# Post-quantum crypto on embedded microcontrollers

Peter Schwabe
peter@cryptojedi.org
https://cryptojedi.org

December 4, 2019

# Crypto today

5 building blocks for a “secure channel”
**Symmetric crypto**

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

# Crypto today

5 building blocks for a "secure channel"
**Symmetric crypto**

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

**Asymmetric crypto**

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)

# Crypto today

## 5 building blocks for a "secure channel"

**Symmetric crypto**

- Block or stream cipher (e.g., AES, ChaCha20)
- Authenticator (e.g., HMAC, GMAC, Poly1305)
- Hash function (e.g., SHA-2, SHA-3)

**Asymmetric crypto**

- Key agreement / public-key encryption (e.g., RSA, Diffie-Hellman, ECDH)
- Signatures (e.g., RSA, DSA, ECDSA, EdDSA)

## The asymmetric monoculture

- All widely deployed asymmetric crypto relies on
  - the **hardness of factoring**, or
  - the **hardness of (elliptic-curve) discrete logarithms**

# 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.

*"In the past, people have said, maybe it's 50 years away, it's a dream, maybe it'll happen sometime. I used to think it was 50. Now I'm thinking like it's 15 or a little more. It's within reach. It's within our lifetime. It's going to happen."*

—Mark Ketchen (IBM), Feb. 2012, about quantum computers

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

**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)

# The NIST competition

| 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.

# The NIST competition (ctd.)

"Key exchange"

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(\mathrm{vk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}()$
  - $(c, k) \leftarrow \mathrm{Encaps}(\mathrm{vk})$
  - $k \leftarrow \mathrm{Decaps}(c, \mathrm{sk})$

# The NIST competition (ctd.)

## "Key exchange"

- What is meant is **key encapsulation mechanisms** (KEMs)
  - $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
  - $(c, k) \leftarrow \mathsf{Encaps}(\mathsf{vk})$
  - $k \leftarrow \mathsf{Decaps}(c, \mathsf{sk})$

## Status of the NIST competition

- In total 69 submissions accepted as "complete and proper"
- Several broken, 5 withdrawn
- Jan 2019: NIST announces 26 round-2 candidates
  - 17 KEMs and PKEs
  - 9 signature schemes

# Round-2 of the NIST PQC project

*"Performance (hardware+software) will play more of a role"*

*—Dustin Moody, May 2019*

# Round-2 of the NIST PQC project

*"Performance (hardware+software) will play more of a role"*

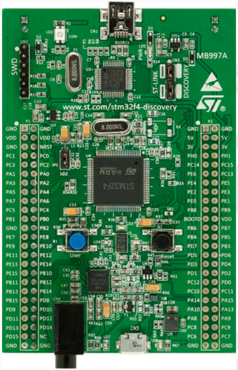
*—Dustin Moody, May 2019*

*"...we will recommend that teams generally focus their hardware implementation efforts on Cortex-M4"*

—Daniel Apon, Feb 2019

Joint work with
**Matthias Kannwischer, Joost Rijneveld, and Ko Stoffelen.**

- Started as part of PQCRYPTO H2020 project
- Continued within EPOQUE ERC StG
- Library and testing/benchmarking framework
  - PQ-crypto on ARM Cortex-M4
  - Uses STM32F4 Discovery board
  - 192 KB of RAM, benchmarks at 24 MHz
- Easy to add schemes using NIST API
- Optimized SHA3 and AES shared across primitives



# pqm4 usage

- Run functional tests of all primitives and implementations:

```
python3 test.py
```

# pqm4 usage

- Run functional tests of all primitives and implementations:

```
python3 test.py
```

- Generate testvectors, compare for consistency (also with host):

```
python3 testvectors.py
```

# pqm4 usage

- Run functional tests of all primitives and implementations:

```
python3 test.py
```

- Generate testvectors, compare for consistency (also with host):

```
python3 testvectors.py
```

- Run speed and stack benchmarks:

```
python3 benchmarks.py
```

# pqm4 usage

- Run functional tests of all primitives and implementations:

```
python3 test.py
```

- Generate testvectors, compare for consistency (also with host):

```
python3 testvectors.py
```

- Run speed and stack benchmarks:

```
python3 benchmarks.py
```

- Easy to evaluate only subset of schemes, e.g.:

```
python3 test.py newhope1024cca sphincs-shake256-128s
```

# Signatures (not) in pqm4

| | **reference** | **optimized** |
|---|---|---|
| CRYSTALS-Dilithium | ✓ | (✓) |
| FALCON | ✗$_{RAM}$ | ✓ |
| GeMSS | ✗$_{Key}$ | — |
| LUOV | ✓ | — |
| MQDSS | ✗$_{RAM}$ | — |
| Picnic | ✗$_{RAM}$ | — |
| qTESLA | ✓ | — |
| Rainbow | ✗$_{Key}$ | — |
| SPHINCS+ | ✓ | — |

✗$_{Key}$: keys too large ✗$_{RAM}$: implementation uses too much RAM

✗$_{Lib}$: available implementations depend on external libraries

# KEMs (not) in pqm4

| | reference | optimized |
|---|---|---|
| BIKE | ✗$_{Lib}$ | — |
| Classic McEliece | ✗$_{Key}$ | — |
| CRYSTALS-Kyber | ✓ | ✓ |
| Frodo-KEM | ✓ | ✓ |
| HQC | ✗$_{Lib}$ | — |
| LAC | ✓ | — |
| LEDAcrypt | ✗$_{RAM}$ | WIP |
| NewHope | ✓ | ✓ |
| NTRU | ✓ | ✓ |
| NTRU Prime | ✓ | — |
| NTS-KEM | ✗$_{Key}$ | — |
| ROLLO | ✗$_{Lib}$ | — |
| Round5 | ✓ | ✓ |
| RQC | ✗$_{Lib}$ | — |
| SABER | ✓ | ✓ |
| SIKE | ✓ | — |
| ThreeBears | ✓ | ✓ |

✗$_{Key}$: keys too large ✗$_{RAM}$: implementation uses too much RAM

✗$_{Lib}$: available implementations depend on external libraries

# KEMs (not) in pqm4

| | **reference** | **optimized** |
|---|---|---|
| BIKE | ✗$_{Lib}$ | — |
| Classic McEliece | ✗$_{Key}$ | — |
| **CRYSTALS-Kyber** | ✓ | ✓ |
| **Frodo-KEM** | ✓ | ✓ |
| HQC | ✗$_{Lib}$ | — |
| **LAC** | ✓ | — |
| LEDAcrypt | ✗$_{RAM}$ | WIP |
| **NewHope** | ✓ | ✓ |
| **NTRU** | ✓ | ✓ |
| **NTRU Prime** | ✓ | — |
| NTS-KEM | ✗$_{Key}$ | — |
| ROLLO | ✗$_{Lib}$ | — |
| **Round5** | ✓ | ✓ |
| RQC | ✗$_{Lib}$ | — |
| **SABER** | ✓ | ✓ |
| SIKE | ✓ | — |
| **ThreeBears** | ✓ | ✓ |

✗$_{Key}$: keys too large ✗$_{RAM}$: implementation uses too much RAM

✗$_{Lib}$: available implementations depend on external libraries

# Learning with errors (LWE)

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

# Learning with errors (LWE)

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

# Learning with errors (LWE)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given "noise distribution" $\chi$
- Given samples $\mathbf{As} + \mathbf{e}$, with $\mathbf{e} \leftarrow \chi$
- Search version: find $\mathbf{s}$
- Decision version: distinguish from uniform random
- Structured lattices: work in $\mathbb{Z}_q[x]/f$

# Learning with rounding (LWR)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k\times\ell}$
- Given samples $\lceil \mathbf{As} \rfloor_p$, with $p < q$

# Learning with rounding (LWR)

- Given uniform $\mathbf{A} \in \mathbb{Z}_q^{k \times \ell}$
- Given samples $\lceil \mathbf{As} \rfloor_p$, with $p < q$
- Search version: find $\mathbf{s}$
- Decision version: distinguish from uniform random
- Structured lattices: work in $\mathbb{Z}_q[x]/f$

# Lattice-based KEMs – the basic idea

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

$$\begin{aligned} &\text{Alice has} \quad \mathbf{v} &&= \mathbf{us} &&= \mathbf{ass}' + \mathbf{e}'\mathbf{s} \\ &\text{Bob has} \quad \mathbf{v}' &&= \mathbf{bs}' &&= \mathbf{ass}' + \mathbf{es}' \end{aligned}$$

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

# Core operation: multiplication in $\mathcal{R}_q = \mathbb{Z}_q[X]/f$

Power-of-two $q$

- Several schemes use $q = 2^m$, for small $m$
- Examples: Round5, NTRU, Saber
- More round-1 examples: Kindi, RLizard

# Core operation: multiplication in $\mathcal{R}_q = \mathbb{Z}_q[X]/f$

## Power-of-two $q$

- Several schemes use $q = 2^m$, for small $m$
- Examples: Round5, NTRU, Saber
- More round-1 examples: Kindi, RLizard

## Prime "NTT-friendly" $q$

- Kyber and NewHope use prime $q$ supporting fast NTT
- For $A, B \in \mathcal{R}_q$, $A \cdot B = \mathrm{NTT}^{-1}(\mathrm{NTT}(A) \circ \mathrm{NTT}(B))$
- NTT is Fourier Transform over finite field
- Use $f = X^n + 1$ for power-of-two $n$

# Multiplication in $\mathbb{Z}_{2^m}[X]$

- Joint work with **Matthias Kannwischer** and **Joost Rijneveld**
- Represent coefficients as 16-bit integers
- No modular reductions required, $2^{16}$ is a multiple of $q = 2^m$

# Multiplication in $\mathbb{Z}_{2^m}[X]$

- Joint work with **Matthias Kannwischer** and **Joost Rijneveld**
- Represent coefficients as 16-bit integers
- No modular reductions required, $2^{16}$ is a multiple of $q = 2^m$
- Schoolbook multiplication takes $n^2$ integer muls, $(n-1)^2$ adds

# Multiplication in $\mathbb{Z}_{2^m}[X]$

- Joint work with **Matthias Kannwischer** and **Joost Rijneveld**
- Represent coefficients as 16-bit integers
- No modular reductions required, $2^{16}$ is a multiple of $q = 2^m$
- Schoolbook multiplication takes $n^2$ integer muls, $(n-1)^2$ adds
- Can do better using Karatsuba:

$$
\begin{aligned}
&(a_\ell + X^k a_h) \cdot (b_\ell + X^k b_h) \\
= \; & a_\ell b_\ell + X^k (a_\ell b_h + a_h b_\ell) + X^n a_h b_h \\
= \; & a_\ell b_\ell + X^k ((a_\ell + a_h)(b_\ell + b_h) - a_\ell b_\ell - a_h b_h) + X^n a_h b_h
\end{aligned}
$$

- Recursive application yields complexity $\Theta(n^{\log_2 3})$

# Multiplication in $\mathbb{Z}_{2^m}[X]$

- Joint work with **Matthias Kannwischer** and **Joost Rijneveld**
- Represent coefficients as 16-bit integers
- No modular reductions required, $2^{16}$ is a multiple of $q = 2^m$
- Schoolbook multiplication takes $n^2$ integer muls, $(n-1)^2$ adds
- Can do better using Karatsuba:

$$
\begin{aligned}
& (a_\ell + X^k a_h) \cdot (b_\ell + X^k b_h) \\
= \; & a_\ell b_\ell + X^k (a_\ell b_h + a_h b_\ell) + X^n a_h b_h \\
= \; & a_\ell b_\ell + X^k ((a_\ell + a_h)(b_\ell + b_h) - a_\ell b_\ell - a_h b_h) + X^n a_h b_h
\end{aligned}
$$

- Recursive application yields complexity $\Theta(n^{\log_2 3})$
- Generalization: Toom-Cook
  - Toom-3: split into 5 multiplications of 1/3 size
  - Toom-4: split into 7 multiplications of 1/4 size
- Approach: Evaluate, multiply, interpolate

# Initial observations

- Karatsuba/Toom is asymptotically faster, but isn't for "small" polynomials

# Initial observations

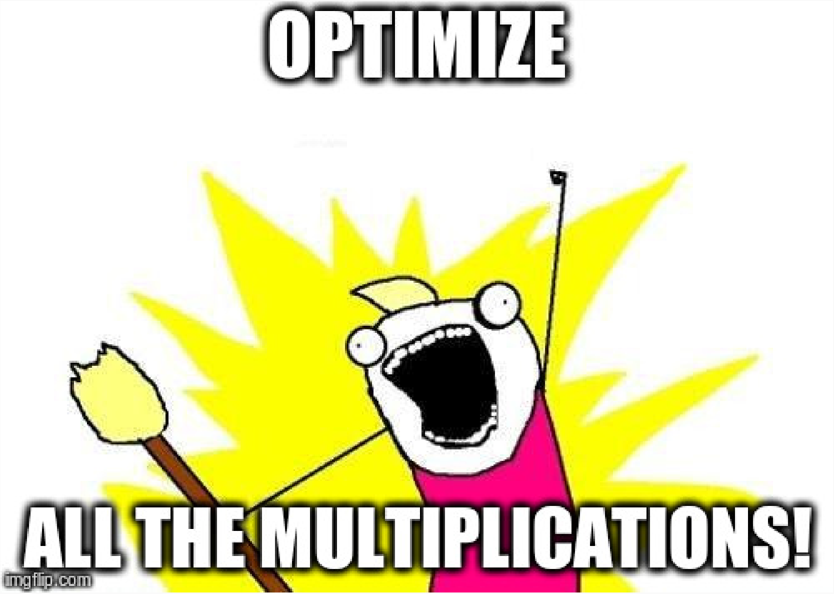
- Karatsuba/Toom is asymptotically faster, but isn't for "small" polynomials
- Toom-3 needs division by 2, loses 1 bit of precision
- Toom-4 needs division by 8, loses 3 bits of precision
- This limits recursive application when using 16-bit integers
- Can use Toom-4 only for $q \leq 2^{13}$

# Initial observations

- Karatsuba/Toom is asymptotically faster, but isn't for "small" polynomials
- Toom-3 needs division by 2, loses 1 bit of precision
- Toom-4 needs division by 8, loses 3 bits of precision
- This limits recursive application when using 16-bit integers
- Can use Toom-4 only for $q \leq 2^{13}$
- Karmakar, Bermudo Mera, Sinha Roy, Verbauwhede (CHES 2018):
    - Optimize Saber, $q = 2^{13}, n = 256$
    - Use Toom-4 + two levels of Karatsuba
    - Optimized 16-coefficient schoolbook multiplication

# Initial observations

- Karatsuba/Toom is asymptotically faster, but isn't for "small" polynomials
- Toom-3 needs division by 2, loses 1 bit of precision
- Toom-4 needs division by 8, loses 3 bits of precision
- This limits recursive application when using 16-bit integers
- Can use Toom-4 only for $q \leq 2^{13}$
- Karmakar, Bermudo Mera, Sinha Roy, Verbauwhede (CHES 2018):
  - Optimize Saber, $q = 2^{13}$, $n = 256$
  - Use Toom-4 + two levels of Karatsuba
  - Optimized 16-coefficient schoolbook multiplication
- **Is this the best approach? How about other values of $q$ and $n$?**


OPTIMIZE
ALL THE MULTIPLICATIONS!
imgflip.com

# Our approach

- Generate optimized assembly for Karatsuba/Toom
- Use Python scripts, receive as input $n$ and $q$
- Hand-optimize "small" schoolbook multiplications
  - Make heavy use of DSP "vector instructions"
  - Perform two $16 \times 16$-bit multiply-accumulate in one cycle
  - Carefully schedule instructions to minimize loads/stores
- Benchmark different options, pick fastest

# Multiplication results

| | approach | "small" | cycles | stack |
|---|---|---|---|---|
| Saber ($n = 256,\ q = 2^{13}$) | Karatsuba only | 16 | 41 121 | 2 020 |
| | Toom-3 | 11 | 41 225 | 3 480 |
| | **Toom-4** | **16** | **39 124** | **3 800** |
| | Toom-4 + Toom-3 | - | - | - |
| Kindi-256-3-4-2 ($n = 256,\ q = 2^{14}$) | **Karatsuba only** | **16** | **41 121** | **2 020** |
| | Toom-3 | 11 | 41 225 | 3 480 |
| | Toom-4 | - | - | - |
| | Toom-4 + Toom-3 | - | - | - |
| NTRU-HRSS ($n = 701,\ q = 2^{13}$) | Karatsuba only | 11 | 230 132 | 5 676 |
| | Toom-3 | 15 | 217 436 | 9 384 |
| | **Toom-4** | **11** | **182 129** | **10 596** |
| | Toom-4 + Toom-3 | - | - | - |
| NTRU-KEM-743 ($n = 743,\ q = 2^{11}$) | Karatsuba only | 12 | 247 489 | 6 012 |
| | Toom-3 | 16 | 219 061 | 9 920 |
| | **Toom-4** | **12** | **196 940** | **11 208** |
| | Toom-4 + Toom-3 | 16 | 197 227 | 12 152 |
| RLizard-1024 ($n = 1024,\ q = 2^{11}$) | Karatsuba only | 16 | 400 810 | 8 188 |
| | Toom-3 | 11 | 360 589 | 13 756 |
| | **Toom-4** | **16** | **313 744** | **15 344** |
| | Toom-4 + Toom-3 | 11 | 315 788 | 16 816 |

# NTT-based multiplication

- Joint work with **Leon Botros** and **Matthias Kannwischer**
- Primary goal: optimize Kyber
- Secondary effect: optimize NewHope (improved by Gérard)

# NTT-based multiplication

- Joint work with **Leon Botros** and **Matthias Kannwischer**
- Primary goal: optimize Kyber
- Secondary effect: optimize NewHope (improved by Gérard)
- NTT is an FFT in a finite field
- Evaluate polynomial $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$ at all $n$-th roots of unity
- Divide-and-conquer approach
    - Write polynomial $f$ as $f_0(X^2) + X f_1(X^2)$

# NTT-based multiplication

- Joint work with **Leon Botros** and **Matthias Kannwischer**
- Primary goal: optimize Kyber
- Secondary effect: optimize NewHope (improved by Gérard)
- NTT is an FFT in a finite field
- Evaluate polynomial $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$ at all $n$-th roots of unity
- Divide-and-conquer approach
  - Write polynomial $f$ as $f_0(X^2) + X f_1(X^2)$
  - Huge overlap between evaluating

$$f(\beta) = f_0(\beta^2) + \beta f_1(\beta^2) \text{ and}$$
$$f(-\beta) = f_0(\beta^2) - \beta f_1(\beta^2)$$

# NTT-based multiplication

- Joint work with **Leon Botros** and **Matthias Kannwischer**
- Primary goal: optimize Kyber
- Secondary effect: optimize NewHope (improved by Gérard)
- NTT is an FFT in a finite field
- Evaluate polynomial $f = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$ at all $n$-th roots of unity
- Divide-and-conquer approach
  - Write polynomial $f$ as $f_0(X^2) + X f_1(X^2)$
  - Huge overlap between evaluating

$$f(\beta) = f_0(\beta^2) + \beta f_1(\beta^2) \text{ and}$$
$$f(-\beta) = f_0(\beta^2) - \beta f_1(\beta^2)$$

  - $f_0$ has $n/2$ coefficients
  - Evaluate $f_0$ at all $(n/2)$-th roots of unity by recursive application
  - Same for $f_1$

# NTT-based multiplication

- First thing to do: replace recursion by iteration
- Loop over $\log n$ levels with $n/2$ "butterflies" each

# NTT-based multiplication

- First thing to do: replace recursion by iteration
- Loop over $\log n$ levels with $n/2$ "butterflies" each
- Butterfly on level $k$:
  - Pick up $f_i$ and $f_{i+2^k}$
  - Multiply $f_{i+2^k}$ by a power of $\omega$ to obtain $t$
  - Compute $f_{i+2^k} \leftarrow a_i - t$
  - Compute $f_i \leftarrow a_i + t$

# NTT-based multiplication

- First thing to do: replace recursion by iteration
- Loop over $\log n$ levels with $n/2$ "butterflies" each
- Butterfly on level $k$:
    - Pick up $f_i$ and $f_{i+2^k}$
    - Multiply $f_{i+2^k}$ by a power of $\omega$ to obtain $t$
    - Compute $f_{i+2^k} \leftarrow a_i - t$
    - Compute $f_i \leftarrow a_i + t$
- Main optimizations on Cortex-M4:
    - "Merge" levels: fewer loads/stores
    - Optimize modular arithmetic (precompute powers of $\omega$ in Montgomery domain)
    - Lazy reductions
    - Carefully optimize using DSP instructions

# Selected optimized lattice KEM cycles

| Scheme | Key Generation | Encapsulation | Decapsulation |
|---|---|---|---|
| ntruhps2048509 | 77 698 713 | 645 329 | 542 439 |
| ntruhps2048677 | 144 383 491 | 955 902 | 836 959 |
| ntruhps4096821 | 211 758 452 | 1 205 662 | 1 066 879 |
| ntruhrss701 | 154 676 705 | 402 784 | 890 231 |
| lightsaber | 459 965 | 651 273 | 678 810 |
| saber | 896 035 | 1 161 849 | 1 204 633 |
| firesaber | 1 448 776 | 1 786 930 | 1 853 339 |
| kyber512 | 514 291 | 652 769 | 621 245 |
| kyber768 | 976 757 | 1 146 556 | 1 094 849 |
| kyber1024 | 1 575 052 | 1 779 848 | 1 709 348 |
| newhope1024cpa | 975 736 | 975 452 | 162 660 |
| newhope1024cca | 1 161 112 | 1 777 918 | 1 760 470 |

**Comparison:** Curve25519 scalarmult: 625 358 cycles

# Selected optimized lattice KEM stack bytes

| Scheme | Key Generation | Encapsulation | Decapsulation |
|---|---:|---:|---:|
| ntruhps2048509 | 21 412 | 15 452 | 14 828 |
| ntruhps2048677 | 28 524 | 20 604 | 19 756 |
| ntruhps4096821 | 34 532 | 24 924 | 23 980 |
| ntruhrss701 | 27 580 | 19 372 | 20 580 |
| lightsaber | 9 656 | 11 392 | 12 136 |
| saber | 13 256 | 15 544 | 16 640 |
| firesaber | 20 144 | 23 008 | 24 592 |
| kyber512 | 2 952 | 2 552 | 2 560 |
| kyber768 | 3 848 | 3 128 | 3 072 |
| kyber1024 | 4 360 | 3 584 | 3 592 |
| newhope1024cpa | 11 096 | 17 288 | 8 308 |
| newhope1024cca | 11 080 | 17 360 | 19 576 |

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
    - HW accelerators for PQ crypto: start with Keccak

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
    - HW accelerators for PQ crypto: start with Keccak
    - Careful about API, want speedup also for hash-based signatures

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
    - HW accelerators for PQ crypto: start with Keccak
    - Careful about API, want speedup also for hash-based signatures
    - Faster Keccak will accelerate several more schemes (e.g., Dilithium)

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
  - HW accelerators for PQ crypto: start with Keccak
  - Careful about API, want speedup also for hash-based signatures
  - Faster Keccak will accelerate several more schemes (e.g., Dilithium)
- NTRU-HPS is currently additionally bottlenecked by slow constant-time sorting

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
  - HW accelerators for PQ crypto: start with Keccak
  - Careful about API, want speedup also for hash-based signatures
  - Faster Keccak will accelerate several more schemes (e.g., Dilithium)
- NTRU-HPS is currently additionally bottlenecked by slow constant-time sorting
- Some more speedups possible ($\approx$ 5%?) by using floating-point registers

# Some lessons learned and TODOs

- Lattice-based KEMs are bottlenecked by hashing
  - HW accelerators for PQ crypto: start with Keccak
  - Careful about API, want speedup also for hash-based signatures
  - Faster Keccak will accelerate several more schemes (e.g., Dilithium)
- NTRU-HPS is currently additionally bottlenecked by slow constant-time sorting
- Some more speedups possible ($\approx 5\%$?) by using floating-point registers
- For FALCON more speedups possible using floating-point arithmetic

# Resources online

- NIST PQC website:

  https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

- NIST "PQC forum" mailing list:

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

# Resources online

- NIST PQC website:
  https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
- NIST "PQC forum" mailing list:
  https://csrc.nist.gov/projects/post-quantum-cryptography/email-list
- pqm4 library and benchmarking suite:
  https://github.com/mupq/pqm4
- Code of $\mathbb{Z}_{2^m}[x]$ multiplication paper, including scripts:
  https://github.com/mupq/polymul-z2mx-m4
- $\mathbb{Z}_{2^m}[x]$ multiplication paper:
  https://cryptojedi.org/papers/#latticem4
- Kyber/NTT optimization paper:
  https://cryptojedi.org/papers/#nttm4