MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Post-Quantum Crypto Software – Embedded and High-Assurance

Peter Schwabe

June 28, 2023

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.

Where are we?

NIST PQC

Nov. 2017 69 proposals $\xrightarrow{\text{Round 1}}$ Feb. 2019 26 proposals $\xrightarrow{\text{Round 2}}$ Jul. 2020 7+8 proposals $\xrightarrow{\text{Round 3}}$ Jul. 2022 **4 “winners”**

Where are we?

NIST PQC

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

"The public-key encryption and key-establishment algorithm that will be standardized is ***CRYSTALS-KYBER****. The digital signatures that will be standardized are CRYSTALS-Dilithium, FALCON, and SPHINCS^{+}. While there are multiple signature algorithms selected, NIST recommends* ***CRYSTALS-Dilithium*** *as the primary algorithm to be implemented"*

—NIST IR 8413-upd1

Should you care now?

"Store now, decrypt later"

https://en.wikipedia.org/wiki/Utah_Data_Center#/media/File:EFF_photograph_of_NSA's_Utah_Data_Center.jpg

Should you care now? (part II)

Menü MOTORRAD

MOTORRAD Pur | Neuheiten | Motorräder | Bekleidung | Zubehör | Reisen | **Ratgeber** | Sport & Szene | Club | Markt

STARTSEITE > Ratgeber > Verkehr & Wirtschaft > Motorräder in Deutschland: Im Schnitt 19 Jahre alt

MOTORRÄDER IN DEUTSCHLAND SIND MEISTENS ALT

Motorräder: Im Durchschnitt grad erwachsen

Youngtimer dominieren: In Deutschland sind zugelassene Motorräder im Schnitt 19,1 Jahre alt.

Jens Kratschmar • 09.08.2022

What does this mean for systems engineers?

Current situation: ECC

Scalar multiplication takes $\approx$50K–100K cycles on 64-bit Intel CPU

Kyber and Dilithium

Kyber768 on Intel Haswell

- Keygen: 44 339 cycles
- Encaps: 60 142 cycles
- Decaps: 48 070 cycles

Dilithium3 on Intel Haswell

- Keygen: 173 344 cycles
- Sign: 359 302 cycles
- Verify: 177 284 cycles

What does this mean for systems engineers?

Current situation: ECC

Public keys have 32 bytes, signatures have 64 bytes

Kyber and Dilithium

Kyber768 sizes

- Public key: 1184 bytes
- Ciphertext: 1088 bytes

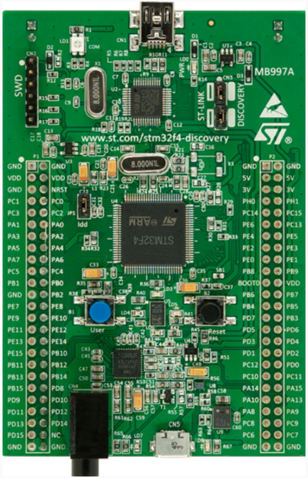
Dilithium3 sizes

- Public key: 1952 bytes
- Signature: 4000 bytes

How about embedded? – pqm4

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

- Library and testing/benchmarking framework
  - PQ-crypto on ARM Cortex-M4
  - Uses STM32F4 Discovery board
- Easy to add schemes using NIST API
- Benchmark speed and memory
- Optimized SHA3 and AES shared across primitives



Kyber and Dilithium in pqm4

Kyber

Cycles

- Keygen: 707 275
- Encaps: 867 363
- Decaps: 788 053

Stack bytes

- Keygen: 2784
- Encaps: 2856
- Decaps: 2872

Dilithium

Cycles

- Keygen: 2 830 024
- Sign: 6 588 465
- Verify: 2 691 283

Stack bytes

- Keygen: 60 836
- Encaps: 68 836
- Decaps: 57 724

Kyber and Dilithium in pqm4

Joppe W. Bos, Joost Renes, Amber Sprenkels. *Dilithium for Memory Constrained Devices*, Africacrypt 2022.

- Reduce Dilithium3 stack usage to <7 KB for signing, <3 KB for verification
- Significant slowdown, exact performance impact not clear

So, are we “done”?

1. Take existing optimized C/asm implementations
2. Possibly tweak for different tradeoffs
3. Possibly use HW accelerators (most important: for Keccak!)
4. Integrate into systems
5. Done.

Bugs, bugs everywhere

Dilithium commit on Dec. 28, 2017

```diff
-     t  = buf[pos];
-     t |= (uint32_t)buf[pos + 1] << 8;
-     t |= (uint32_t)buf[pos + 2] << 16;
-     t &= 0xFFFFF;
+     t0  = buf[pos];
+     t0 |= (uint32_t)buf[pos + 1] << 8;
+     t0 |= (uint32_t)buf[pos + 2] << 16;
+     t0 &= 0xFFFFF;

-     t  = buf[pos + 2] >> 4;
-     t |= (uint32_t)buf[pos + 3] << 4;
-     t |= (uint32_t)buf[pos + 4] << 12;
+     t1  = buf[pos + 2] >> 4;
+     t1 |= (uint32_t)buf[pos + 3] << 4;
+     t1 |= (uint32_t)buf[pos + 4] << 12;
```

- Bug in Dilithium sampler
- Two consecutive coefficients are equal
- Allows key recovery
- Reported by Peter Pessl on Dec. 27, 2017

Bugs, bugs everywhere

Questions about the range analysis of iNTT for "Faster Kyber and Dilithium on the Cortex-M4" #226

Closed **JunhaoHuang** opened this issue on Mar 3 · 4 comments

JunhaoHuang commented on Mar 3 • edited

Hi team, I am reading the Kyber code regarding the recent paper "Faster Kyber and Dilithium on the Cortex-M4", and I have a question about the matrix-vector product and Better Accumulation part regarding the *f_stack* version code.

I see that using the better accumulation technique in the f_speed version code, we can reduce each element of the output vector of matrix-vector product down to (-q,q). Since poly_invntt is normally used after the matrix-vector product, the range of the input vector of **poly_invntt** lies in (-q,q) in the *f_speed* version code. The **invntt** function works in this situation.

What I wonder is that in the *f_stack* version code, the **matacc** function actually uses the previous double basemul accumulation function, and it should produce the result vector with element in (-kq, kq), k is the security parameter of Kyber. For Kyber1024, the range of each polynomial element that **invntt** takes should be (-4q,4q). However, the **invntt** function is the same as the f_speed version code. The first four layers of the light butterflies in **invntt** involve some additions and subtractions without multiplication. Therefore, For Kyber1024 in the *f_stack* version code, two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the *f_stack* code and why does it still work?

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

Bugs, bugs everywhere

". . . two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?"

Bugs, bugs everywhere

“. . . two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?”

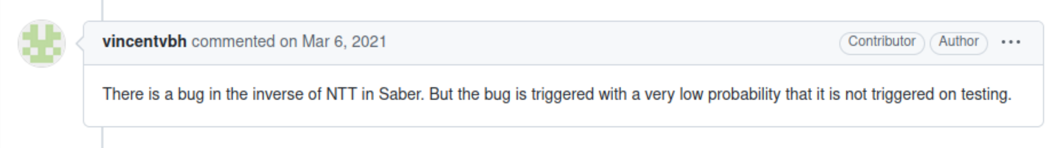
“. . . On your question on why it still works, I believe that this is an edge case that does not get triggered by the testing scripts.”

Bugs, bugs everywhere

vincentvbh commented on Mar 6, 2021 Contributor Author ...

There is a bug in the inverse of NTT in Saber. But the bug is triggered with a very low probability that it is not triggered on testing.

Bugs, bugs everywhere


vincentvbh commented on Mar 6, 2021 Contributor Author ...

There is a bug in the inverse of NTT in Saber. But the bug is triggered with a very low probability that it is not triggered on testing.

Both NTT bugs found by Yang, Liu, Shi, Hwang, Tsai, Wang, and Seiler (TCHES 2022/4)

Implementation security

Hardware side-channels

- Require physical access to device
- Examples: Power, EM attacks
- Protection through dedicated countermeasures
- Typical slowdown of much more than 100%
- Progress, but no “conclusion”; we don’t know how to protect PQC!

Implementation security

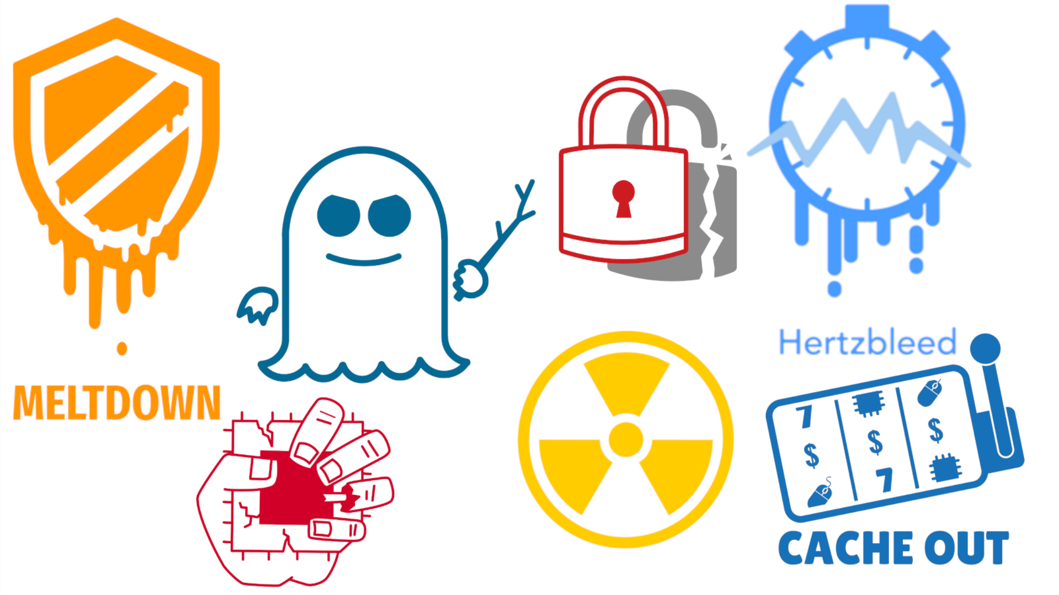
Hardware side-channels

- Require physical access to device
- Examples: Power, EM attacks
- Protection through dedicated countermeasures
- Typical slowdown of much more than 100%
- Progress, but no “conclusion”; we don’t know how to protect PQC!

Software side-channels

- Leak through microarchitectural side-channels
- No physical access required, can run *remotely*
- Traditional countermeasure: **constant-time**
  - No branching on secrets
  - No memory access at secret location
  - No variable-time arithmetic on secrets

While the cryptographers were busy. . .



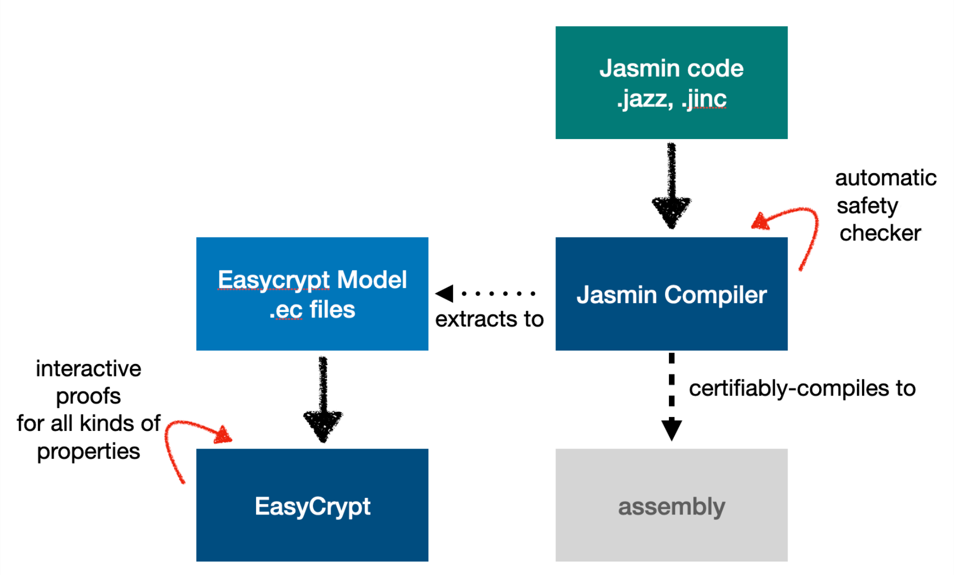
High-assurance PQC

FORMOSA CRYPTO

- Effort to **formally verify** crypto
- Currently three main projects:
  - EasyCrypt proof assistant
  - jasmin programming language
  - Libjade (PQ-)crypto library
- Core community of $\approx$ 30–40 people
- Discussion forum with $\approx$150 people

University of Bristol · Max Planck Institute for Security and Privacy · IMDEA Software Institute · U. Porto · INESC TEC · Radboud University · Inria · TU/e Eindhoven University of Technology

The toolchain and workflow



Libjade – Goals

- High-performance implementations of all NIST PQC primitives (first focus on Kyber and Dilithium)
- Multi-architecture support (first focus on AMD64)
- Easy “drop in” integration for most protocol libraries and systems

Libjade – Goals

- High-performance implementations of all NIST PQC primitives (first focus on Kyber and Dilithium)
- Multi-architecture support (first focus on AMD64)
- Easy “drop in” integration for most protocol libraries and systems
- Automated proofs of thread safety and memory safety
- Certified compilation to assembly

Libjade – Goals

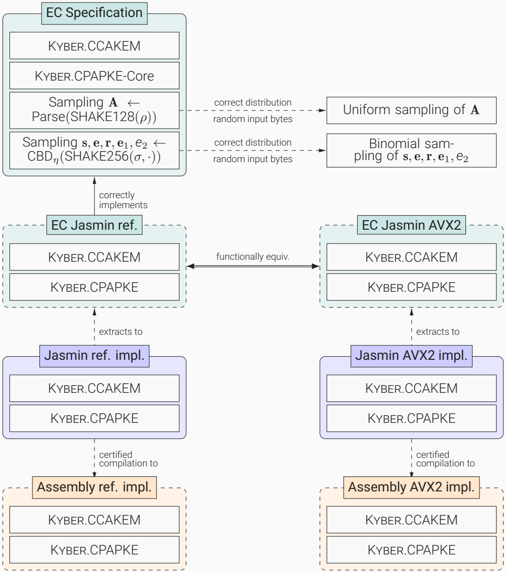
- High-performance implementations of all NIST PQC primitives (first focus on Kyber and Dilithium)
- Multi-architecture support (first focus on AMD64)
- Easy “drop in” integration for most protocol libraries and systems
- Automated proofs of thread safety and memory safety
- Certified compilation to assembly
- Verified resistance against “classical” timing attacks
- Verified resistance against (certain) Spectre attacks
- Verified memory zeroization on return

Libjade – Goals

- High-performance implementations of all NIST PQC primitives (first focus on Kyber and Dilithium)
- Multi-architecture support (first focus on AMD64)
- Easy “drop in” integration for most protocol libraries and systems
- Automated proofs of thread safety and memory safety
- Certified compilation to assembly
- Verified resistance against “classical” timing attacks
- Verified resistance against (certain) Spectre attacks
- Verified memory zeroization on return
- Computer-verified (manual) proofs of functional correctness
- Connection to computer-verified (manual) cryptographic proofs

Formally verified Kyber

- Specify Kyber in EasyCrypt
- Two jasmin implementations
- Interactive proofs of functional correctness
- Performance similar to optimized C/asm
- 3-year effort
- Improvements to jasmin/EasyCrypt



Almeida, Barbosa, Barthe, Grégoire, Laporte, Léchenet, Oliveira, Pacheco, Quaresma, Schwabe, Séré, and Strub. *Formally verifying Kyber – Episode IV: Implementation Correctness.* TCHES 2023-3

Spectre v1 (“Speculative bounds-check bypass”)

```
stack u8[16] public;
stack u8[32] secret;
reg u8 t;
reg u64 r, i;

i = 0;
while(i < 16) {
  t = public[(int) i] ;
  r = leak(t);
  ...
}
```

Protecting against Spectre v1

- Security type system in jasmin
- Enforce no branching on secrets, no memory access at secret position
- Also enforce this **in speculative execution after misspeculated conditional branch**

Protecting against Spectre v1

- Security type system in jasmin
- Enforce no branching on secrets, no memory access at secret position
- Also enforce this **in speculative execution after misspeculated conditional branch**
- Guide programmer to protect code
- Selective speculative load hardening (selSLH):
  - Misspeculation flag in register
  - Mask “transient” values with flag before leaking them

Protecting against Spectre v1

- Security type system in jasmin
- Enforce no branching on secrets, no memory access at secret position
- Also enforce this **in speculative execution after misspeculated conditional branch**
- Guide programmer to protect code
- Selective speculative load hardening (selSLH):
  - Misspeculation flag in register
  - Mask “transient” values with flag before leaking them
- Overhead for Kyber768 (on Intel Comet Lake):
  - 0.28% for Keypair
  - 0.55% for Encaps
  - 0.75% for Decaps
- Exploits synergies with protections against “traditional” timing attacks

Ammanaghatta Shivakumar, Barthe, Grégoire, Laporte, Oliveira, Priya, Schwabe, and Tabary-Maujean. *Typing High-Speed Cryptography against Spectre v1.* IEEE S&P 2023.

Interested?

https://github.com/mupq/pqm4

https://formosa-crypto.org

https://formosa-crypto.zulipchat.com/