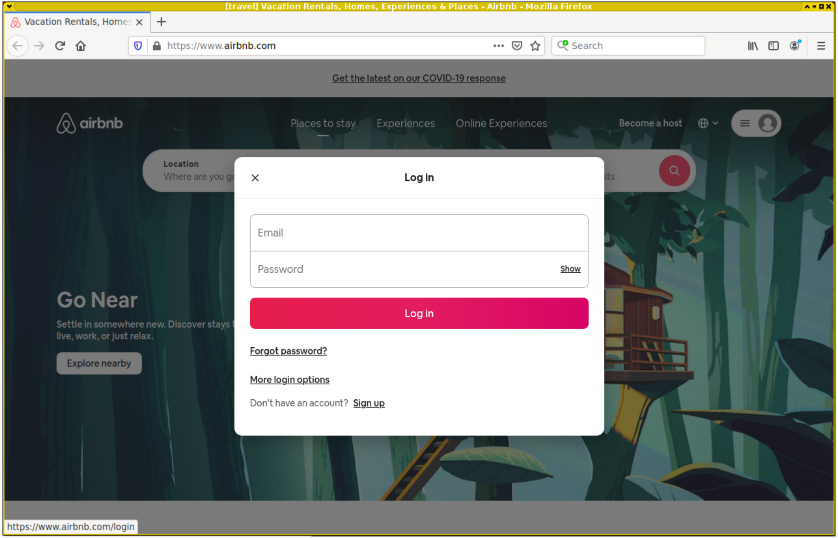
MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

NIST PQC: Ein Blick zurück und in die Zukunft

Peter Schwabe

February 21, 2023



Get the latest on our COVID-19 response

Go Near

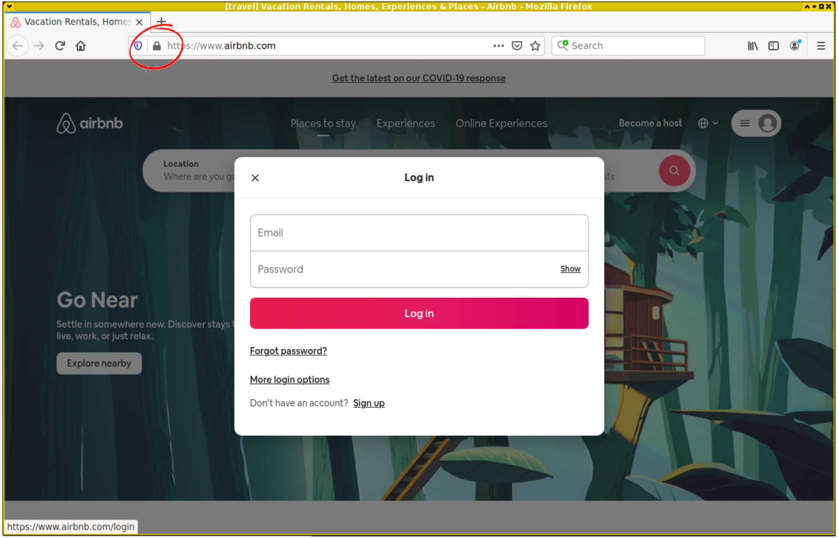
Settle in somewhere new. Discover stays t
live, work, or just relax.

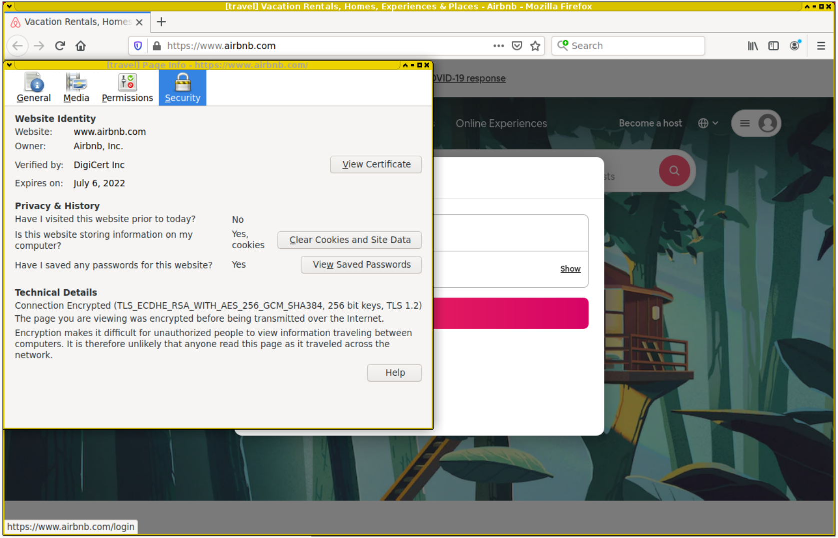
Log in

Forgot password?

More login options

Don't have an account? Sign up


[travel] Vacation Rentals, Homes, Experiences & Places - Airbnb - Mozilla Firefox
Vacation Rentals, Homes
https://www.airbnb.com
Search
Get the latest on our COVID-19 response
airbnb
Places to stay
Experiences
Online Experiences
Become a host
Location
Where are you g
Log in
Email
Password
Show
Log in
Forgot password?
More login options
Don't have an account? Sign up
Go Near
Settle in somewhere new. Discover stays t
live, work, or just relax.
Explore nearby
https://www.airbnb.com/login



Website Identity

Website:	www.airbnb.com
Owner:	Airbnb, Inc.
Verified by:	DigiCert Inc
Expires on:	July 6, 2022

Privacy & History

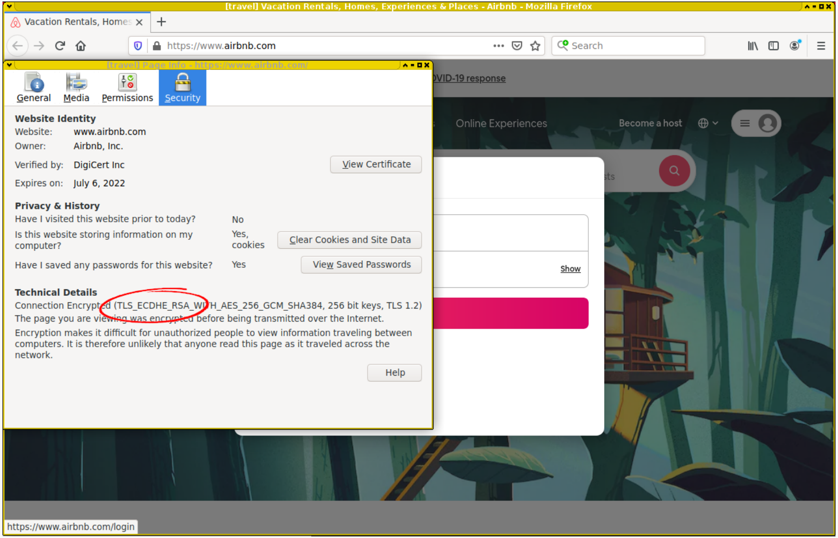
Have I visited this website prior to today?	No
Is this website storing information on my computer?	Yes, cookies
Have I saved any passwords for this website?	Yes

Technical Details

Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, 256 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.



Website Identity

Website: www.airbnb.com

Owner: Airbnb, Inc.

Verified by: DigiCert Inc

Expires on: July 6, 2022

Privacy & History

Have I visited this website prior to today? No

Is this website storing information on my computer? Yes, cookies

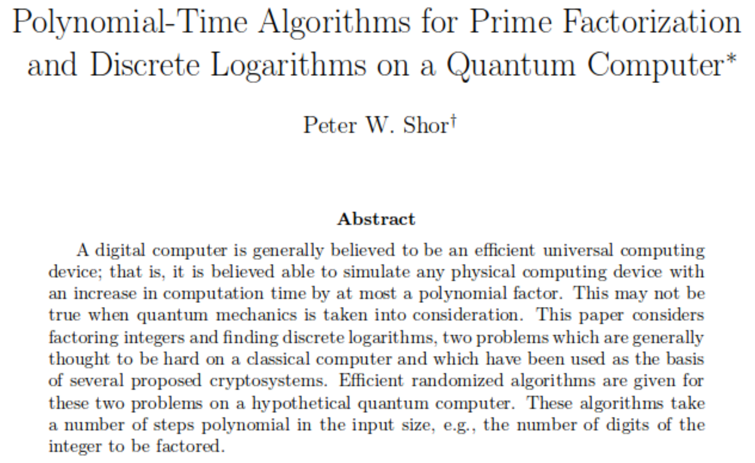
Have I saved any passwords for this website? Yes

Technical Details

Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, 256 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

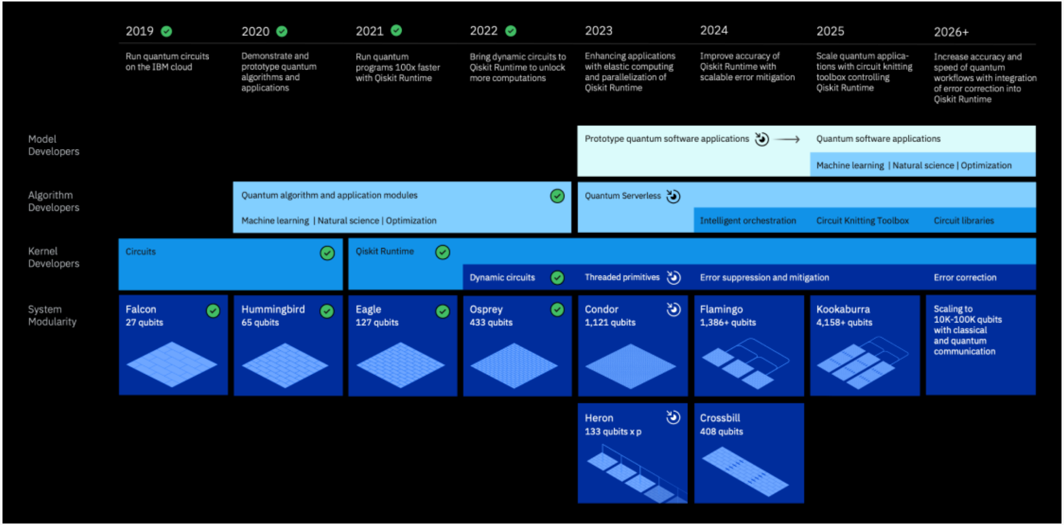


Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

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.



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

2026+ Increase accuracy and speed of quantum workflows with integration of error correction into Qiskit Runtime

See `https://www.ibm.com/quantum/roadmap`

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 (it's complicated. . .)

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
    - AES, running from 1997 to 2000
    - SHA3, running from 2007 to 2012

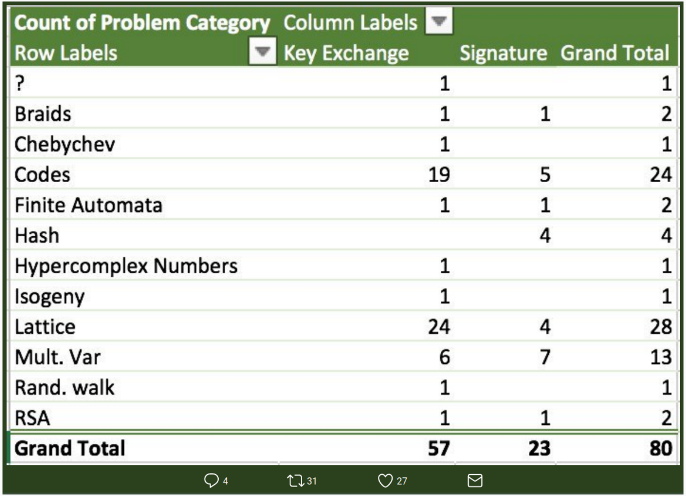
The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
    - AES, running from 1997 to 2000
    - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST

The NIST PQC “not-a-competition”

- Inspired by two earlier NIST crypto competitions:
  - AES, running from 1997 to 2000
  - SHA3, running from 2007 to 2012
- Approach: NIST specifies criteria, everybody is welcome to submit proposals
- Selection through an open process and multiple rounds
- Actual decisions are being made by NIST
- PQC project:
  - Announcement: Feb 2016
  - Call for proposals: Dec 2016 (based on community input)
  - Deadline for submissions: Nov 2017

The NIST competition: initial overview



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, Feb 2019

Encryption / Key agreement

- 9 lattice-based
- 7 code-based
- 1 isogeny-based

Signature schemes

- 3 lattice-based
- 2 symmetric-crypto based
- 4 MQ-based

The NIST competition: Jul 2020

Finalists

- 4 key-agreement schemes
  - 3 lattice-based
  - 1 code-based
- 3 signature schemes
  - 2 lattice-based
  - 1 MQ-based

Alternate schemes

- 5 key-agreement schemes
  - 2 lattice-based
  - 2 code-based
  - 1 isogeny-based
- 3 signature schemes
  - 2 symmetric-crypto based
  - 1 MQ-based

The NIST competition: Jul 2022

4 schemes selected for standardization

- CRYSTALS-Kyber: lattice-based key agreement
- CRYSTALS-Dilithium: lattice-based signature
- Falcon: lattice-based signature
- SPHINCS$^+$: hash-based signature

4 schemes advanced to round 4

- Classic McEliece: code-based key agreement
- BIKE: code-based key agreement
- HQC: code-based key agreement
- SIKE: isogeny-based key agreement († 30.07.2022)

The NIST competition: Jul 2022

4 schemes selected for standardization

- CRYSTALS-Kyber: lattice-based key agreement
- CRYSTALS-Dilithium: lattice-based signature
- Falcon: lattice-based signature
- SPHINCS$^+$: hash-based signature

4 schemes advanced to round 4

- Classic McEliece: code-based key agreement
- BIKE: code-based key agreement
- HQC: code-based key agreement
- SIKE: isogeny-based key agreement (✝ 30.07.2022)
- **Additionally:** call for more signature proposals

The SIKE shock

Castryck, Decru, 2022: *An efficient key recovery attack on SIDH*

The SIKE shock

Castryck, Decru, 2022: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)

The SIKE shock

Castryck, Decru, 2022: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
  (following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))

The SIKE shock

Castryck, Decru, 2022: *An efficient key recovery attack on SIDH*

- SIDH was “A decade unscathed” (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
  (following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))
- Competent, smart people tried to break it
  (e.g., Martindale, Panny: *How to not break SIDH* (ePrint 2019/558))

The SIKE shock

Castryck, Decru, 2022: *An efficient key recovery attack on SIDH*

- SIDH was "A decade unscathed" (Craig Costello, ePrint 2021/543)
- SIKE **lowered** parameters during NIST PQC
  (following Jaques, Schanck: *Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE* (ePrint 2019/103))
- Competent, smart people tried to break it
  (e.g., Martindale, Panny: *How to not break SIDH* (ePrint 2019/558))

Yet, **full break without any "warning"**

So, where are we?

> *“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

So, where are we?

Next steps for deployment

1. Take existing C/asm implementations of Kyber and Dilithium.
2. Integrate into systems and protocols.

Mission accomplished – The world is safe again!

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions
- **2008**: ***Rogue CA certificate*** using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions
- **2008**: ***Rogue CA certificate*** using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)
- **2012**: Flame malware exploits MD5 weaknesses

A bit of history: the case of MD5

- MD5 is a cryptographic hash function
- Hash functions are used as building blocks all over the place
- **1991**: MD5 is proposed by Rivest
- **1993**: Collisions in MD5 compression function (den Boer, Bosselaers)
- **1996**: Dobbertin, Bosselaers, Preneel: concerns about MD5
- **2004**: Wang presents MD5 collisions
- **2008**: ***Rogue CA certificate*** using MD5
  (Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger)
- **2012**: Flame malware exploits MD5 weaknesses

Replacing MD5 was "easy"!

Challenge 1: 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

Challenge 1: Performance

X25519 speed

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

X25519 sizes

- public key: 32 bytes

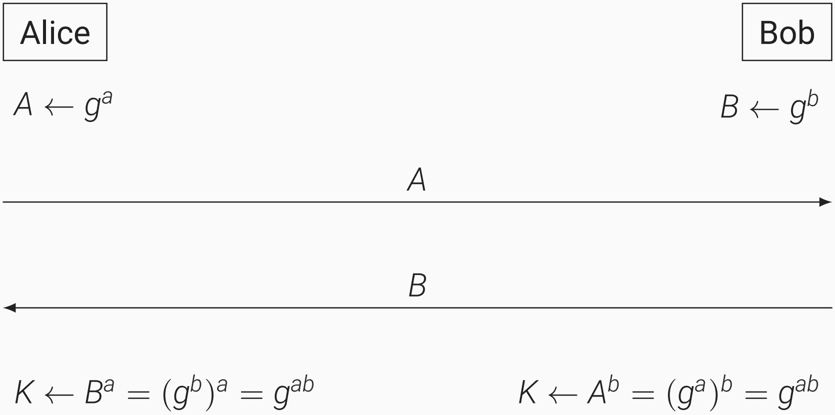
Kyber-768 speed

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

Kyber-768 sizes

- public key: 1184 bytes
- ciphertext: 1088 bytes

Challenge 2: A KEM is not DH!



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

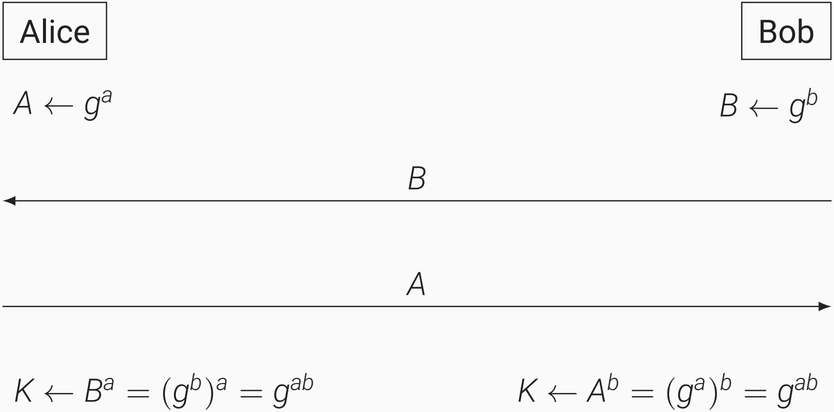
A

B

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

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

Challenge 2: A KEM is not DH!



Alice

Bob

$A \leftarrow g^a$

$B \leftarrow g^b$

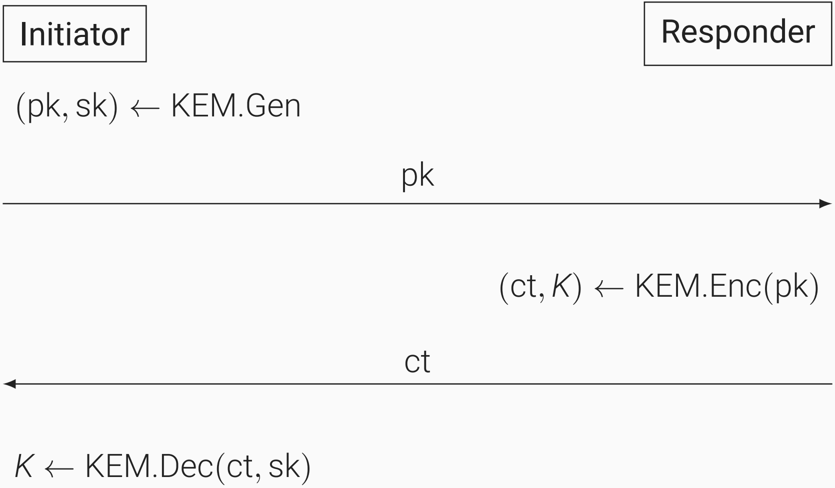
B

A

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

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

Challenge 2: A KEM is not DH!



Initiator

Responder

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

pk

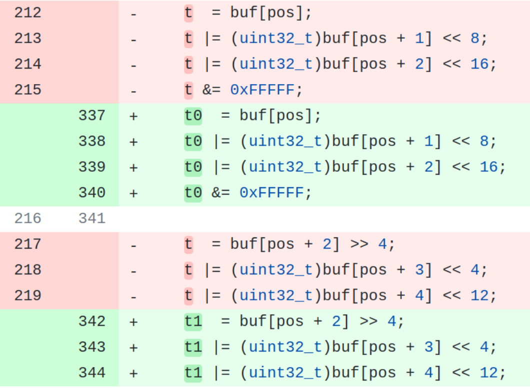
$(\mathrm{ct}, K) \leftarrow \mathrm{KEM.Enc}(\mathrm{pk})$

ct

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

Challenge 3: Bugs, bugs everywhere

Dilithium commit on Dec. 28, 2017

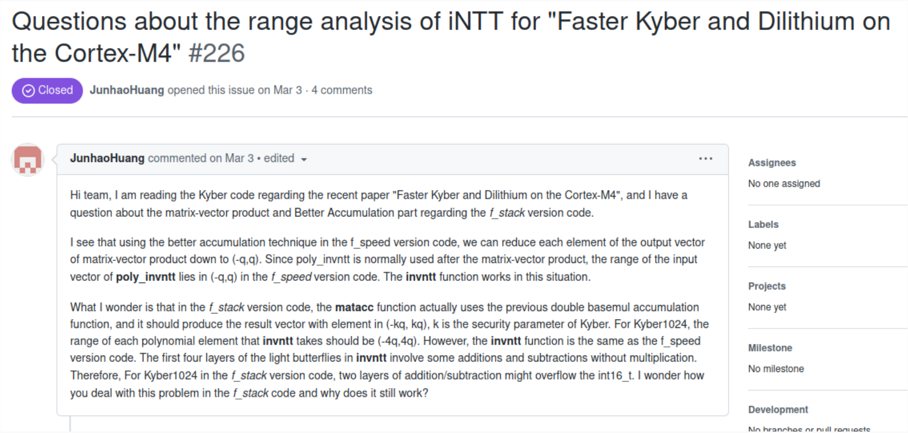


```
-    t  = buf[pos];
-    t |= (uint32_t)buf[pos + 1] << 8;
-    t |= (uint32_t)buf[pos + 2] << 16;
-    t &= 0xFFFFF;
+    t0  = buf[pos];
+    t0 |= (uint32_t)buf[pos + 1] << 8;
+    t0 |= (uint32_t)buf[pos + 2] << 16;
+    t0 &= 0xFFFFF;

-    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

Challenge 3: 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

Challenge 3: Bugs, bugs everywhere

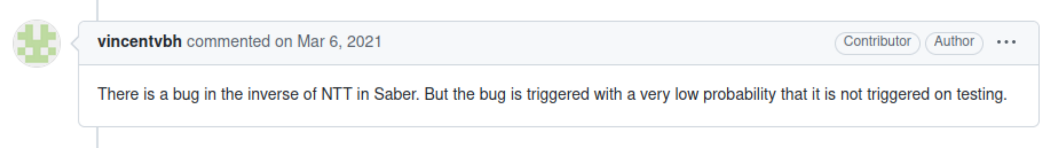
“. . . two layers of addition/subtraction might overflow the int16_t. I wonder how you deal with this problem in the f_stack code and why does it still work?”

Challenge 3: 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.”

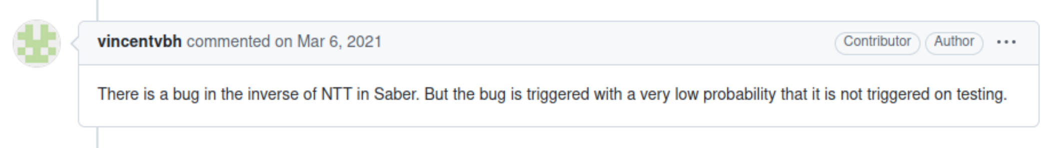
Challenge 3: Bugs, bugs everywhere



vincentvbh commented on Mar 6, 2021

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.

Challenge 3: Bugs, bugs everywhere



vincentvbh commented on Mar 6, 2021

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)

Challenge 3b: Bugs in proofs

“We note that a potential issue is that the security proof does not directly apply to Kyber itself, but rather to a modified version of the scheme which does not compress the public key.”

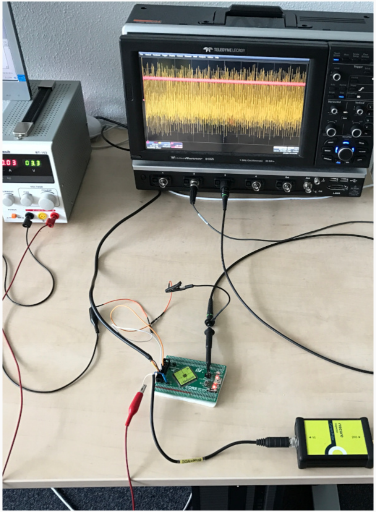
—NIST IR 8240

“In this comment, we would like to point out a flaw of existing security proofs of the SPHINCS+ hash-based scheme.”

—Mikhail Kudinov, Evgeniy Kiktenko, Aleksey Fedorov (July 2020)

Challenge 4: Implementation Security

Challenge 4: Implementation Security



- Attackers see more than input/output:
  - Power consumption
  - Electromagnetic radiation
  - Timing
- **Side-channel attacks:**
  - Measure information
  - Use to obtain secret data

Challenge 4: Side-channel countermeasures

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!

Challenge 4: Side-channel countermeasures

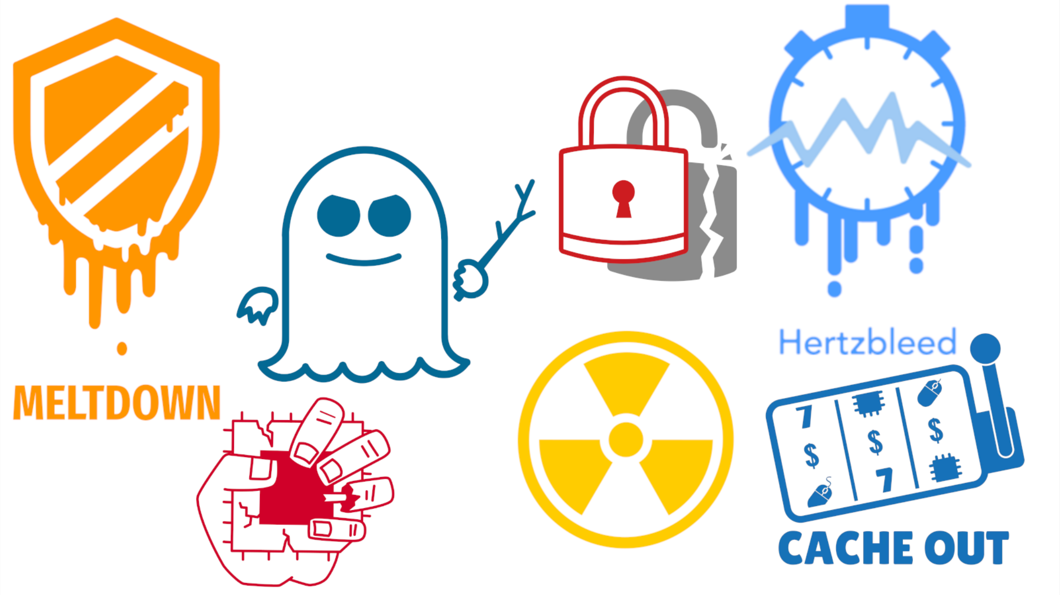
Hardware side-channels

- 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

Advanced microarchitectural attacks



High-assurance PQC

FORMOSA
CRYPTO

- **Form**ally verified **o**pen-**s**ource **a**mazing crypto
- Effort to formally verify crypto
- Currently three main projects:
  - EasyCrypt proof assistant
  - jasmin programming language
  - libjade (PQ-)crypto library
- Core community of ≈ 30–40 people
- Discussion forum with >100 people

University of BRISTOL

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

institute imdea software

U. PORTO

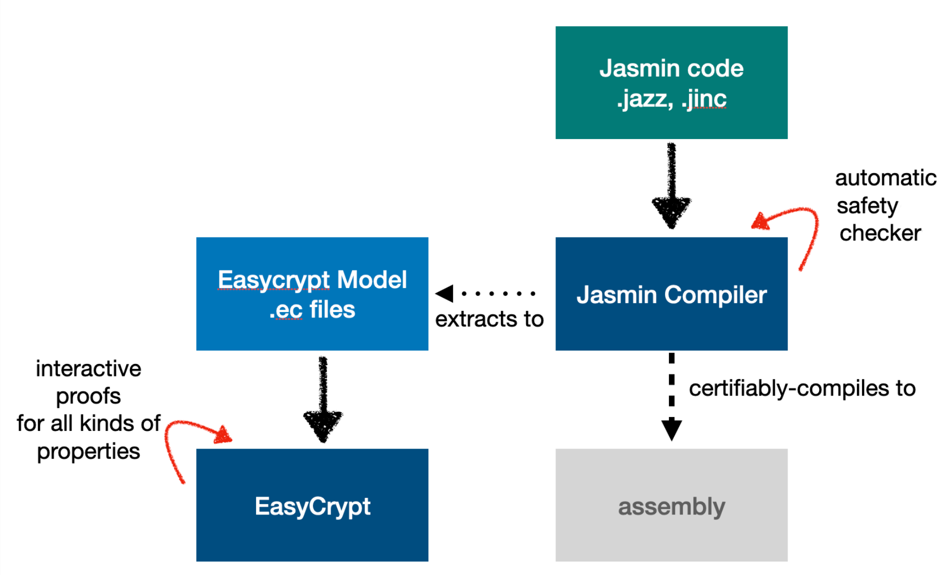
INESCTEC

Radboud University

Inria
INVENTEURS DU MONDE NUMÉRIQUE

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

The toolchain and workflow



Jasmin code
.jazz, .jinc

automatic
safety
checker

Easycrypt Model
.ec files

extracts to

Jasmin Compiler

interactive
proofs
for all kinds of
properties

certifiably-compiles to

EasyCrypt

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

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

libjade – Goals

First release of libjade

`https://github.com/formosa-crypto/libjade/releases/tag/v2022.12.0`
(big thanks to Tiago Oliveira!)

libjade – Goals

First release of libjade

`https://github.com/formosa-crypto/libjade/releases/tag/v2022.12.0`
(big thanks to Tiago Oliveira!)

Formally proven Kyber implementation

`https://eprint.iacr.org/2023/215` (Joint work with José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Antoine Séré, and Pierre-Yves Strub)

What if. . . ?

Had NIST required computer-verified software and proofs,

What if. . . ?

Had NIST required computer-verified software and proofs,

- we would have had way fewer bugs in PQC software;
- we would have much higher confidence in all proofs;

What if. . . ?

Had NIST required computer-verified software and proofs,

- we would have had way fewer bugs in PQC software;
- we would have much higher confidence in all proofs;
- attacks could be much more focused;

What if. . . ?

Had NIST required computer-verified software and proofs,

- we would have had way fewer bugs in PQC software;
- we would have much higher confidence in all proofs;
- attacks could be much more focused;

. . . and we would probably not have had a single submission.

Want to know more?

PQC resources

- NIST PQC website:
  `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`
- NIST mailing list:
  `https://csrc.nist.gov/projects/post-quantum-cryptography/email-list`
  `https://groups.google.com/a/list.nist.gov/g/pqc-forum`
- PQC Wiki: `https://pqc-wiki.fau.edu`

Formosa resources

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