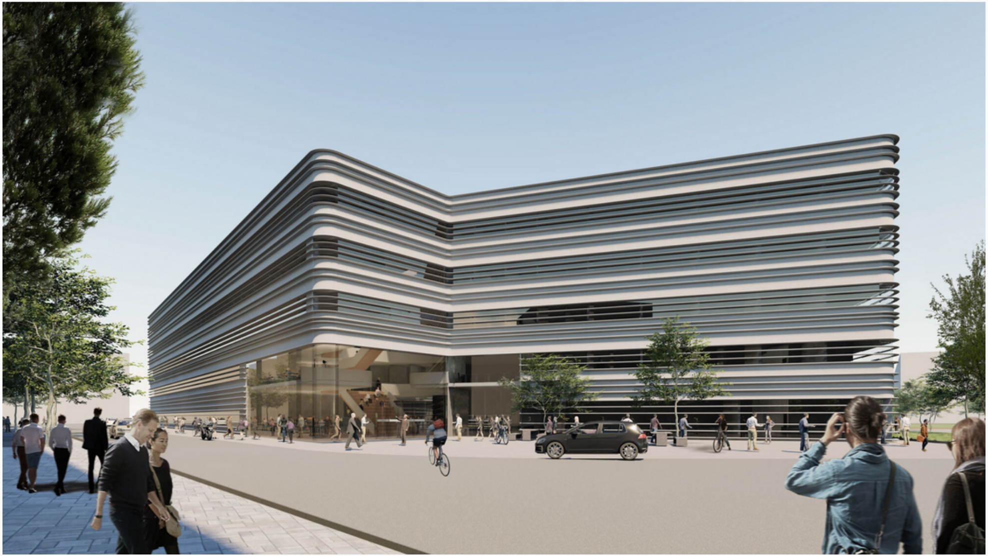
> *“It’s going to be exhausting. You will start feeling it around the middle of the week.”*
>
> *—Peter Druschel, July 29, 2024*

MPI-SP?

- Located in Bochum
- Founded in 2019
- Currently 11 PIs
- Aim to have
  - 6 Directors
  - 12 MPRGLs
  - Around 250 people total
- Currently on RUB campus

From `https://imprs-brain-behavior.mpg.de/Max_Planck_Society`

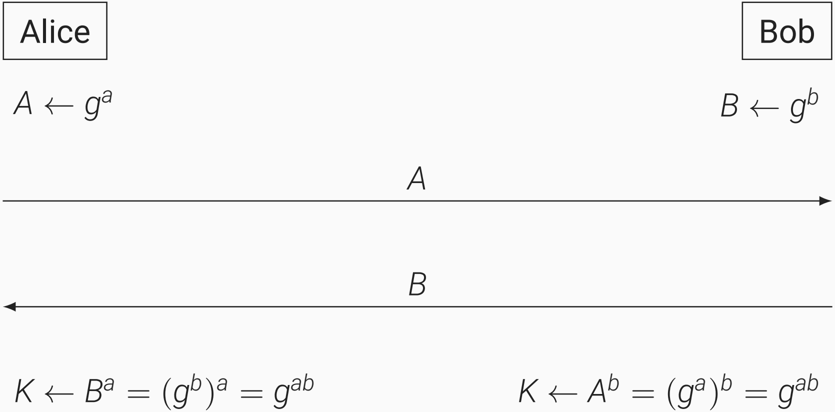
Our new home (move planned for 2027)



[A small demo]

ECDH and X25519

Let G be a finite cyclic group with generator g.



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

$$\xrightarrow{\quad A \quad}$$

$$\xleftarrow{\quad B \quad}$$

$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

ECDH and X25519

- Diffie, Hellman, 1976: Use $G = GF(q)^*$
- Miller, Koblitz (independently), 1985/86: Use group of points on an elliptic curve
- Bernstein, 2006: Use specific elliptic curve over $GF(2^{255} - 19)$

(EC)DH is everywhere

Nitrokey
secure your digital life

WIREGUARD
FAST, MODERN, SECURE VPN TUNNEL

Tor

zoom

The Discrete Logarithm Problem

Definition

Given $P, Q \in G$ such that $Q \in \langle P \rangle$, find an integer k such that $P^k = Q$.

The Discrete Logarithm Problem

Definition

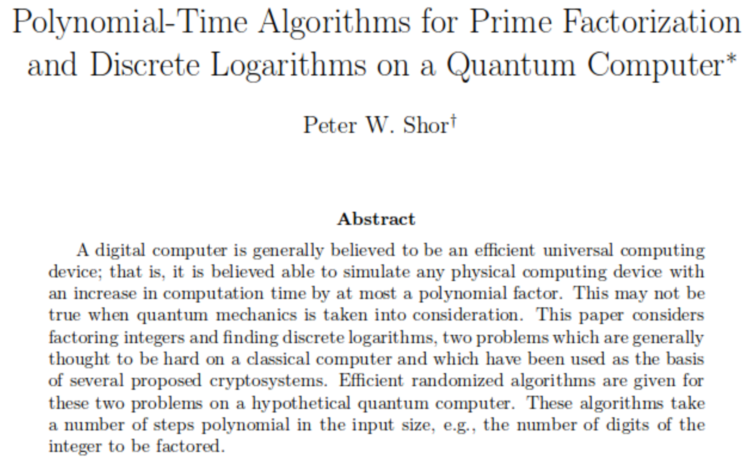
Given $P, Q \in G$ such that $Q \in \langle P \rangle$, find an integer k such that $kP = Q$.

The Discrete Logarithm Problem

Definition

Given $P, Q \in G$ such that $Q \in \langle P \rangle$, find an integer k such that $kP = Q$.

- DH needs group where DLP is hard
- (EC)DLP-based crypto also for signatures (DSA, ECDSA, EdDSA...)
- Prominent alternative: RSA (based on factoring)

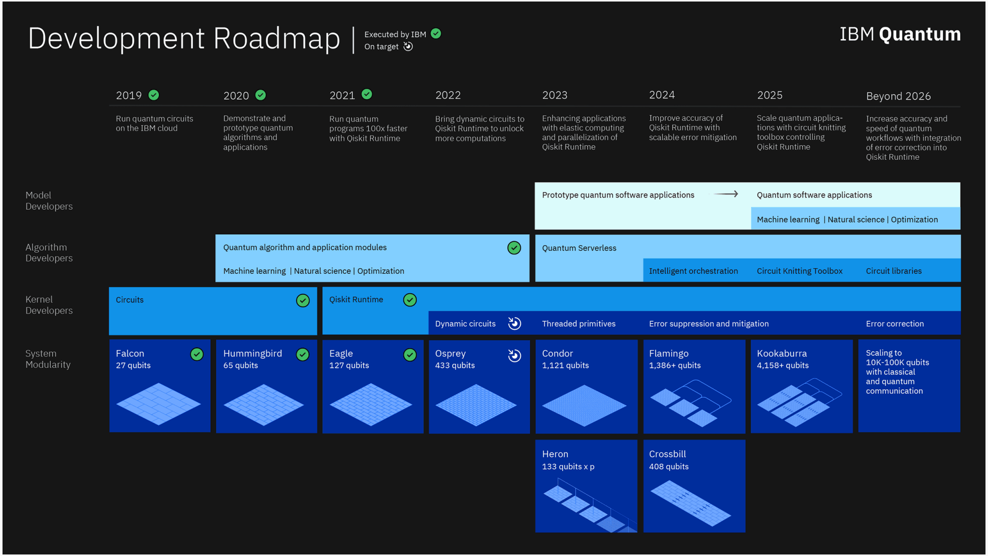


Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer[*]

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.



Development Roadmap | Executed by IBM ✓ On target ↻

IBM **Quantum**

2019 ✓	Run quantum circuits on the IBM cloud
2020 ✓	Demonstrate and prototype quantum algorithms and applications
2021 ✓	Run quantum programs 100x faster with Qiskit Runtime
2022	Bring dynamic circuits to Qiskit Runtime to unlock more computations
2023	Enhancing applications with elastic computing and parallelization of Qiskit Runtime
2024	Improve accuracy of Qiskit Runtime with scalable error mitigation
2025	Scale quantum applications with circuit knitting toolbox controlling Qiskit Runtime
Beyond 2026	Increase accuracy and speed of quantum workflows with integration of error correction into Qiskit Runtime

Model Developers
- Prototype quantum software applications → Quantum software applications
- Machine learning | Natural science | Optimization

Algorithm Developers
- Quantum algorithm and application modules ✓
- Machine learning | Natural science | Optimization
- Quantum Serverless
- Intelligent orchestration
- Circuit Knitting Toolbox
- Circuit libraries

Kernel Developers
- Circuits ✓
- Qiskit Runtime ✓
- Dynamic circuits ↻
- Threaded primitives
- Error suppression and mitigation
- Error correction

System Modularity
- Falcon 27 qubits ✓
- Hummingbird 65 qubits ✓
- Eagle 127 qubits ✓
- Osprey 433 qubits ↻
- Condor 1,121 qubits
- Flamingo 1,386+ qubits
- Kookaburra 4,158+ qubits
- Scaling to 10K-100K qubits with classical and quantum communication
- Heron 133 qubits x p
- Crossbill 408 qubits

See `https://www.ibm.com/quantum/blog/ibm-quantum-roadmap-2025`

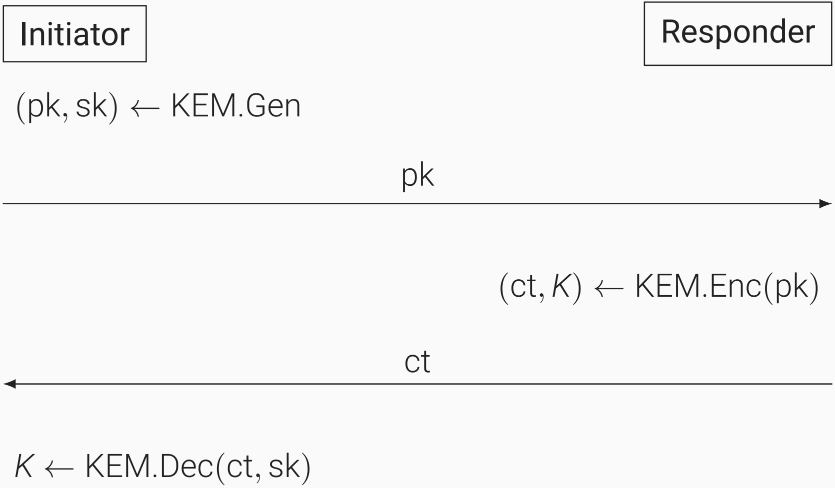
[Back to our demo]

POST-QUANTUM KEY EXCHANGE

A NEW HOPE

ERDEM ALKIM
LÉO DUCAS
THOMAS PÖPPELMANN
PETER SCHWABE

Key Encapsulation Mechanisms (KEMs)



Initiator

Responder

$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KEM.Gen}$

$\xrightarrow{\mathrm{pk}}$

$(\mathrm{ct}, K) \leftarrow \mathrm{KEM.Enc}(\mathrm{pk})$

$\xleftarrow{\mathrm{ct}}$

$K \leftarrow \mathrm{KEM.Dec}(\mathrm{ct}, \mathrm{sk})$

Learning with errors (LWE)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given “noise distribution” χ
- Given samples $\mathbf{A}\mathbf{s} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$

Learning with errors (LWE)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given “noise distribution” χ
- Given samples $\mathbf{A}\mathbf{s} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$
- Search version: find $\mathbf{s}$
- Decision version: distinguish from uniform random

Ring Learning with errors (RLWE)

- Given uniform $\mathbf{a} \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Given “noise distribution” χ
- Given samples $\mathbf{as} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$

Ring Learning with errors (RLWE)

- Given uniform $\mathbf{a} \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Given “noise distribution” χ
- Given samples $\mathbf{as} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$
- Search version: find $\mathbf{s}$
- Decision version: distinguish from uniform random

How to build a KEM?

Alice (server)		Bob (client)
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{\mathbf{b}}$	$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
	$\xleftarrow{\mathbf{u}}$	

$$\begin{array}{llll} \text{Alice has} & \mathbf{v} & = \mathbf{u}\mathbf{s} & = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s} \\ \text{Bob has} & \mathbf{v}' & = \mathbf{b}\mathbf{s}' & = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}' \end{array}$$

- Secret and noise polynomials $\mathbf{s}, \mathbf{s}', \mathbf{e}, \mathbf{e}'$ are small
- $\mathbf{v}$ and $\mathbf{v}'$ are *approximately* the same

How to build a KEM, part 2

Alice		Bob
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \quad \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b} \qquad)}$	
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}'$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u} \quad)}$	

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, \mathit{seed})}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}\quad)}$	

How to build a KEM, part 2

Alice		Bob
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{XOF}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
		$k \xleftarrow{\$} \{0,1\}^{n}$
		$\mathbf{k} \leftarrow \mathsf{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b},\mathit{seed})}$	$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^{n}$
		$\mathbf{k} \leftarrow \mathrm{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u},\mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, \mathit{seed})}$	$\mathbf{a} \leftarrow \text{Parse}(\text{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^{n}$
		$\mathbf{k} \leftarrow \text{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, \mathit{seed})}$	$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \xleftarrow{\$} \{0,1\}^n$
		$\mathbf{k} \leftarrow \mathrm{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \mathrm{Extract}(\mathbf{k})$
$\mu \leftarrow \mathrm{Extract}(\mathbf{k}')$		

How to build a KEM, part 2

Alice		Bob
$\mathit{seed} \overset{\$}{\leftarrow} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$		
$\mathbf{s}, \mathbf{e} \overset{\$}{\leftarrow} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \overset{\$}{\leftarrow} \chi$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, \mathit{seed})}$	$\mathbf{a} \leftarrow \mathrm{Parse}(\mathrm{XOF}(\mathit{seed}))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
		$k \overset{\$}{\leftarrow} \{0,1\}^{n}$
		$\mathbf{k} \leftarrow \mathrm{Encode}(k)$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \mathrm{Extract}(\mathbf{k})$
$\mu \leftarrow \mathrm{Extract}(\mathbf{k}')$		

Encryption scheme by Lyubashevsky, Peikert, Regev. Eurocrypt 2010.

Encode and Extract

- Encoding in LPR encryption: map *n* bits to *n* coefficients:
  - A zero bit maps to 0
  - A one bit maps to $q/2$
- Idea: Noise affects low bits of coefficients, put data into high bits

Encode and Extract

- Encoding in LPR encryption: map *n* bits to *n* coefficients:
  - A zero bit maps to 0
  - A one bit maps to $q/2$
- Idea: Noise affects low bits of coefficients, put data into high bits
- Decode: map coefficient into $[-q/2, q/2]$
  - Closer to 0 (i.e., in $[-q/4, q/4]$): set bit to zero
  - Closer to $\pm q/2$: set bit to one

NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
    - Originally proposed by Ding (2012)
    - Improvements by Peikert (2014)
    - More improvements in NEWHOPE

NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
  - Originally proposed by Ding (2012)
  - Improvements by Peikert (2014)
  - More improvements in NEWHOPE
- NEWHOPE-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)

NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
    - Originally proposed by Ding (2012)
    - Improvements by Peikert (2014)
    - More improvements in NEWHOPE
- NEWHOPE-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Parameters chosen to enable fast implementations (NTT)

NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
  - Originally proposed by Ding (2012)
  - Improvements by Peikert (2014)
  - More improvements in NEWHOPE
- NEWHOPE-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Parameters chosen to enable fast implementations (NTT)
- Centered binomial noise ψ_k (HW(a)$-$HW(b) for k-bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup

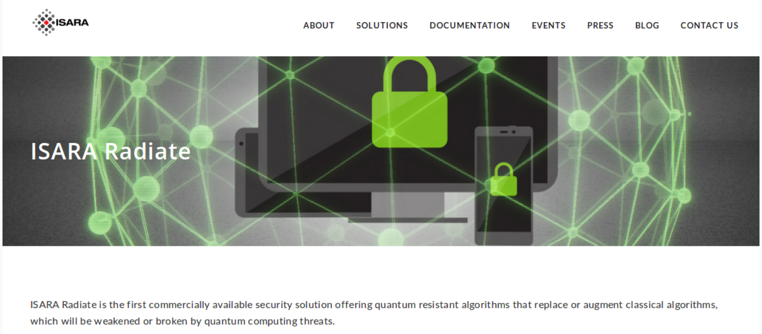
NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
    - Originally proposed by Ding (2012)
    - Improvements by Peikert (2014)
    - More improvements in NEWHOPE
- NEWHOPE-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Parameters chosen to enable fast implementations (NTT)
- Centered binomial noise ψ_k (HW(a)−HW(b) for k-bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup
- Choose a fresh parameter $\mathbf{a}$ for every protocol run (more later)

NEWHOPE (USENIX Security 2016)

- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconcilation to go from approximate agreement to agreement
  - Originally proposed by Ding (2012)
  - Improvements by Peikert (2014)
  - More improvements in NEWHOPE
- NEWHOPE-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Parameters chosen to enable fast implementations (NTT)
- Centered binomial noise ψ_k (HW(a)$-$HW(b) for k-bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup
- Choose a fresh parameter $\mathbf{a}$ for every protocol run (more later)
- Multiple implementations

Beyond the paper. . .



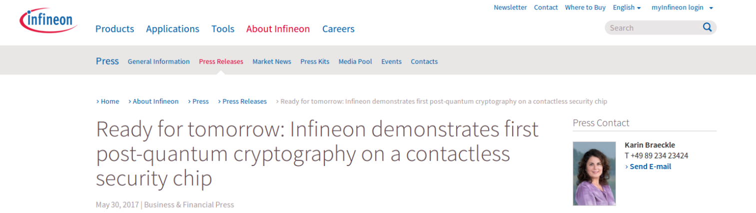
ISARA Radiate

ISARA Radiate is the first commercially available security solution offering quantum resistant algorithms that replace or augment classical algorithms, which will be weakened or broken by quantum computing threats.

“Key Agreement using the ‘NewHope’ lattice-based algorithm detailed in the New Hope paper, and LUKE (Lattice-based Unique Key Exchange), an ISARA speed-optimized version of the NewHope algorithm.”

`https://www.isara.com/isara-radiate/`

Beyond the paper. . .



Ready for tomorrow: Infineon demonstrates first post-quantum cryptography on a contactless security chip

May 30, 2017 | Business & Financial Press

Press Contact

Karin Braeckle
T +49 89 234 23424
› Send E-mail

“The deployed algorithm is a variant of “New Hope”, a quantum-resistant cryptosystem”

`https://www.infineon.com/cms/en/about-infineon/press/press-releases/2017/INFCCS201705-056.html`

Beyond the paper. . .

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Experimenting with Post-Quantum Cryptography

July 7, 2016

Posted by Matt Braithwaite, Software Engineer

“We’re indebted to Erdem Alkim, Léo Ducas, Thomas Pöppelmann and Peter Schwabe, the researchers who developed “New Hope”, the post-quantum algorithm that we selected for this experiment.”

`https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`

Also back in 2016: NIST PQC

- National Institute of Standards and Technology
- Public call for PQC proposals, aims at finding schemes for standardization
- Similar to earlier AES and SHA-3 efforts
- Process draft online in August 2016, comments by September 2016
- Call for proposals in December 2016, deadline November 2017

Also back in 2016: NIST PQC

- National Institute of Standards and Technology
- Public call for PQC proposals, aims at finding schemes for standardization
- Similar to earlier AES and SHA-3 efforts
- Process draft online in August 2016, comments by September 2016
- Call for proposals in December 2016, deadline November 2017

How it went (so far)

Nov. 2017
69 proposals

Round 1 →

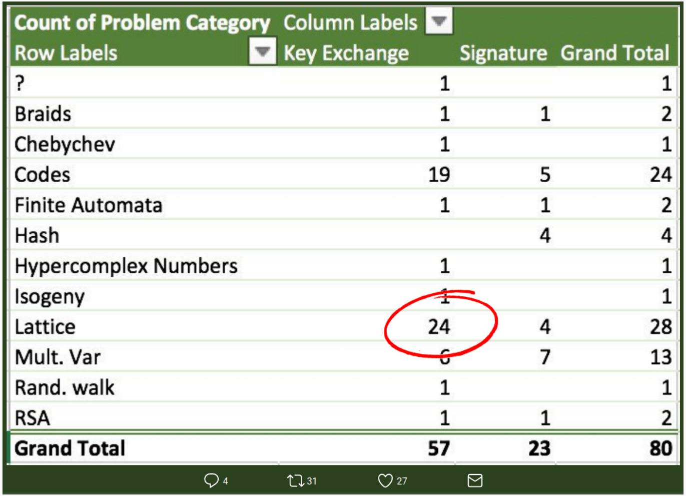
Feb. 2019
26 proposals

Round 2 →

Jul. 2020
7+8 proposals

Round 3 →

Jul. 2022
4 “winners”



Count of Problem Category	Column Labels		
Row Labels	Key Exchange	Signature	Grand Total
?	1		1
Braids	1	1	2
Chebychev	1		1
Codes	19	5	24
Finite Automata	1	1	2
Hash		4	4
Hypercomplex Numbers	1		1
Isogeny	1		1
Lattice	24	4	28
Mult. Var	6	7	13
Rand. walk	1		1
RSA	1	1	2
Grand Total	**57**	**23**	**80**

Overview tweeted by Jacob Alperin-Sheriff on Dec 4, 2017.

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- **Encrypt:**
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- **Encrypt:**
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- **Decrypt:**
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m})$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- **Encrypt:**
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- **Decrypt:**
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m})$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
    - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
    - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- **Encrypt:**
    - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
    - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
    - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- **Decrypt:**
    - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- Keygen:
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- Encrypt:
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- Decrypt:
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$
  - Compute $\mathbf{m} = \mathbf{v} \cdot \mathbf{f}_p \mod p$

Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters q and $p = 3$
- **Keygen:**
  - Find $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ and $\mathbf{f}_q = \mathbf{f}^{-1} \mod q, \mathbf{f}_p = \mathbf{f}^{-1} \mod p$
  - public key: $\mathbf{h} = p\mathbf{f}_q\mathbf{g}$, secret key: $(\mathbf{f}, \mathbf{f}_p)$
- **Encrypt:**
  - Map message m to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
- **Decrypt:**
  - Compute $\mathbf{v} = \mathbf{f} \cdot \mathbf{e} = \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) = \mathbf{f}(\mathbf{r} \cdot (p\mathbf{f}_q\mathbf{g}) + \mathbf{m}) = p\mathbf{r}\mathbf{g} + \mathbf{f} \cdot \mathbf{m}$
  - Compute $\mathbf{m} = \mathbf{v} \cdot \mathbf{f}_p \mod p$
- Advantages/Disadvantages compared to LPR:
  - Asymptotically weaker than Ring-LWE approach
  - Slower keygen, but faster encryption/decryption

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
    - q typically either prime or a power of two
    - f typically of degree between 512 and 1024

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
    - q typically either prime or a power of two
    - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - q typically either prime or a power of two
  - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$

Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
    - q typically either prime or a power of two
    - f typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, n prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** q prime, $f = (X^n + 1)$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** q prime, $f = (X^n - X - 1)$ irreducible, n prime (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$
- NewHope and Kyber have fastest (NTT-based) arithmetic

Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$

Design space 2: module vs. ring?

- “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
- Alternative: Module-LWE (MLWE):
  - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE

Design space 2: module vs. ring?

- “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
- Alternative: Module-LWE (MLWE):
    - Choose smaller n, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
    - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
- MLWE eliminates some of the structure of Ring-LWE

Design space 2: module vs. ring?

- “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
- Alternative: Module-LWE (MLWE):
  - Choose smaller *n*, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
- MLWE eliminates some of the structure of Ring-LWE
- MLWE can very easily scale security (change dimension of matrix):
  - Optimize arithmetic in $\mathcal{R}_q$ once
  - Use same optimized $\mathcal{R}_q$ arithmetic for all security levels

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
    - more security from the underlying hard problem
    - higher failure probability of decryption

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext
  - **LWE or LWR**
    - LWE considered more conservative (independent noise)
    - LWR easier to implement (no noise sampling)
    - LWR allows more compact public key and ciphertext

Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in $\{-1, 0, 1\}$) not conservative
    - Wide noise requires larger q (or more failures)
    - Larger q means larger public key and ciphertext
  - **LWE or LWR**
    - LWE considered more conservative (independent noise)
    - LWR easier to implement (no noise sampling)
    - LWR allows more compact public key and ciphertext
  - **Fixed-weight noise or not?**
    - Fixed-weight noise needs random permutation (sorting)
    - Naive implementations leak secrets through timing
    - Advantage of fixed-weight: easier to bound (or eliminate) decryption failures

Design space 4: public parameters?

- “Traditional” approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  “Let $\mathbf{a}$ *be a uniformly random. . . ”*

Design space 4: public parameters?

- “Traditional” approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  “Let $\mathbf{a}$ *be a uniformly random…”*

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once

Design space 4: public parameters?

- “Traditional” approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  “Let $\mathbf{a}$ *be a uniformly random. . . ”*

- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- “Solution”: Nothing-up-my-sleeves (involves endless discussion!)

Design space 4: public parameters?

- “Traditional” approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  “Let $\mathbf{a}$ be a uniformly random…”
- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- “Solution”: Nothing-up-my-sleeves (involves endless discussion!)
- Even without backdoor:
  - Perform massive precomputation based on $\mathbf{a}$
  - Use precomputation to break *all* key exchanges
  - Infeasible today, but who knows…
  - Attack in the spirit of Logjam

Design space 4: public parameters?

- "Traditional" approach to choosing $\mathbf{a}$ in LWE/LWR schemes:

  "Let $\mathbf{a}$ *be a uniformly random. . . "*
- Before NewHope: *real-world* approach: generate fixed $\mathbf{a}$ once
- What if $\mathbf{a}$ is backdoored?
- Parameter-generating authority can break key exchange
- "Solution": Nothing-up-my-sleeves (involves endless discussion!)
- Even without backdoor:
  - Perform massive precomputation based on $\mathbf{a}$
  - Use precomputation to break *all* key exchanges
  - Infeasible today, but who knows. . .
  - Attack in the spirit of Logjam
- Solution in NewHope: Choose a fresh $\mathbf{a}$ every time ("Against all authority")
- Server can cache $\mathbf{a}$ for some time (e.g., 1h)
- All NIST PQC candidates ended up using this approach

Design space 5: active security

- Decryption failures are a function of $\mathbf{s}$, $\mathbf{e}$, $\mathbf{s}'$, $\mathbf{e}'$
- Attacker can choose larger secret/noise $\mathbf{e}'$ and $\mathbf{s}'$
- Observe if decryption fails
- Learn something about $\mathbf{s}$

Design space 5: active security

- Decryption failures are a function of $\mathbf{s}$, $\mathbf{e}$, $\mathbf{s}'$, $\mathbf{e}'$
- Attacker can choose larger secret/noise $\mathbf{e}'$ and $\mathbf{s}'$
- Observe if decryption fails
- Learn something about $\mathbf{s}$
- This is a chosen ciphertext attack (CCA)
- Learn full $\mathbf{s}$ after a few thousand queries

Design space 5: active security

- Decryption failures are a function of $\mathbf{s}$, $\mathbf{e}$, $\mathbf{s}'$, $\mathbf{e}'$
- Attacker can choose larger secret/noise $\mathbf{e}'$ and $\mathbf{s}'$
- Observe if decryption fails
- Learn something about $\mathbf{s}$
- This is a chosen ciphertext attack (CCA)
- Learn full $\mathbf{s}$ after a few thousand queries
- NEWHOPE never claimed CCA-security!
- This “attack” is completely expected
- Not a problem for ephemeral $\mathbf{s}$

Design space 5: active security (ctd.)

The Fujisaki-Okamoto Transform (idea)

- Build CCA-secure KEM from passively secure encryption scheme
- Make failure probability negligible for honest $\mathbf{s}'$, $\mathbf{e}'$, $\mathbf{e}''$
- Force encapsulator to generate $\mathbf{s}'$, $\mathbf{e}'$, $\mathbf{e}''$ honestly

Design space 5: active security (ctd.)

The Fujisaki-Okamoto Transform

Alice (Server)		Bob (Client)
Gen():		Encaps(pk):
$\mathrm{pk}, \mathrm{sk} \leftarrow \mathrm{KeyGen}()$	$\xrightarrow{\mathrm{pk}}$	$x \leftarrow \{0, \ldots, 255\}^{32}$ $k, \mathrm{coins} \leftarrow \text{SHA3-512}(x)$
	$\xleftarrow{\mathrm{ct}}$	$\mathrm{ct} \leftarrow \mathrm{Encrypt}(\mathrm{pk}, x, \mathrm{coins})$
Decaps((sk, pk), ct): $x' \leftarrow \mathrm{Decrypt}(\mathrm{sk}, \mathrm{ct})$ $k', coins' \leftarrow \text{SHA3-512}(x')$ $\mathrm{ct}' \leftarrow \mathrm{Encrypt}(\mathrm{pk}, x', \mathrm{coins}')$ **verify if** $\mathrm{ct} = \mathrm{ct}'$		

Design space 5: active security (ctd.)

- Ephemeral key exchange does not need active security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication

Design space 5: active security (ctd.)

- Ephemeral key exchange does not need active security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication
- **Advantages:**
  - Higher failure probability $\rightarrow$ more compact
  - Simpler to implement, no CCA transform
  - More flexibility for secret/noise generation

Design space 5: active security (ctd.)

- Ephemeral key exchange does not need active security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication
- Advantages:
  - Higher failure probability → more compact
  - Simpler to implement, no CCA transform
  - More flexibility for secret/noise generation
- Disadvantages:
  - Less robust (will somebody reuse keys?)
  - More options (CCA vs. CPA): easier to make mistakes

Design space 6: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
    - Hash public-key into coins: multitarget protection (for non-zero failure probability)

Design space 6: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory

Design space 6: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
    - Hash public-key into coins: multitarget protection (for non-zero failure probability)
    - Hash public-key into shared key: KEM becomes contributory
    - Hash ciphertext into shared key: binding properties

Design space 6: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: binding properties
- How to handle rejection?
  - Return special symbol (`return -1`): explicit
  - Return $\mathrm{H}(s, C)$ for secret s: implicit

Design space 6: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: binding properties
- How to handle rejection?
  - Return special symbol (**`return -1`**): explicit
  - Return $\mathrm{H}(s, C)$ for secret s: implicit
- From round 2, no proposal used explicit rejection
  - Would break some security reduction
  - More robust in practice (return value always `0`)
  - Various recent papers argue **for** explicit rejection

Design space 7: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)

Design space 7: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., 2^{-30}
  - Reduce size of public key and ciphertext

Design space 7: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger q)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., 2^{-30}
  - Reduce size of public key and ciphertext
- Active (CCA) security needs negligible failure probability

Design space 8: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- “Encrypt” messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability

Design space 8: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- "Encrypt" messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding

Design space 8: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- "Encrypt" messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding
- LAC, Round5: more advanced ECC
  - Correct more errors, obtain smaller public key and ciphertext
  - More complex to implement, in particular without leaking through timing

KYBER

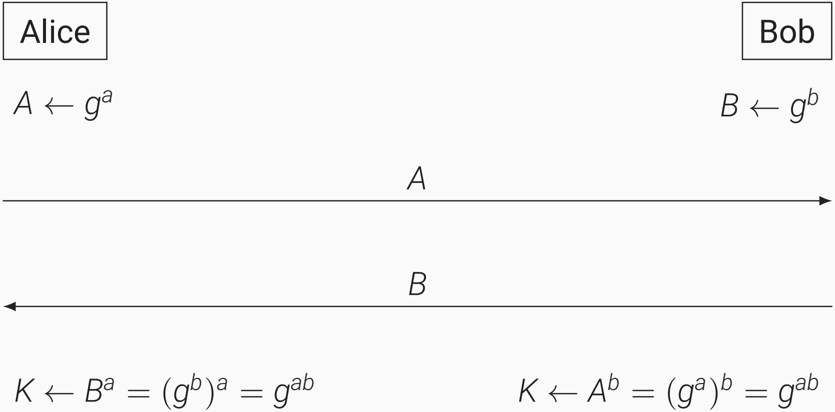
THE KEM

Roberto Avanzi, **Joppe Bos**, **Léo Ducas**, **Eike Kiltz**,
Tancrède Lepoint, **Vadim Lyubashevsky**, **John M. Schanck**,
Peter Schwabe, **Gregor Seiler**, **Damien Stehle**, Jintai Ding

The design of Kyber / ML-KEM

- MLWE version of LPR encryption:
  - Small-dimension vectors and matrices over $\mathcal{R}_q = Zq[X]/(X^{256}+1)$, $q = 3329$
  - Narrow, centered binomial noise with $k = 2$ or $k = 3$
  - Three parameter sets: Kyber-512, Kyber-768, Kyber-1024
- Tweaked FO transform (hash pk into coins and shared key)
- No error-correcting codes; simple encoding of bits into coefficients
- Negligible probability of decryption errors
- All symmetric crypto based on Keccak permutation (SHA-3)

Kyber vs. ECDH: a KEM is not DH!



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

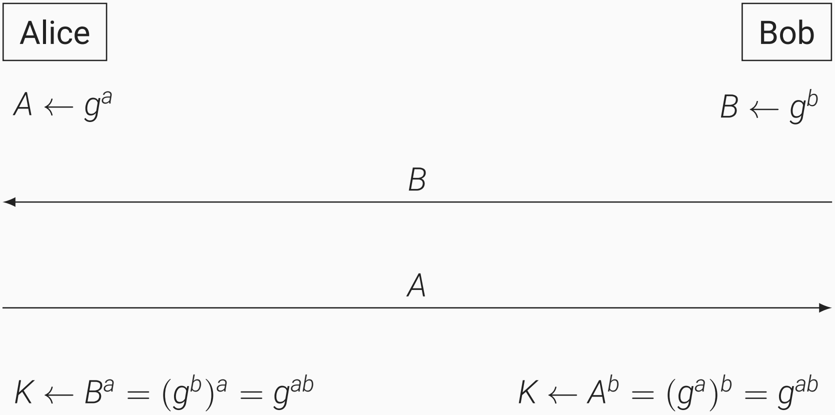
A

B

$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

Kyber vs. ECDH: a KEM is not DH!



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

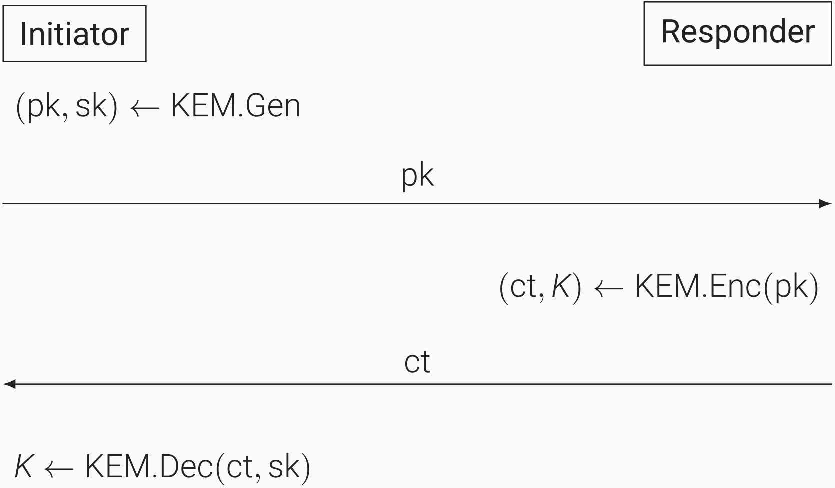
B

A

$K \leftarrow B^a = (g^b)^a = g^{ab}$

$K \leftarrow A^b = (g^a)^b = g^{ab}$

Kyber vs. ECDH: a KEM is not DH!



Initiator

Responder

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM.Gen}$

pk

$(\mathsf{ct}, K) \leftarrow \mathsf{KEM.Enc}(\mathsf{pk})$

ct

$K \leftarrow \mathsf{KEM.Dec}(\mathsf{ct}, \mathsf{sk})$

Kyber vs. ECDH: performance

X25519 speed

- keygen: 28187 Skylake cycles
- shared: 87942 Skylake cycles

Kyber-768 speed

- keygen: 39750 Skylake cycles
- encaps: 53936 Skylake cycles
- decaps: 42339 Skylake cycles

Kyber vs. ECDH: performance

X25519 speed

- keygen: 28187 Skylake cycles
- shared: 87942 Skylake cycles

X25519 sizes

- public key: 32 bytes

Kyber-768 speed

- keygen: 39750 Skylake cycles
- encaps: 53936 Skylake cycles
- decaps: 42339 Skylake cycles

Kyber-768 sizes

- public key: 1184 bytes
- ciphertext: 1088 bytes

Learn more

- Kyber website: `https://pq-crystals.org/kyber/`
- NIST PQC: `https://csrc.nist.gov/projects/post-quantum-cryptography`
- pqc-forum: `https://groups.google.com/a/list.nist.gov/g/pqc-forum`