Engineering Cryptographic Software

High-assurance post-quantum cryptography

Peter Schwabe

January 2026

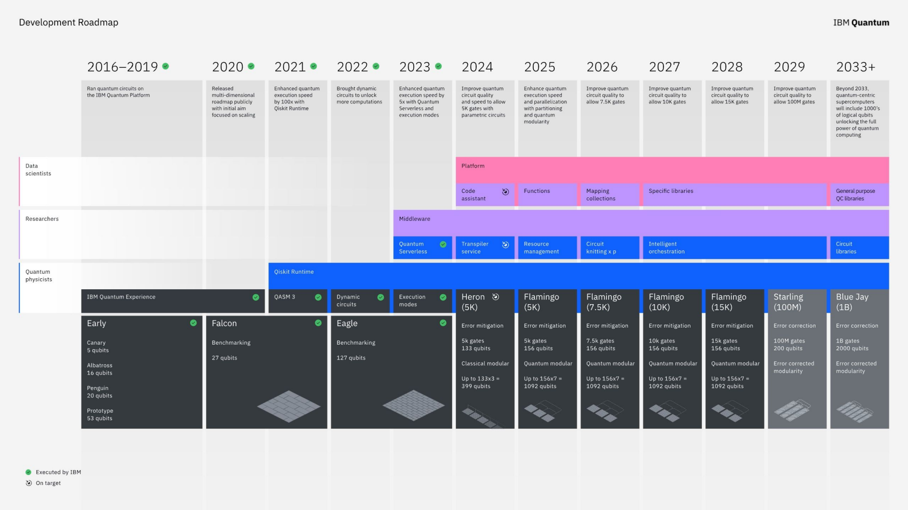
(EC)DH is everywhere

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.



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

Post-quantum crypto

Definition

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

Post-quantum crypto

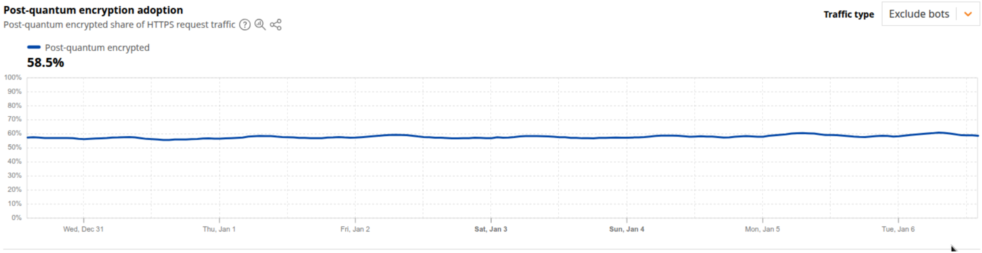
Definition

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

5 main directions

- Lattice-based crypto (PKE and Sigs)
- Code-based crypto (mainly PKE)
- Multivariate-based crypto (mainly Sigs)
- Hash-based signatures (only Sigs)
- Isogeny-based crypto (so far, mainly PKE)

Real-world impact: PQC deployment



https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption

- Hundreds of billions of connections per day at Cloudflare alone
- Also used in secure messaging (Signal, iMessage)
- Also in cloud infrastructure (AWS)

Hybrid deployment

"Post-quantum schemes should only be used in combination with classical schemes ("hybrid") if possible."

—Recommendations by the BSI

https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Quantentechnologien-und-Post-Quanten-Kryptografie/quantentechnologien-und-post-quanten-kryptografie_node.html

Hybrid deployment

"Post-quantum schemes should only be used in combination with classical schemes ("hybrid") if possible."

—Recommendations by the BSI

https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Quantentechnologien-und-Post-Quanten-Kryptografie/quantentechnologien-und-post-quanten-kryptografie_node.html

Motivation

- Cryptanalysis of PQ schemes is not as stable as for ECC
- Implementation security of PQ schemes is not as mature as for ECC

POST-QUANTUM KEY EXCHANGE

A NEW HOPE

ERDEM ALKIM
LÉO DUCAS
THOMAS PÖPPELMANN
PETER SCHWABE

Key Encapsulation Mechanisms (KEMs)

Initiator | **Responder**

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

Initiator → Responder: pk

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

Responder → Initiator: ct

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

Ring learning with errors (RLWE)

- Given $\mathbf{a}$, uniformly random
- Given “noise distribution” χ
- Given samples $\mathbf{as} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$

Ring learning with errors (RLWE)

- Given $\mathbf{a}$, uniformly random
- 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

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Short answer

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

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Longer answer

Polynomials with n coefficients, each coefficient in $\{0, \ldots, q-1\}$
Arithmetic uses reduction modulo q and modulo $(X^n + 1)$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned} \mathbf{a} + \mathbf{b} &= 10X^3 + 9X^2 + 2X + 5 \\ &= 3X^3 + 2X^2 + 2X + 5 \end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}\mathbf{a} - \mathbf{b} &= -2X^3 + X^2 + 2X - 1 \\ &= 5X^3 + X^2 + 2X + 6\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\ & 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\ & 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6 \\ = \quad & 24X^6 + 46X^5 + 32X^4 + 32X^3 + 23X^2 + 6\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}
\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\
& 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6 \\
= \quad & 24X^6 + 46X^5 + 32X^4 + 32X^3 + 23X^2 + 6 \\
= \quad & 3X^6 + 4X^5 + 4X^4 + 4X^3 + 2X^2 + 6
\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}
\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\
& 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6 \\
= \quad & 24X^6 + 46X^5 + 32X^4 + 32X^3 + 23X^2 + 6 \\
= \quad & 3X^6 + 4X^5 + 4X^4 + 4X^3 + 2X^2 + 6 \\
= \quad & -3X^2 - 4X - 4 + 4X^3 + 2X^2 + 6
\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$\begin{aligned}
\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\
& 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6 \\
= \quad & 24X^6 + 46X^5 + 32X^4 + 32X^3 + 23X^2 + 6 \\
= \quad & 3X^6 + 4X^5 + 4X^4 + 4X^3 + 2X^2 + 6 \\
= \quad & -3X^2 - 4X - 4 + 4X^3 + 2X^2 + 6 \\
= \quad & -X^2 - 4X + 4X^3 + 2
\end{aligned}$$

Where do $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{s}$ live?

Example

Let $q = 7$ and $n = 4$.
Let $\mathbf{a} = (4X^3 + 5X^2 + 2X + 2)$ and $\mathbf{b} = (6X^3 + 4X^2 + 3)$

$$
\begin{aligned}
\mathbf{a} \cdot \mathbf{b} = \quad & 24X^6 + 16X^5 + 12X^3 + 30X^5 + 20X^4 + 15X^2 + \\
& 12X^4 + 8X^3 + 6X + 12X^3 + 8X^2 + 6 \\
= \quad & 24X^6 + 46X^5 + 32X^4 + 32X^3 + 23X^2 + 6 \\
= \quad & 3X^6 + 4X^5 + 4X^4 + 4X^3 + 2X^2 + 6 \\
= \quad & -3X^2 - 4X - 4 + 4X^3 + 2X^2 + 6 \\
= \quad & -X^2 - 4X + 4X^3 + 2 \\
= \quad & 4X^3 + 6X^2 + 3X + 2
\end{aligned}
$$

How to build a KEM?

The basic idea

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

Alice has $\mathbf{v} = \mathbf{u}\mathbf{s} = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s}$

Bob has $\mathbf{v}' = \mathbf{b}\mathbf{s}' = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}'$

- 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
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}\quad)}$	
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}\quad)}$	

How to build a KEM, part 2

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

How to build a KEM, part 2

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

How to build a KEM, part 2

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

How to build a KEM, part 2

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

How to build a KEM, part 2

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

How to build a KEM, part 2

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

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 (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
- 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
- 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

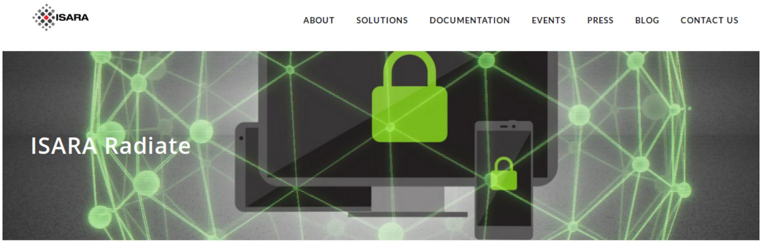
NewHope (USENIX Security 2016)

- 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
- 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

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
- 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
- Multiple implementations

Beyond the paper. . .

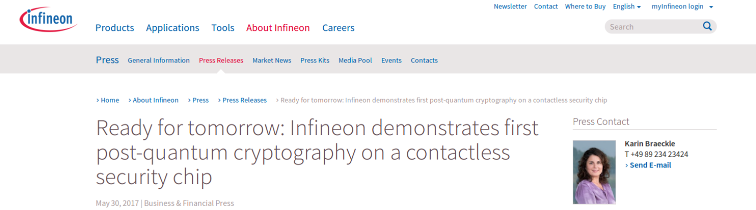


ISARA Radiate is the first commercially available security solution offering quantum resistant algorithms that replace or augment classical algorithms, which will be weakened or broken by quantum computing threats.

"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. . .



“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

Search blog ...
Archive

“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

Nov. 2017
69 proposals
→ Round 1 →
Feb. 2019
26 proposals
→ Round 2 →
Jul. 2020
7+8 proposals
→ Round 3 →
Jul. 2022
4 "winners"

- Second KEM selected for standardization in March 2025

NIST PQC: the beginning

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**

4 31 27

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

KYBER

Roberto Avanzi
Léo Ducas
Vadim Lyubashevsky
Gregor Seiler

Joppe Bos
Eike Kiltz
John M. Schanck
Damien Stehlé

Jintai Ding
Tancrede Lepoint
Peter Schwabe

From NewHope to Kyber

MLWE instead of RLWE

IND-CCA2 Security

From NewHope to Kyber

MLWE instead of RLWE

- Easily scale security
- Optimized routines the same for all security levels

IND-CCA2 Security

From NewHope to Kyber

MLWE instead of RLWE

- Easily scale security
- Optimized routines the same for all security levels

IND-CCA2 Security

- Support static (or cached) keys
- More robust
- Useful for authenticated key exchange
- Easy to construct PKE

Module Learning with Errors (MLWE)

- RLWE uses arithmetic on large degree polynomials
- For example, NEWHOPE uses $n = 1024$, $q = 12289$

Module Learning with Errors (MLWE)

- RLWE uses arithmetic on large degree polynomials
- For example, NEWHOPE uses $n = 1024$, $q = 12289$
- MLWE uses matrices and vectors of smaller polynomials of small dimension

Module Learning with Errors (MLWE)

- RLWE uses arithmetic on large degree polynomials
- For example, NEWHOPE uses $n = 1024$, $q = 12289$
- MLWE uses matrices and vectors of smaller polynomials of small dimension
- Kyber: $n = 256$, $q = 3329$
  - Security level 1 (AES-128): $d = 2$
  - Security level 3 (AES-192): $d = 3$
  - Security level 5 (AES-256): $d = 4$
- Core arithmetic is in $\mathbb{Z}_{3329}[X]/(X^{256} + 1)$ **for all security levels**

Module Learning with Errors (MLWE)

- RLWE uses arithmetic on large degree polynomials
- For example, NEWHOPE uses $n = 1024$, $q = 12289$
- MLWE uses matrices and vectors of smaller polynomials of small dimension
- Kyber: $n = 256$, $q = 3329$
  - Security level 1 (AES-128): $d = 2$
  - Security level 3 (AES-192): $d = 3$
  - Security level 5 (AES-256): $d = 4$
- Core arithmetic is in $\mathbb{Z}_{3329}[X]/(X^{256} + 1)$ **for all security levels**
- Noise is centered binomial $\text{HW}(x) - \text{HW}(y)$ for 2-bit x and y

Chosen-ciphertext attacks

- 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 $\mathbf{s}$

Chosen-ciphertext attacks

- 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

Chosen-ciphertext attacks

- 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 $\mathbf{s}$

From passive to CCA security

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

From passive to CCA security

The Fujisaki-Okamoto Transform

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

Kyber → ML-KEM

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

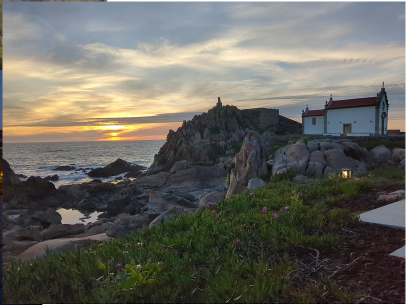
Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.203

Published August 13, 2024

February 2020. . .



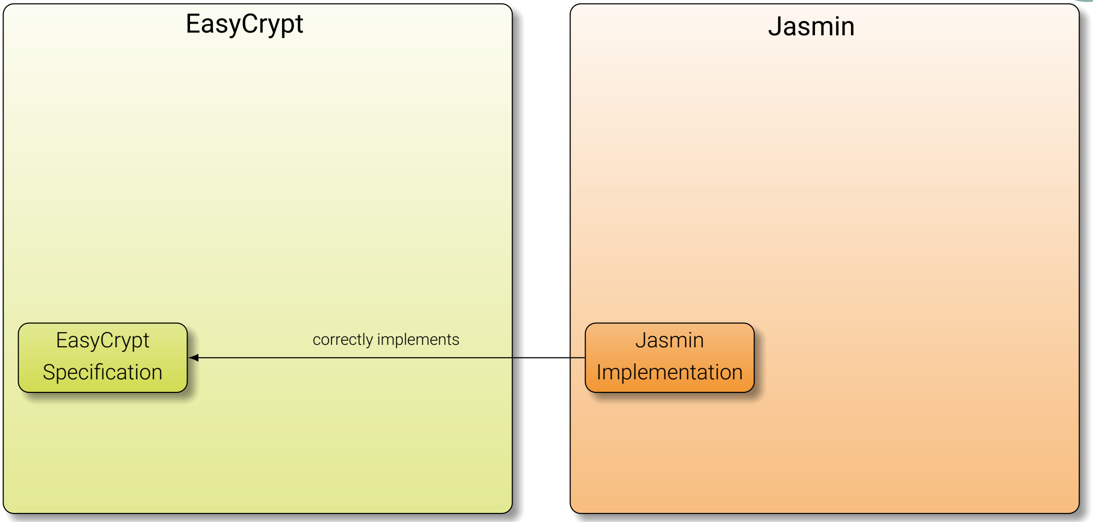
Goal: Formally verified implementation of Kyber

Almost 6 years later

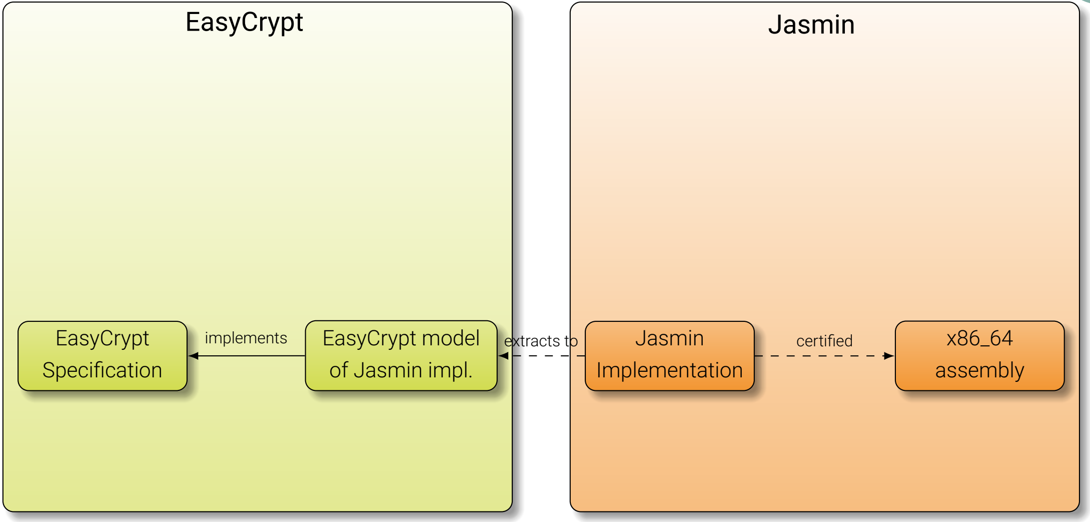
≈9 papers, 30+ collaborators

Basavesh Ammanaghatta Shivakumar, **Santiago Arranz Olmos**, José Bacelar Almeida, Gustavo Xavier Delerue Marinho Alves, Manuel Barbosa, **Francisca Barros**, Gilles Barthe, Lionel Blatter, Chitchanok Chuengsatiansup, Ignacio Cuevas, François Dupressoir, Luís Esquível, Ruben Gonzalez, **Benjamin Grégoire**, Andreas Hülsing, Vincent Hwang, Jan Jancar, Matthias Kannwischer, **Vincent Laporte**, **Jean-Christophe Léchenet**, Ting-han Lim, Cameron Low, Tiago Oliveira, Hugo Pacheco, Swarn Priya, Miguel Quaresma, Rolfe Schmidt, Antoine Séré, Lucas Tabary-Maujean, Pierre-Yves Strub, Yuval Yarom, Zhiyuan Zhang, **Jieyu Zheng**

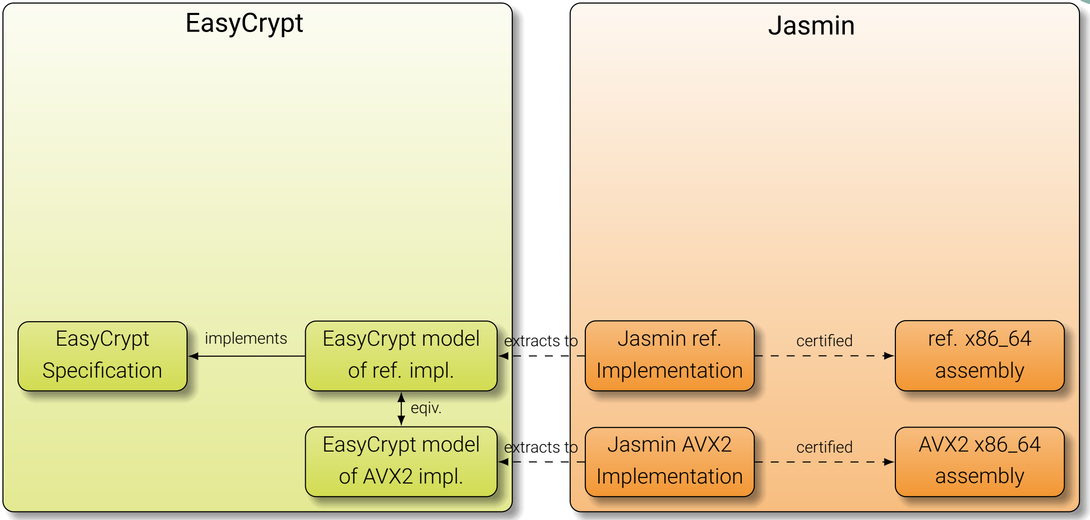
My naive view back in 2020



Refining our high-level view



Refining our high-level view



EasyCrypt

- Interactive verification framework
- Two languages, one functional, one imperative
- Main purpose: Security reductions, game-hopping proofs
  - Security goals and hardness assumptions as probabilistic programs
  - Support for standard arguments used in crypto proofs
  - “Formally verify typical pen-and-paper security proofs”

EasyCrypt

- Interactive verification framework
- Two languages, one functional, one imperative
- Main purpose: Security reductions, game-hopping proofs
  - Security goals and hardness assumptions as probabilistic programs
  - Support for standard arguments used in crypto proofs
  - “Formally verify typical pen-and-paper security proofs”
- Additional feature: correctness proofs
  - Relate functional and imperative programs
  - Prove equivalence of imperative programs

EasyCrypt

- Interactive verification framework
- Two languages, one functional, one imperative
- Main purpose: Security reductions, game-hopping proofs
  - Security goals and hardness assumptions as probabilistic programs
  - Support for standard arguments used in crypto proofs
  - “Formally verify typical pen-and-paper security proofs”
- Additional feature: correctness proofs
  - Relate functional and imperative programs
  - Prove equivalence of imperative programs
- Proofs are manual, but with some automation

The specification

- Manuel translation from PDF (Kyber spec, FIPS 203) to EasyCrypt
- Uses both imperative and functional features
- Some interesting tradeoffs, e.g., input/output types for NTT
  - Strong typing: two different isomorphic rings
  - Weaker typing: both just coefficient arrays
  - Weaker typing allows in-place NTT

The specification

- Manuel translation from PDF (Kyber spec, FIPS 203) to EasyCrypt
- Uses both imperative and functional features
- Some interesting tradeoffs, e.g., input/output types for NTT
  - Strong typing: two different isomorphic rings
  - Weaker typing: both just coefficient arrays
  - Weaker typing allows in-place NTT
- Translation and checking could be avoided by **machine-readable standards**

The specification

Algorithm 14 K-PKE.Encrypt($\mathsf{ek}_{\mathsf{PKE}}, m, r$)

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

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

$N \leftarrow 0$
$\hat{\mathbf{t}} \leftarrow \mathsf{ByteDecode}_{12}(\mathsf{ek}_{\mathsf{PKE}}[0 : 384k])$
$\rho \leftarrow \mathsf{ek}_{\mathsf{PKE}}[384k : 384k + 32]$
for $(i \leftarrow 0; i < k; i{+}{+})$
  for $(j \leftarrow 0; j < k; j{+}{+})$
    $\hat{\mathbf{A}}[i,j] \leftarrow \mathsf{SampleNTT}(\rho \| j \| i)$
  end for
end for
for $(i \leftarrow 0; i < k; i{+}{+})$
  $\mathbf{y}[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_1}(\mathsf{PRF}_{\eta_1}(r, N))$
  $N \leftarrow N + 1$
end for
for $(i \leftarrow 0; i < k; i{+}{+})$
  $\mathbf{e_1}[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$
  $N \leftarrow N + 1$
end for
$e_2 \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$
$\hat{\mathbf{y}} \leftarrow \mathsf{NTT}(\mathbf{y})$
$\mathbf{u} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \hat{\mathbf{y}}) + \mathbf{e_1}$
$\mu \leftarrow \mathsf{Decompress}_1(\mathsf{ByteDecode}_1(m))$
$v \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^\top \circ \hat{\mathbf{y}}) + e_2 + \mu$
$c_1 \leftarrow \mathsf{ByteEncode}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u}))$
$c_2 \leftarrow \mathsf{ByteEncode}_{d_v}(\mathsf{Compress}_{d_v}(v))$
return $c \leftarrow (c_1 \| c_2)$

```
proc enc_derand(pk : pkey, m : plaintext, r : W8.t Array32.t) : ciphertext = {
  (tv,rho) ⟵ pk;
  _N ⟵ 0;
  thati ⟵ EncDec.decode12_vec(tv);
  that ⟵ ofipolyvec thati;
  i ⟵ 0;
  while (i < kvec) {
    j ⟵ 0;
    while (j < kvec) {
      XOF(O).init(rho,W8.of_int i, W8.of_int j);
      c ⟵ Parse(XOF,O).sample();
      aT[(i,j)] ⟵ c;
      j ⟵ j + 1;
    }
    i ⟵ i + 1;
  }
  i ⟵ 0;
  while (i < kvec) {
    c ⟵ CBD2(PRF).sample(r,_N);
    yv ⟵ set yv i c;
    _N ⟵ _N + 1;
    i ⟵ i + 1;
  }
  i ⟵ 0;
  while (i < kvec) {
    c ⟵ CBD2(PRF).sample(r,_N);
    e1 ⟵ set e1 i c;
    _N ⟵ _N + 1;
    i ⟵ i + 1;
  }
  e2 ⟵ CBD2(PRF).sample(r,_N);
  yhat ⟵ nttv yv;
  u ⟵ invnttv (ntt_mmul aT yhat) + e1;
  mp ⟵ EncDec.decode1(m);
  v ⟵ invntt (ntt_dotp that yhat) + e2 + decompress_poly 1 mp;
  c1 ⟵ EncDec.encode10_vec(compress_polyvec 10 u);
  c2 ⟵ EncDec.encode4(compress_poly 4 v);
  return (c1,c2);
}
```

Mechanizing the security proof

Goal: Link specification to IND-CCA notion

Goal: Link specification to IND-CCA notion

Step 1: IND-CPA security of K-PKE

- Typical game-hopping proof for abstract LWE-based PKE
- Refine to concrete parameters of K-PKE
- Reduction from *hashed MLWE*

Goal: Link specification to IND-CCA notion

Step 1: IND-CPA security of K-PKE

- Typical game-hopping proof for abstract LWE-based PKE
- Refine to concrete parameters of K-PKE
- Reduction from *hashed MLWE*

Step 2: Decryption-failure bounds for K-PKE

- Original analysis assumes compression of uniform coefficients
- Cannot rely on MLWE assumption in correctness game
- Move from adversarial notion to statistic notion
- Distribution too complex to fully compute

Mechanizing the security proof

Goal: Link specification to IND-CCA notion

Step 1: IND-CPA security of K-PKE

- Typical game-hopping proof for abstract LWE-based PKE
- Refine to concrete parameters of K-PKE
- Reduction from *hashed MLWE*

Step 2: Decryption-failure bounds for K-PKE

- Original analysis assumes compression of uniform coefficients
- Cannot rely on MLWE assumption in correctness game
- Move from adversarial notion to statistic notion
- Distribution too complex to fully compute

Step 3: IND-CCA security of ML-KEM

- Prove IND-CCA for MLKEM_OP
- Central difference: SHA3-512 replaced by RO

Linking the spec to the code

Spec ↔ ref impl.

- Montgomery and Barrett reductions
- Lazy reductions (bounds checking!)
- “Funny” code, e.g., for `a = a/3329`

```
a *= 80635;
a >>= 28;
```

- Magic values in NTT code
- Link functional and imperative code

Linking the spec to the code

Spec ↔ ref impl.

- Montgomery and Barrett reductions
- Lazy reductions (bounds checking!)
- "Funny" code, e.g., for `a = a/3329`

```
a *= 80635;
a >>= 28;
```

- Magic values in NTT code
- Link functional and imperative code

ref. impl. ↔ AVX2 impl.

- Optimizations *across* functions
- Different internal data representations
- Non-word-size operations (compress)
- Complex vectorized rejection sampling

Linking the spec to the code

Spec ↔ ref impl.

- Montgomery and Barrett reductions
- Lazy reductions (bounds checking!)
- "Funny" code, e.g., for `a = a/3329`

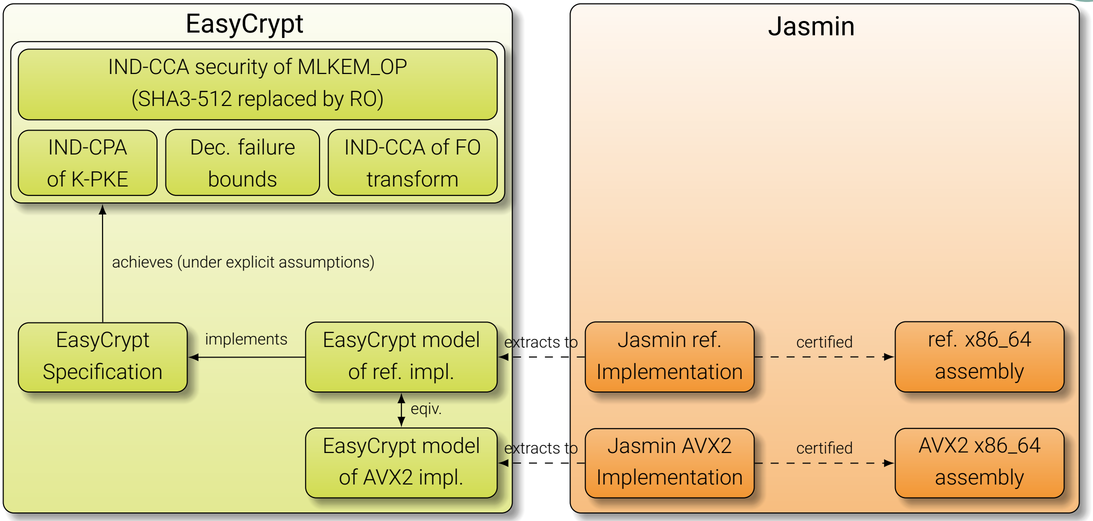
  ```
  a *= 80635;
  a >>= 28;
  ```
- Magic values in NTT code
- Link functional and imperative code

ref. impl. ↔ AVX2 impl.

- Optimizations *across* functions
- Different internal data representations
- Non-word-size operations (compress)
- Complex vectorized rejection sampling

- Massive manual effort with quite some student frustration
- Significant improvement 2025:
  - *automated* circuit-equivalence checking
  - integrated in EasyCrypt, interleave with deductive reasoning
  - faster verification, more robust proofs, easier proof maintenance

So, where are we?



Safety and constant-time

Safety

- Initially: use "old" safety checker
- Very cumbersome, motivated safety-checker updates
- Now: new safety checker by Francisca Barros
- Modular, reasonably fast, support from EasyCrypt

Safety and constant-time

Safety

- Initially: use "old" safety checker
- Very cumbersome, motivated safety-checker updates
- Now: new safety checker by Francisca Barros
- Modular, reasonably fast, support from EasyCrypt

Constant-time

- Use Jasmin CT type system
- All `#declassify` easy to explain
- CT property preserved by compilation

Future-proof “constant time”

- Some arithmetic instructions also leak through timing
- Example: `DIV` (exploited in “KyberSlash”, CHES 2025)
- Which instructions are safe to use, *also for future CPUs*?

Future-proof “constant time”

- Some arithmetic instructions also leak through timing
- Example: `DIV` (exploited in “KyberSlash”, CHES 2025)
- Which instructions are safe to use, *also for future CPUs*?
- Intel Core Security Team, 2022: Data Operand Independent Timing (DOIT)
  - Subset of x86_64 instructions **guaranteed** to not leak through timing
  - Requires switching CPU to DOIT mode

Future-proof “constant time”

- Some arithmetic instructions also leak through timing
- Example: `DIV` (exploited in “KyberSlash”, CHES 2025)
- Which instructions are safe to use, *also for future CPUs*?
- Intel Core Security Team, 2022: Data Operand Independent Timing (DOIT)
  - Subset of x86_64 instructions **guaranteed** to not leak through timing
  - Requires switching CPU to DOIT mode
- Support for DOIT in Jasmin compiler
  - Use only DOIT instructions on secret inputs
  - No serious limitation for most *optimized* code
  - All relevant vector instructions are DOIT

Spectre v1 (“Speculative bounds-check bypass”)

```
stack u8[10] public;
stack u8[32] secret;
reg u8 t;
reg u64 r, i;

i = 0;
while(i < 10) {
  t = public[(int) i] ;
  r = leak(t);
  ...
}
```

Countermeasures

Fencing

- Can prevent speculation through **barriers** (LFENCE)
- Protecting *all* branches is possible but costly

Countermeasures

Fencing

- Can prevent speculation through **barriers** (LFENCE)
- Protecting *all* branches is possible but costly

Speculative Load Hardening

- Idea: maintain misprediction predicate `ms` (in a register)
- At every branch use arithmetic to update predicate
- Option 1: Mask every loaded value with `ms`
- Option 2: Mask every address with `ms`
- Effect: during misspeculation “leak” constant value

Countermeasures

Fencing

- Can prevent speculation through **barriers** (LFENCE)
- Protecting *all* branches is possible but costly

Speculative Load Hardening

- Idea: maintain misprediction predicate `ms` (in a register)
- At every branch use arithmetic to update predicate
- Option 1: Mask every loaded value with `ms`
- Option 2: Mask every address with `ms`
- Effect: during misspeculation “leak” constant value
- Implemented in LLVM since version 8
  - Still large performance overhead
  - No formal guarantees of security

Do we need to mask/protect all loads?

Do we need to mask/protect all loads?

- No need to mask loads into registers that never enter leaking instructions

Do we need to mask/protect all loads?

- No need to mask loads into registers that never enter leaking instructions
- `secret` registers never enter leaking instructions!
- Obvious idea: mask only loads into `public` registers

Extending the type system

- Type system gets three security levels:
  - **`secret`:** secret
  - **`public`:** public, also during misspeculation
  - **`transient`:** public, but possibly secret during misspeculation

Extending the type system

- Type system gets three security levels:
  - `secret:` secret
  - `public:` public, also during misspeculation
  - `transient:` public, but possibly secret during misspeculation
- Maintain misspeculation flag `ms`:
  - `ms = #init_msf()`: Translate to `LFENCE`, set `ms` to zero
  - `ms = #set_msf(b, ms)`: Set `ms` according to branch condition `b`
  - Branches invalidate `ms`

Extending the type system

- Type system gets three security levels:
  - `secret:` secret
  - `public:` public, also during misspeculation
  - `transient:` public, but possibly secret during misspeculation
- Maintain misspeculation flag `ms`:
  - `ms = #init_msf()`: Translate to `LFENCE`, set `ms` to zero
  - `ms = #set_msf(b, ms)`: Set `ms` according to branch condition `b`
  - Branches invalidate `ms`
- Two operations to lower level:
  - `x = #protect(x, ms)`: Go from `transient` to `public`
  - `#protect` translates to mask by `ms`

Extending the type system

- Type system gets three security levels:
  - `secret:` secret
  - `public:` public, also during misspeculation
  - `transient:` public, but possibly secret during misspeculation
- Maintain misspeculation flag `ms`:
  - `ms = #init_msf()`: Translate to `LFENCE`, set `ms` to zero
  - `ms = #set_msf(b, ms)`: Set `ms` according to branch condition `b`
  - Branches invalidate `ms`
- Two operations to lower level:
  - `x = #protect(x, ms)`: Go from `transient` to `public`
  - `#protect` translates to mask by `ms`
  - `#declassify r`: Go from `secret` to `transient`
  - `#declassify` requires cryptographic proof/argument

Extending the type system

- Type system gets three security levels:
  - `secret`: secret
  - `public`: public, also during misspeculation
  - `transient`: public, but possibly secret during misspeculation
- Maintain misspeculation flag `ms`:
  - `ms = #init_msf()`: Translate to `LFENCE`, set `ms` to zero
  - `ms = #set_msf(b, ms)`: Set `ms` according to branch condition `b`
  - Branches invalidate `ms`
- Two operations to lower level:
  - `x = #protect(x, ms)`: Go from `transient` to `public`
  - `#protect` translates to mask by `ms`
  - `#declassify r`: Go from `secret` to `transient`
  - `#declassify` requires cryptographic proof/argument
- Still: allow branches and indexing only for `public`
- Also, allow non-DOIT instructions only on `public` inputs

The special case of crypto

- We know what inputs are **secret** and what inputs are **public**
- Most of the state is actually **secret**
- Most loads do not need `protect`!

The special case of crypto

- We know what inputs are **secret** and what inputs are **public**
- Most of the state is actually **secret**
- Most loads do not need `protect`!
- Even better: mark additional inputs as **secret**
- No cost if those inputs don't flow into leaking instructions

The special case of crypto

- We know what inputs are **secret** and what inputs are **public**
- Most of the state is actually **secret**
- Most loads do not need `protect`!
- Even better: mark additional inputs as **secret**
- No cost if those inputs don't flow into leaking instructions
- Even better: Spills don't need `protect` if there is no branch between store and load

The special case of crypto

- We know what inputs are **secret** and what inputs are **public**
- Most of the state is actually **secret**
- Most loads do not need `protect`!
- Even better: mark additional inputs as **secret**
- No cost if those inputs don't flow into leaking instructions
- Even better: Spills don't need `protect` if there is no branch between store and load
- Even better: "Spill" public data to MMX registers instead of stack

If there's a Spectre v1. . .

- Spectre v2: indirect branches (not supported by Jasmin)
- Spectre v3, aka "Meltdown": fix in HW and firmware
- Spectre v4: speculative store bypass (disable with SSBD)

If there's a Spectre v1. . .

- Spectre v2: indirect branches (not supported by Jasmin)
- Spectre v3, aka "Meltdown": fix in HW and firmware
- Spectre v4: speculative store bypass (disable with SSBD)
- Spectre-RSB: function return speculates *anywhere*
- Attacker can choose to speculate right behind any defense!

If there's a Spectre v1. . .

- Spectre v2: indirect branches (not supported by Jasmin)
- Spectre v3, aka "Meltdown": fix in HW and firmware
- Spectre v4: speculative store bypass (disable with SSBD)
- **Spectre-RSB**: function return speculates *anywhere*
- Attacker can choose to speculate right behind any defense!
- Solution in Jasmin:
  - Rewrite all returns through "branch table"
  - Implement branch table through *conditional* branches
  - Speculate only to one of the call sites
  - At call sites all **public** values become **transient**
  - Use `#protect` and `ms`

Clearing sensitive data

“. . . A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys”

—FIPS 140-3, Section 9.7.A

Clearing sensitive data

"...A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys"

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Clearing sensitive data

“. . . A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys”

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Failure modes

0. Don’t perform any zeroization

Clearing sensitive data

". . . A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys"

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Failure modes

0. Don't perform any zeroization
1. Dead-store elimination

Clearing sensitive data

“. . . A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys”

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Failure modes

0. Don’t perform any zeroization
1. Dead-store elimination
2. Only API-level stack zeroization

Clearing sensitive data

"...A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys"

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Failure modes

0. Don't perform any zeroization
1. Dead-store elimination
2. Only API-level stack zeroization
3. Don't scrub source-level invisible data

Clearing sensitive data

“. . . A cryptographic module shall provide methods to zeroize all plaintext secret and private cryptographic keys”

—FIPS 140-3, Section 9.7.A

Goal of zeroization

Scrub all (sensitive) data from memory (stack) and registers when crypto routine returns.

Failure modes

0. Don’t perform any zeroization
1. Dead-store elimination
2. Only API-level stack zeroization
3. Don’t scrub source-level invisible data
4. Mis-estimate stack space when scrubbing from caller

Clearing sensitive data (ctd.)

Solution in Jasmin compiler

Zeroize used stack space and registers when returning from export function

Clearing sensitive data (ctd.)

Solution in Jasmin compiler

Zeroize used stack space and registers when returning from export function

- Make use of multiple features of Jasmin:
  - Compiler has global view
  - All stack usage is known at compile time
  - Entry/return point is clearly defined

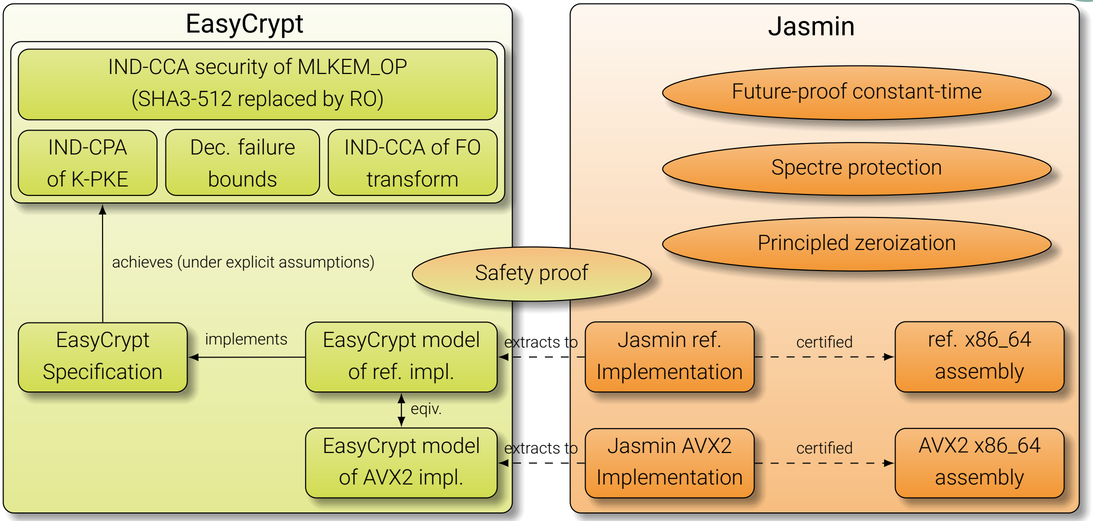
Clearing sensitive data (ctd.)

Solution in Jasmin compiler

Zeroize used stack space and registers when returning from export function

- Make use of multiple features of Jasmin:
  - Compiler has global view
  - All stack usage is known at compile time
  - Entry/return point is clearly defined
- Performance overhead for Kyber768 (on Comet Lake):
  - 0.59% for Keypair
  - 0.24% for Encaps
  - 1.04% for Decaps

The next generation of crypto software



Is it fast?

Cycles for ML-KEM-768

CPU	Implementation	keypair	enc	dec
8700K	Jasmin AVX2*	40134	40599	43437
	pq-crystals	39722	39761	46161
11700K	Jasmin AVX2*	37458	37798	39970
	pq-crystals	36958	38082	42566
13900K	Jasmin AVX2*	34732	35212	43784
	pq-crystals	31448	32090	36064

*with Spectre-v1 (without Spectre-RSB) protections

Stay tuned

Work in progress (selection)

- Deployment in Signal's contact discovery (RWC talk upcoming!)
- Integrate Spectre-RSB protections
- Implement in **crypto agent** process
- Extend to more architectures and more primitives (next up: ML-DSA)
- Interface to super-optimizers (e.g., CryptOpt, SLOTHY)
- Improve usability/scalability of tools

Longer-term plans (selection)

- Guarantee preservation of speculative constant time
- Masked implementations
- Machine-readable standards (?)

References

- Barbosa, Kannwischer, Lim, Schwabe, Strub. *Formally Verified Correctness Bounds for Lattice-Based Cryptography.* ACM CCS 2025.
- Almeida, Marinho Alves, Barbosa, Barthe, Esquível, Hwang, Oliveira, Pacheco, Schwabe, Strub. *Faster Verification of Faster Implementations: Combining Deductive and Circuit-Based Reasoning in EasyCrypt.* IEEE S&P 2025.
- Arranz Olmos, Barthe, Grégoire, Jancar, Laporte, Oliveira, Schwabe. *Let's DOIT: Using Intel's Extended HW/SW Contract for Secure Compilation of Crypto Code.* TCHES 2025-3.
- Arranz Olmos, Barthe, Chuengsatiansup, Grégoire, Laporte, Oliveira, Schwabe, Yarom, Zhang. *Protecting Cryptographic Code Against Spectre-RSB (and, in Fact, All Known Spectre Variants).* ASPLOS 2025.
- Barbosa, Schwabe. *Kyber terminates.* Polynesian Journal of Mathematics, vol. 1, issue 6 (2024).

References

- Almeida, Arranz Olmos, Barbosa, Barthe, Dupressoir, Grégoire, Laporte, Léchenet, Low, Oliveira, Pacheco, Quaresma, Schwabe, Strub. *Formally verifying Kyber – Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt.* CRYPTO 2024.
- Arranz Olmos, Barthe, Gonzalez, Grégoire, Laporte, Léchenet, Oliveira, Schwabe. *High-assurance zeroization.* TCHES 2024-1.
- Almeida, Barbosa, Barthe, Grégoire, Laporte, Léchenet, Oliveira, Pacheco, Quaresma, Schwabe, Séré, Strub. *Formally verifying Kyber – Episode IV: Implementation Correctness.* TCHES 2023-3.
- Ammanaghatta Shivakumar, Barthe, Grégoire, Laporte, Oliveira, Priya, Schwabe, Tabary-Maujean. *Typing High-Speed Cryptography against Spectre v1.* IEEE S&P 2023.

Thank you!

https://github.com/pq-code-package/mlkem-libjade

https://formosa-crypto.org