

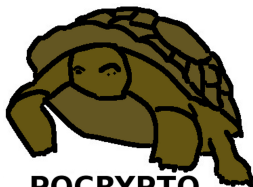
Post-quantum cryptography

Daniel J. Bernstein¹ Tanja Lange² Peter Schwabe³

University of Illinois at Chicago

Technische Universiteit Eindhoven

Radboud University



PQCRYPTO
ICT-645622

04 July 2017

Cryptography

- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.



- Literal meaning of cryptography: "secret writing".
- Achieves various security goals by secretly transforming messages.

https://www.iacr.org/

16 / 55

16 / 55


16 / 55


www.iacr.org

Your connection to this site is private.

Permissions

Connection


 The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server. [Certificate information](#)

 Your connection to www.iacr.org is encrypted using a modern cipher suite.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.

[What do these mean?](#)



2013

iacrmemHEREATiacr.org

1702

1245

457

Members

(1580 in 2012)

Regular+

Students



https://www.iacr.org/



https://www.iacr.org/docs/minutes/c2013mem-slides.pdf

IACR Mem

www.iacr.org



Your connection to this site is private.

Permissions

Connection



The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server.

[Certificate information](#)



Your connection to www.iacr.org is encrypted using a modern cipher suite.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.



[What do these mean?](#)

16 /

iacrm



Secret-key encryption



- ▶ Prerequisite: Alice and Bob share a secret key .
- ▶ Prerequisite: Eve doesn't know .
- ▶ Alice and Bob exchange any number of messages.
- ▶ Security goal #1: **Confidentiality** despite Eve's espionage.



Secret-key authenticated encryption



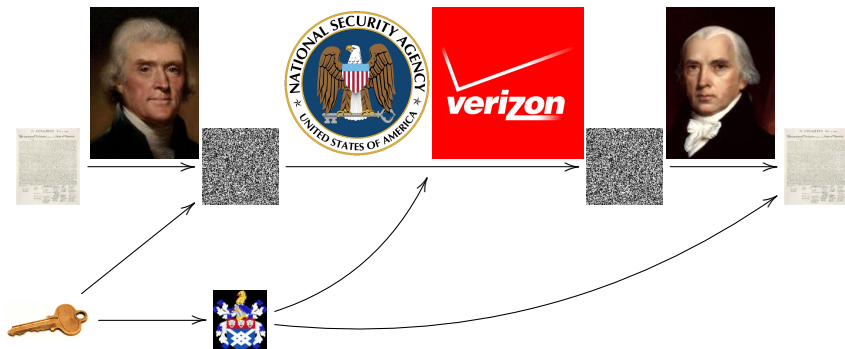
- ▶ Prerequisite: Alice and Bob share a secret key .
- ▶ Prerequisite: Eve doesn't know .
- ▶ Alice and Bob exchange any number of messages.
- ▶ Security goal #1: **Confidentiality** despite Eve's espionage.
- ▶ Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.





Secret-key authenticated encryption



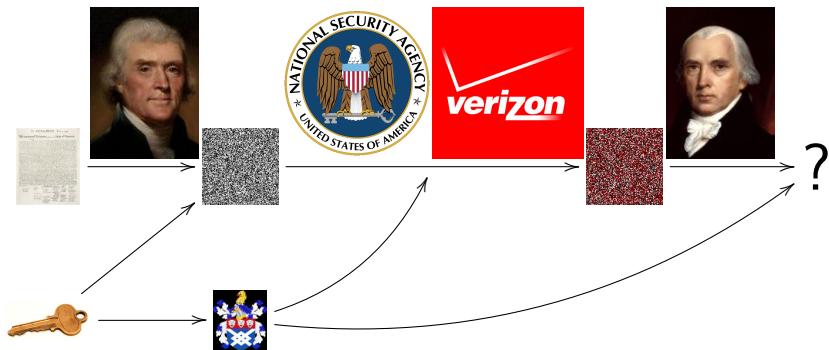
- ▶ Prerequisite: Alice and Bob share a secret key .
- ▶ Prerequisite: Eve doesn't know .
- ▶ Alice and Bob exchange any number of messages.
- ▶ Security goal #1: **Confidentiality** despite Eve's espionage.
- ▶ Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.





Public-key signatures



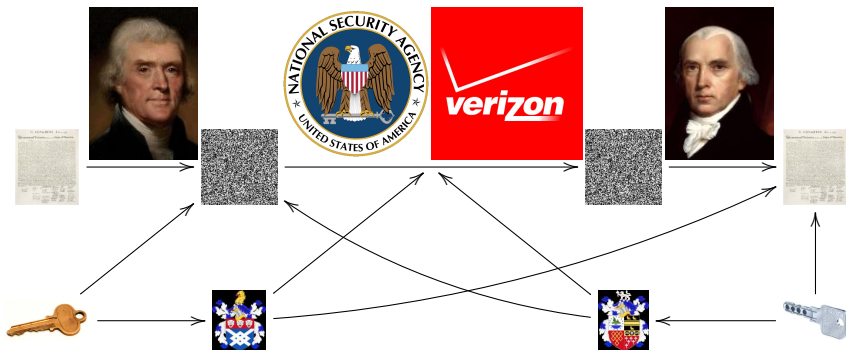
- ▶ Prerequisite: Alice has a secret key  and public key .
- ▶ Prerequisite: Eve doesn't know . Everyone knows .
- ▶ Alice publishes any number of messages.
- ▶ Security goal: Integrity.





Public-key signatures



- ▶ Prerequisite: Alice has a secret key  and public key .
- ▶ Prerequisite: Eve doesn't know . Everyone knows .
- ▶ Alice publishes any number of messages.
- ▶ Security goal: Integrity.

Public-key authenticated encryption (“DH” data flow)



- ▶ Prerequisite: Alice has a secret key  and public key .
- ▶ Prerequisite: Bob has a secret key  and public key .
- ▶ Alice and Bob exchange any number of messages.
- ▶ Security goal #1: Confidentiality.
- ▶ Security goal #2: Integrity.

Many more security goals studied in cryptography

- ▶ Protecting against denial of service.
- ▶ Stopping traffic analysis.
- ▶ Securely tallying votes.
- ▶ Searching encrypted data.
- ▶ Much more.

Attackers exploit physical reality

- ▶ 1996 Kocher: Typical crypto is broken by **side channels**.
- ▶ Response: Hundreds of papers on side-channel defenses.

Attackers exploit physical reality

- ▶ 1996 Kocher: Typical crypto is broken by **side channels**.
- ▶ Response: Hundreds of papers on side-channel defenses.
- ▶ Today's focus: Large universal **quantum computers**.
- ▶ Mark Ketchen, IBM Research, 2012, on quantum computing:
“We’re actually doing things that are making us think like, ‘hey this isn’t 50 years off, this is maybe just 10 years off, or 15 years off.’ It’s within reach.”
- ▶ Fast-forward to 2022, or 2027. Universal quantum computers exist.
- ▶ Shor’s algorithm solves in polynomial time:
 - ▶ Integer factorization. RSA is dead.
 - ▶ The discrete-logarithm problem in finite fields. DSA is dead.
 - ▶ The discrete-logarithm problem on elliptic curves. ECDHE is dead.
- ▶ This breaks all current public-key cryptography on the Internet!

Attackers exploit physical reality

- ▶ 1996 Kocher: Typical crypto is broken by **side channels**.
- ▶ Response: Hundreds of papers on side-channel defenses.
- ▶ Today's focus: Large universal **quantum computers**.
- ▶ Mark Ketchen, IBM Research, 2012, on quantum computing:
“We're actually doing things that are making us think like, ‘hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.’ It's within reach.”
- ▶ Fast-forward to 2022, or 2027. Universal quantum computers exist.
- ▶ Shor's algorithm solves in polynomial time:
 - ▶ Integer factorization. RSA is dead.
 - ▶ The discrete-logarithm problem in finite fields. DSA is dead.
 - ▶ The discrete-logarithm problem on elliptic curves. ECDHE is dead.
- ▶ This breaks all current public-key cryptography on the Internet!
- ▶ Also, Grover's algorithm speeds up brute-force searches.
- ▶ Example: Only 2^{64} quantum operations to break AES-128;
 2^{128} quantum operations to break AES-256.



Physical cryptography: a return to the dark ages

- ▶ Imagine a lockable-briefcase salesman proposing a “locked-briefcase Internet” using “provably secure locked-briefcase cryptography”:
 - ▶ Alice puts secret information into a lockable briefcase.
 - ▶ Alice locks the briefcase.
 - ▶ A courier transports the briefcase from Alice to Bob.
 - ▶ Bob unlocks the briefcase and retrieves the information.
 - ▶ There is a mathematical proof that the information is hidden!
 - ▶ Throw away algorithmic cryptography!



Physical cryptography: a return to the dark ages

- ▶ Imagine a lockable-briefcase salesman proposing a “locked-briefcase Internet” using “provably secure locked-briefcase cryptography”:
 - ▶ Alice puts secret information into a lockable briefcase.
 - ▶ Alice locks the briefcase.
 - ▶ A courier transports the briefcase from Alice to Bob.
 - ▶ Bob unlocks the briefcase and retrieves the information.
 - ▶ There is a mathematical proof that the information is hidden!
 - ▶ Throw away algorithmic cryptography!
- ▶ Most common reactions from security experts:
 - ▶ This would make security much worse.



Physical cryptography: a return to the dark ages

- ▶ Imagine a lockable-briefcase salesman proposing a “locked-briefcase Internet” using “provably secure locked-briefcase cryptography”:
 - ▶ Alice puts secret information into a lockable briefcase.
 - ▶ Alice locks the briefcase.
 - ▶ A courier transports the briefcase from Alice to Bob.
 - ▶ Bob unlocks the briefcase and retrieves the information.
 - ▶ There is a mathematical proof that the information is hidden!
 - ▶ Throw away algorithmic cryptography!
- ▶ Most common reactions from security experts:
 - ▶ This would make security much worse.
 - ▶ This would be insanely expensive.



Physical cryptography: a return to the dark ages

- ▶ Imagine a lockable-briefcase salesman proposing a “locked-briefcase Internet” using “provably secure locked-briefcase cryptography”:
 - ▶ Alice puts secret information into a lockable briefcase.
 - ▶ Alice locks the briefcase.
 - ▶ A courier transports the briefcase from Alice to Bob.
 - ▶ Bob unlocks the briefcase and retrieves the information.
 - ▶ There is a mathematical proof that the information is hidden!
 - ▶ Throw away algorithmic cryptography!
- ▶ Most common reactions from security experts:
 - ▶ This would make security much worse.
 - ▶ This would be insanely expensive.
 - ▶ We should not dignify this proposal with a response.



Security advantages of algorithmic cryptography

- ▶ Keep secrets heavily shielded inside authorized computers.
- ▶ Reduce trust in third parties:
 - ▶ Reduce reliance on closed-source software and hardware.
 - ▶ Increase comprehensiveness of audits.
 - ▶ Increase comprehensiveness of formal verification.
 - ▶ Design systems to be secure even if **keys are public**.
Critical example: **signed** software updates.
- ▶ Understand security as thoroughly as possible:
 - ▶ Publish comprehensive specifications.
 - ▶ Build large research community with clear security goals.
 - ▶ Publicly document attack efforts.
 - ▶ Require systems to convincingly survive many years of analysis.

Confidence-inspiring crypto takes time to build

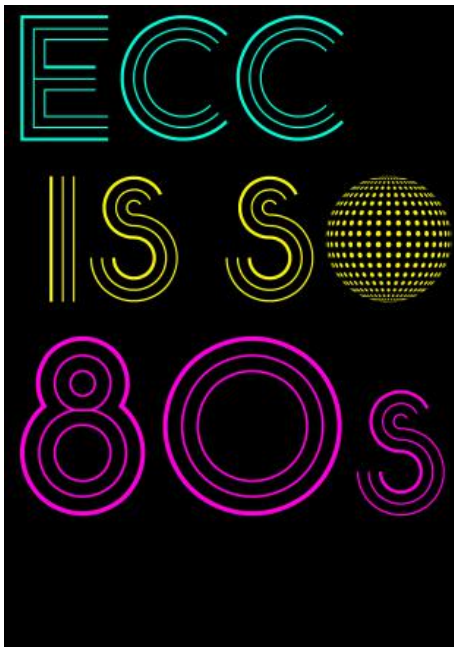
- ▶ Many stages of research from cryptographic design to deployment:
 - ▶ Explore space of cryptosystems.
 - ▶ Study algorithms for the attackers.
 - ▶ Focus on secure cryptosystems.

Confidence-inspiring crypto takes time to build

- ▶ Many stages of research from cryptographic design to deployment:
 - ▶ Explore space of cryptosystems.
 - ▶ Study algorithms for the attackers.
 - ▶ Focus on secure cryptosystems.
 - ▶ Study algorithms for the users.
 - ▶ Study implementations on real hardware.
 - ▶ Study side-channel attacks, fault attacks, etc.
 - ▶ Focus on secure, reliable implementations.
 - ▶ Focus on implementations meeting performance requirements.
 - ▶ Integrate securely into real-world applications.

Confidence-inspiring crypto takes time to build

- ▶ Many stages of research from cryptographic design to deployment:
 - ▶ Explore space of cryptosystems.
 - ▶ Study algorithms for the attackers.
 - ▶ Focus on secure cryptosystems.
 - ▶ Study algorithms for the users.
 - ▶ Study implementations on real hardware.
 - ▶ Study side-channel attacks, fault attacks, etc.
 - ▶ Focus on secure, reliable implementations.
 - ▶ Focus on implementations meeting performance requirements.
 - ▶ Integrate securely into real-world applications.
- ▶ Example: ECC introduced **1985**; big advantages over RSA. Robust ECC started to take over the Internet in **2015**.
- ▶ Can't wait for quantum computers before finding a solution!



Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- ▶ PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- ▶ PQCrypto 2008, PQCrypto 2010, PQCrypto 2011, PQCrypto 2013.
- ▶ 2014 EU publishes H2020 call including post-quantum crypto as topic.
- ▶ PQCrypto 2014.
- ▶ April 2015 NIST hosts first workshop on post-quantum cryptography.
- ▶ August 2015 NSA wakes up ...





NSA announcements

August 11, 2015

IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.

NSA announcements

August 11, 2015

IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.

August 19, 2015

IAD will initiate a transition to quantum resistant algorithms in the not too distant future.

NSA comes late to the party and botches its grand entrance.



NSA announcements

August 11, 2015

IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.

August 19, 2015

IAD will initiate a transition to quantum resistant algorithms in the not too distant future.

NSA comes late to the party and botches its grand entrance.



Post-quantum becoming mainstream

- ▶ PQCrypto 2016: 22–26 Feb in Fukuoka, Japan, with more than 200 participants



- ▶ PQCrypto 2017 took place last week in Utrecht, Netherlands.
- ▶ NIST is calling for post-quantum proposals: 5-year competition.

Post-Quantum Cryptography for Long-term Security

- ▶ Project funded by EU in Horizon 2020, running 2015 – 2018.
- ▶ 11 partners from academia and industry, TU/e is coordinator



Radboud Universiteit



TECHNISCHE
UNIVERSITÄT
DARMSTADT



University of Haifa



Work packages

PQCRYPTO is designing a portfolio of high-security post-quantum public-key systems, and will improve the speed of these systems, adapting to the different performance challenges of mobile devices, the cloud, and the Internet.

Technical work packages

- ▶ WP1: Post-quantum cryptography for small devices
Leader: Tim Güneysu, co-leader: Peter Schwabe
- ▶ WP2: Post-quantum cryptography for the Internet
Leader: Daniel J. Bernstein, co-leader: Bart Preneel
- ▶ WP3: Post-quantum cryptography for the cloud
Leader: Nicolas Sendrier, co-leader: Christian Rechberger

Non-technical work packages

- ▶ WP4: Management and dissemination
Leader: Tanja Lange
- ▶ WP5: Standardization
Leader: Walter Fumy



Initial recommendations of long-term secure post-quantum systems

Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos,
Johannes Buchmann, Wouter Castryck, Orr Dunkelman,
Tim Güneysu, Shay Gueron, Andreas Hülsing,
Tanja Lange, Mohamed Saied Emam Mohamed,
Christian Rechberger, Peter Schwabe, Nicolas Sendrier,
Frederik Vercauteren, Bo-Yin Yang

Initial recommendations

- ▶ **Symmetric encryption** Thoroughly analyzed, 256-bit keys:

- ▶ AES-256
- ▶ Salsa20 with a 256-bit key

Evaluating: Serpent-256, ...

- ▶ **Symmetric authentication** Information-theoretic MACs:

- ▶ GCM using a 96-bit nonce and a 128-bit authenticator
- ▶ Poly1305

- ▶ **Public-key encryption** McEliece with binary Goppa codes:

- ▶ length $n = 6960$, dimension $k = 5413$, $t = 119$ errors

Evaluating: QC-MDPC, Stehlé-Steinfeld NTRU, ...

- ▶ **Public-key signatures** Hash-based (minimal assumptions):

- ▶ XMSS with any of the parameters specified in CFRG draft
- ▶ SPHINCS-256

Evaluating: HFEv-, ...



Hash-based signatures

- ▶ Secret key s , public key p .
- ▶ Only one prerequisite: a good hash function, e.g. SHA3-512, ...
Hash functions map long strings to fixed-length strings.
Signature schemes use hash functions in handling m .
- ▶ Old idea: 1979 Lamport one-time signatures.
- ▶ 1979 Merkle extends to more signatures.
- ▶ Many further improvements.
- ▶ Security thoroughly analyzed.

Signatures for 1-bit messages

Key generation

- ▶ Generate 256-bit random values $(r_0, r_1) = s$ (secret key)
- ▶ Compute $(h(r_0), h(r_1)) = (p_0, p_1) = p$ (public key)

Signatures for 1-bit messages

Key generation

- ▶ Generate 256-bit random values $(r_0, r_1) = s$ (secret key)
- ▶ Compute $(h(r_0), h(r_1)) = (p_0, p_1) = p$ (public key)

Signing

- ▶ Signature for message $b = 0$: $\sigma = r_0$
- ▶ Signature for message $b = 1$: $\sigma = r_1$



Signatures for 1-bit messages

Key generation

- ▶ Generate 256-bit random values $(r_0, r_1) = s$ (secret key)
- ▶ Compute $(h(r_0), h(r_1)) = (p_0, p_1) = p$ (public key)

Signing

- ▶ Signature for message $b = 0$: $\sigma = r_0$
- ▶ Signature for message $b = 1$: $\sigma = r_1$

Verification

Check that $h(\sigma) = p_b$

One-time signatures for 2-bit messages

Key generation

- Generate 256-bit random values $(r_{0,0}, r_{0,1}, r_{1,0}, r_{1,1}) = s$
- Compute $(h(r_{0,0}), h(r_{0,1}), h(r_{1,0}), h(r_{1,1})) = (p_{0,0}, p_{0,1}, p_{1,0}, p_{1,1}) = p$

One-time signatures for 2-bit messages

Key generation

- ▶ Generate 256-bit random values $(r_{0,0}, r_{0,1}, r_{1,0}, r_{1,1}) = s$
- ▶ Compute $(h(r_{0,0}), h(r_{0,1}), h(r_{1,0}), h(r_{1,1})) = (p_{0,0}, p_{0,1}, p_{1,0}, p_{1,1}) = p$

Signing

- ▶ Signature for message (b_0, b_1) : $\sigma = (\sigma_0, \sigma_1) = (r_{0,b_0}, r_{1,b_1})$

One-time signatures for 2-bit messages

Key generation

- ▶ Generate 256-bit random values $(r_{0,0}, r_{0,1}, r_{1,0}, r_{1,1}) = s$
- ▶ Compute $(h(r_{0,0}), h(r_{0,1}), h(r_{1,0}), h(r_{1,1})) = (p_{0,0}, p_{0,1}, p_{1,0}, p_{1,1}) = p$

Signing

- ▶ Signature for message (b_0, b_1) : $\sigma = (\sigma_0, \sigma_1) = (r_{0,b_0}, r_{1,b_1})$

Verification

- ▶ Check that $h(\sigma_0) = p_{0,b_0}$
- ▶ Check that $h(\sigma_1) = p_{1,b_1}$



One-time signatures for 256-bit messages

Key generation

- ▶ Generate 256-bit random values $s = (r_{0,0}, r_{0,1} \dots, r_{255,0}, r_{255,1})$
- ▶ Compute $p = (h(r_{0,0}), h(r_{0,1}), \dots, h(r_{255,0}), h(r_{255,1})) = (p_{0,0}, p_{0,1}, \dots, p_{255,0}, p_{255,1})$

One-time signatures for 256-bit messages

Key generation

- ▶ Generate 256-bit random values $s = (r_{0,0}, r_{0,1} \dots, r_{255,0}, r_{255,1})$
- ▶ Compute $p = (h(r_{0,0}), h(r_{0,1}), \dots, h(r_{255,0}), h(r_{255,1})) = (p_{0,0}, p_{0,1}, \dots, p_{255,0}, p_{255,1})$

Signing

- ▶ Signature for message (b_0, \dots, b_{255}) :
 $\sigma = (\sigma_0, \dots, \sigma_{255}) = (r_{0,b_0}, \dots, r_{255,b_{255}})$

One-time signatures for 256-bit messages

Key generation

- ▶ Generate 256-bit random values $s = (r_{0,0}, r_{0,1} \dots, r_{255,0}, r_{255,1})$
- ▶ Compute $p = (h(r_{0,0}), h(r_{0,1}), \dots, h(r_{255,0}), h(r_{255,1})) = (p_{0,0}, p_{0,1}, \dots, p_{255,0}, p_{255,1})$

Signing

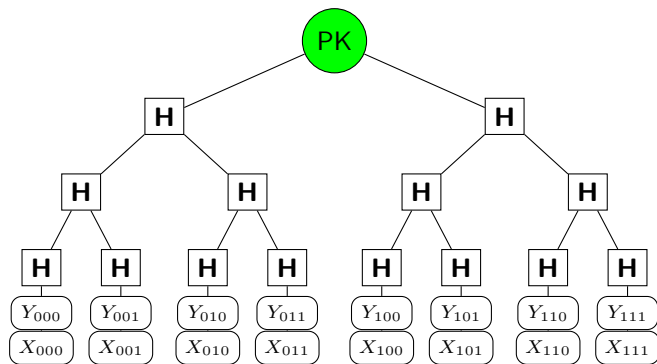
- ▶ Signature for message (b_0, \dots, b_{255}) :
 $\sigma = (\sigma_0, \dots, \sigma_{255}) = (r_{0,b_0}, \dots, r_{255,b_{255}})$

Verification

- ▶ Check that $h(\sigma_0) = p_{0,b_0}$
- ▶ ...
- ▶ Check that $h(\sigma_{255}) = p_{255,b_{255}}$

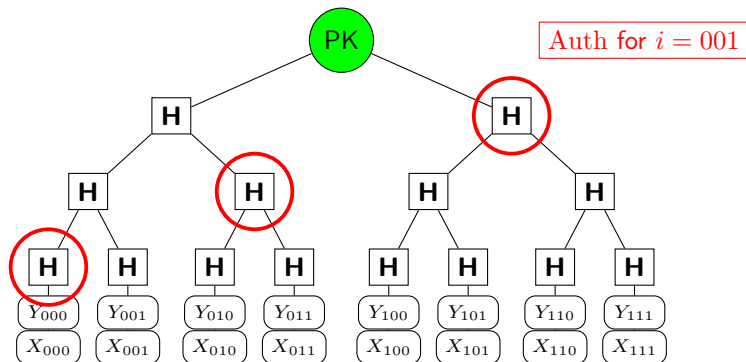


Merkle Trees



- Merkle, 1979: Leverage one-time signatures to multiple messages
- Binary hash tree on top of OTS public keys

Merkle Trees



- Use OTS keys sequentially
- $SIG = (i, \text{sign}(M, X_i), Y_i, \text{Auth})$
- Need to remember current *index* (\Rightarrow stateful scheme)



XMSS-T

- ▶ State of the art of (stateful) hash-based signatures: XMSS-T
- ▶ Many improvements to the “simple” Merkle-tree construction
- ▶ Currently being adopted by CFRG (IETF)
- ▶ Performance for 128-bit post-quantum security:
 - ▶ Public key: 64 bytes
 - ▶ Secret key: 2.2 KB
 - ▶ Signature: 2.9 KB
 - ▶ Signing: 10ms (Intel Core i7 3.5GHz)
- ▶ Speed is from a C implementation using OpenSSL

XMSS-T

- ▶ State of the art of (stateful) hash-based signatures: XMSS-T
- ▶ Many improvements to the “simple” Merkle-tree construction
- ▶ Currently being adopted by CFRG (IETF)
- ▶ Performance for 128-bit post-quantum security:
 - ▶ Public key: 64 bytes
 - ▶ Secret key: 2.2 KB
 - ▶ Signature: 2.9 KB
 - ▶ Signing: 10ms (Intel Core i7 3.5GHz)
- ▶ Speed is from a C implementation using OpenSSL
- ▶ Common pattern for post-quantum crypto:
 - ▶ Not necessarily (much) slower than, say, ECC
 - ▶ Considerably larger keys, signatures, ciphertexts, ...

About the state

- Used for *security*:
Stores index $i \Rightarrow$ Prevents using one-time keys twice.
- Used for *efficiency*:
Stores intermediate results for fast Auth computation.

About the state

- ▶ Used for *security*:
Stores index $i \Rightarrow$ Prevents using one-time keys twice.
- ▶ Used for *efficiency*:
Stores intermediate results for fast Auth computation.
- ▶ Problems:
 - ▶ Load-balancing
 - ▶ Multi-threading
 - ▶ Backups
 - ▶ Virtual-machine images
 - ▶ ...

About the state

- ▶ Used for *security*:
Stores index $i \Rightarrow$ Prevents using one-time keys twice.
- ▶ Used for *efficiency*:
Stores intermediate results for fast Auth computation.
- ▶ Problems:
 - ▶ Load-balancing
 - ▶ Multi-threading
 - ▶ Backups
 - ▶ Virtual-machine images
 - ▶ ...
- ▶ This is not even compatible with the *definition* of cryptographic signatures
- ▶ “Huge foot-cannon” (Adam Langley, Google)



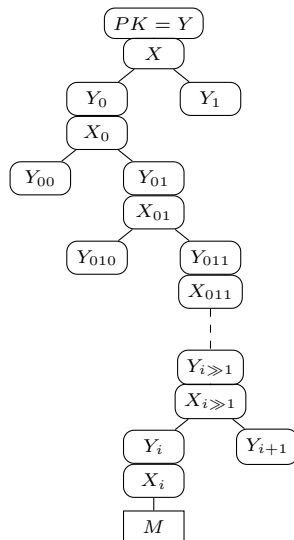
About the state

- ▶ Used for *security*:
Stores index $i \Rightarrow$ Prevents using one-time keys twice.
- ▶ Used for *efficiency*:
Stores intermediate results for fast Auth computation.
- ▶ Problems:
 - ▶ Load-balancing
 - ▶ Multi-threading
 - ▶ Backups
 - ▶ Virtual-machine images
 - ▶ ...
- ▶ This is not even compatible with the *definition* of cryptographic signatures
- ▶ “Huge foot-cannon” (Adam Langley, Google)
- ▶ Question: can we get rid of the state?



Stateless hash-based signatures

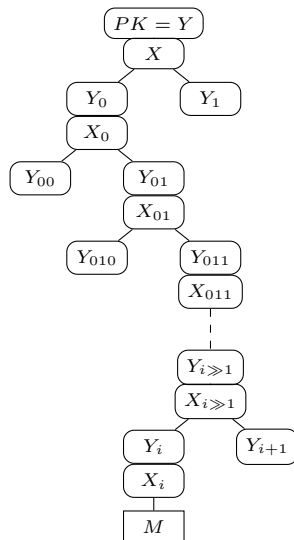
Goldreich's approach: binary tree as in Merkle, but:



Stateless hash-based signatures

Goldreich's approach: binary tree as in Merkle, but:

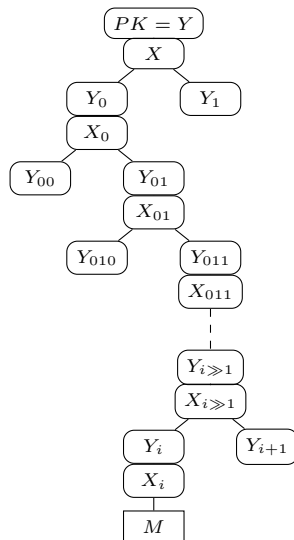
- For security
 - pick index i *at random*;
 - requires huge tree to avoid index collisions (e.g., height $h = 256$).



Stateless hash-based signatures

Goldreich's approach: binary tree as in Merkle, but:

- ▶ For security
 - ▶ pick index i *at random*;
 - ▶ requires huge tree to avoid index collisions (e.g., height $h = 256$).
- ▶ For efficiency:
 - ▶ use binary *certification tree* of OTS;
 - ▶ all OTS secret keys are generated pseudorandomly.



It works, but signatures are painfully long

- ▶ 0.6 MB for Goldreich signature using short-public-key Winternitz-16 one-time signatures.
- ▶ Would dominate traffic in typical applications, and add user-visible latency on typical network connections.

It works, but signatures are painfully long

- ▶ 0.6 MB for Goldreich signature using short-public-key Winternitz-16 one-time signatures.
- ▶ Would dominate traffic in typical applications, and add user-visible latency on typical network connections.
- ▶ Example:
 - ▶ Debian operating system is designed for frequent upgrades.
 - ▶ At least one new signature for each upgrade.
 - ▶ Typical upgrade: one package or just a few packages.
 - ▶ 1.2 MB average package size.
 - ▶ 0.08 MB median package size.



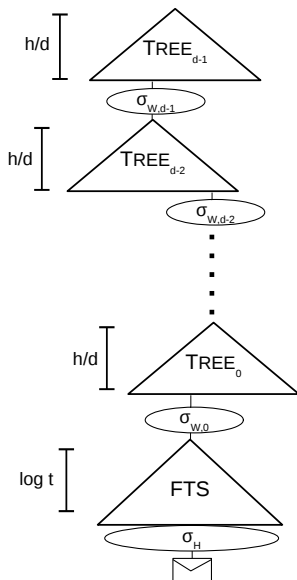
It works, but signatures are painfully long

- ▶ 0.6 MB for Goldreich signature using short-public-key Winternitz-16 one-time signatures.
- ▶ Would dominate traffic in typical applications, and add user-visible latency on typical network connections.
- ▶ Example:
 - ▶ Debian operating system is designed for frequent upgrades.
 - ▶ At least one new signature for each upgrade.
 - ▶ Typical upgrade: one package or just a few packages.
 - ▶ 1.2 MB average package size.
 - ▶ 0.08 MB median package size.
- ▶ Example:
 - ▶ HTTPS typically sends multiple signatures per page.
 - ▶ 1.8 MB average web page in Alexa Top 1000000.



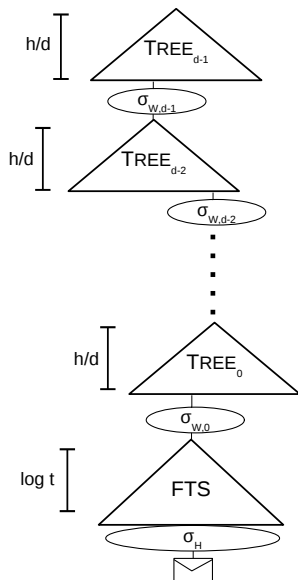
The SPHINCS approach

- ▶ Paper by Bernstein, Hopwood, Hülsing, Lange, Niederhagen, Papachristodoulou, Schneider, Schwabe, Wilcox-O'Hearn at Eurocrypt 2015.
- ▶ Use a “hyper-tree” of total height h
- ▶ Parameter $d \geq 1$, such that $d \mid h$
- ▶ Each (Merkle) tree has height h/d
- ▶ (h/d) -ary certification tree



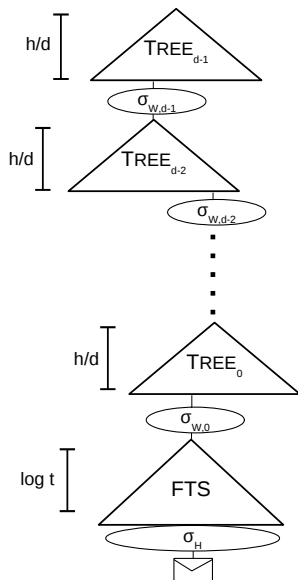
The SPHINCS approach

- Pick index (pseudo-)randomly
- Messages signed with *few-time* signature scheme
- Significantly reduce total tree height
- Require $\Pr[r\text{-times Coll}] \cdot \Pr[\text{Forgery after } r \text{ signatures}] = \text{negl}(n)$



The SPHINCS approach

- ▶ SPHINCS-256 for 128-bit post-quantum security
- ▶ 12 trees of height 5 each
- ▶ 256-bit hashes in OTS and FTS



SPHINCS-256 speed and sizes

SPHINCS-256 sizes

- ▶ ≈ 41 KB signature ($\approx 15\times$ smaller than Goldreich!)
- ▶ ≈ 1 KB public key
- ▶ ≈ 1 KB private key

SPHINCS-256 speed and sizes

SPHINCS-256 sizes

- ▶ ≈ 41 KB signature ($\approx 15\times$ smaller than Goldreich!)
- ▶ ≈ 1 KB public key
- ▶ ≈ 1 KB private key

High-speed implementation

- ▶ Target Intel Haswell with 256-bit AVX2 vector instructions
- ▶ Use $8\times$ parallel hashing, vectorize on high level
- ▶ ≈ 1.6 cycles/byte for custom high-performance hash

SPHINCS-256 speed and sizes

SPHINCS-256 sizes

- ▶ ≈ 41 KB signature ($\approx 15\times$ smaller than Goldreich!)
- ▶ ≈ 1 KB public key
- ▶ ≈ 1 KB private key

High-speed implementation

- ▶ Target Intel Haswell with 256-bit AVX2 vector instructions
- ▶ Use $8\times$ parallel hashing, vectorize on high level
- ▶ ≈ 1.6 cycles/byte for custom high-performance hash

SPHINCS-256 speed

- ▶ Signing: < 52 Mio. Haswell cycles (> 200 sigs/sec, 4 Core, 3GHz)
- ▶ Verification: < 1.5 Mio. Haswell cycles
- ▶ Keygen: < 3.3 Mio. Haswell cycles
- ▶ “Fair comparison” to XMSS-T: slowdown of $30\times$



More information

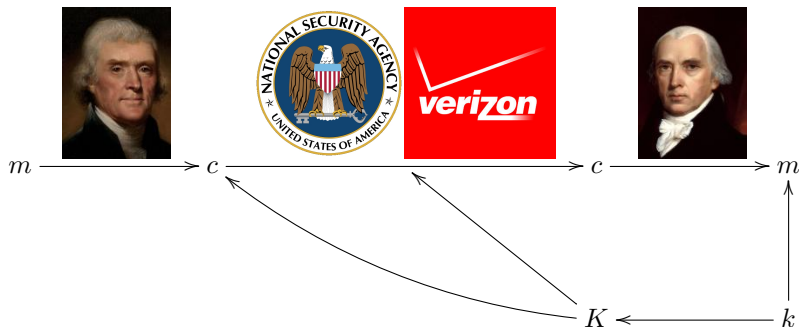
<https://sphincs.cr.yp.to/>

Post-quantum secret-key authenticated encryption



- ▶ Very easy solutions if secret key k is long uniform random string:
 - ▶ “One-time pad” for encryption.
 - ▶ “Wegman–Carter MAC” for authentication.
- ▶ AES-256: Standardized method to expand 256-bit k into string indistinguishable from long k .
- ▶ AES introduced in 1998 by Daemen and Rijmen. Security analyzed in papers by dozens of cryptanalysts.
- ▶ No credible threat from quantum algorithms. Grover costs 2^{128} .

Post-quantum public-key encryption: code-based



- ▶ Alice uses Bob's public key K to encrypt.
- ▶ Bob uses his secret key k to decrypt.
- ▶ Code-based crypto proposed by McEliece in 1978.
- ▶ Almost as old as RSA, but much stronger security history.
- ▶ Many further improvements.

Even higher urgency for long-term confidentiality

- ▶ Attacker can break currently used encryption (ECC, RSA) with a quantum computer.
- ▶ Even worse, today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. All data can be recovered in clear from recording traffic and breaking the public key scheme.
- ▶ How many years are you required to keep your data secret? From whom?



- ▶ Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement

Even higher urgency for long-term confidentiality

- ▶ Attacker can break currently used encryption (ECC, RSA) with a quantum computer.
- ▶ Even worse, today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. All data can be recovered in clear from recording traffic and breaking the public key scheme.
- ▶ How many years are you required to keep your data secret? From whom?



- ▶ Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement ... and an important function of signatures is to protect operating system upgrades.
- ▶ Protect your upgrades *now* with post-quantum signatures.

Error correction

- ▶ Digital media is exposed to memory corruption.
- ▶ Many systems check whether data was corrupted in transit:
 - ▶ ISBN numbers have check digit to detect corruption.
 - ▶ ECC RAM detects up to two errors and can correct one error.
64 bits are stored as 72 bits: extra 8 bits for checks and recovery.
- ▶ In general, k bits of data get stored in n bits, adding some redundancy.
- ▶ If no error occurred, these n bits satisfy $n - k$ parity check equations; else can correct errors from the error pattern.
- ▶ Good codes can correct many errors without blowing up storage too much;
offer guarantee to correct t errors (often can correct or at least detect more).
- ▶ To represent these check equations we need a matrix.





Hamming code

Parity check matrix ($n = 7, k = 4$):

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{cccccccl} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., 1, 0, 1 means



Hamming code

Parity check matrix ($n = 7, k = 4$):

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{cccccccl} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., 1, 0, 1 means b_4 flipped.

Hamming code

Parity check matrix ($n = 7, k = 4$):

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{cccccccl} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.

Failure pattern uniquely identifies the error location,

e.g., 1, 0, 1 means b_4 flipped.

In math notation, the failure pattern is $H \cdot \mathbf{b}$.

Coding theory

- ▶ Names: code word \mathbf{c} , error vector \mathbf{e} , received word $\mathbf{b} = \mathbf{c} + \mathbf{e}$.
- ▶ Very common to transform the matrix so that the left part has just 1 on the diagonal (no need to store that part).

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

- ▶ Many special constructions discovered in 65 years of coding theory:
 - ▶ Large matrix H .
 - ▶ Fast decoding algorithm to find \mathbf{e} given $\mathbf{s} = H \cdot (\mathbf{c} + \mathbf{e})$, whenever \mathbf{e} doesn't have too many bits set.
- ▶ Given large H , usually very hard to find fast decoding algorithm.
- ▶ Use this difference in complexities for encryption.



Code-based encryption

- ▶ 1971 Goppa: Fast decoders for many matrices H .
- ▶ 1978 McEliece: Use Goppa codes for public-key cryptography.
 - ▶ Original parameters designed for 2^{64} security.
 - ▶ 2008 Bernstein–Lange–Peters: broken in $\approx 2^{60}$ cycles.
 - ▶ Easily scale up for higher security.
- ▶ 1986 Niederreiter: Simplified and smaller version of McEliece.
 - ▶ Public key: H with 1's on the diagonal. This form hides the efficient way of decoding this code.
 - ▶ Secret key: the fast Goppa decoder.
 - ▶ Encryption: Randomly generate e with t bits set.
Send $H \cdot e$.
 - ▶ Use hash of e to encrypt message with symmetric crypto (with 256 bits key).
- ▶ The passive attacker is facing a t -error correcting problem for the public key H , which looks like a random code.

Security analysis

- Some papers studying algorithms for attackers:
1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
2015 May–Ozerov.

Security analysis

- ▶ Some papers studying algorithms for attackers:
1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
2015 May–Ozerov.
- ▶ 256 KB public key for 2^{146} pre-quantum security.
- ▶ 512 KB public key for 2^{187} pre-quantum security.
- ▶ 1024 KB public key for 2^{263} pre-quantum security.

Security analysis

- ▶ Some papers studying algorithms for attackers:
1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
2015 May–Ozerov.
- ▶ 256 KB public key for 2^{146} pre-quantum security.
- ▶ 512 KB public key for 2^{187} pre-quantum security.
- ▶ 1024 KB public key for 2^{263} pre-quantum security.
- ▶ Post-quantum (Grover): below 2^{263} , above 2^{131} .

McBits (Bernstein, Chou, Schwabe, CHES 2013)

- ▶ Encryption is super fast anyways (just a vector-matrix multiplication, done with $1 + 1 = 0$).
- ▶ Main step in decryption is decoding of Goppa code. The McBits software achieves this in **constant time**.
- ▶ Decoding speed at 2^{128} pre-quantum security:
 $(n; t) = (4096; 41)$ uses 60493 Ivy Bridge cycles.
- ▶ Decoding speed at 2^{263} pre-quantum security:
 $(n; t) = (6960; 119)$ uses 306102 Ivy Bridge cycles.
- ▶ Very fast constant-time decryption:
<https://binary.cr.yp.to/mcbits.html>.
- ▶ Main time spent on public-key encryption. symmetric crypto adds very little to that.

POST-QUANTUM KEY EXCHANGE



A NEW HOPE

ERDEM ALKIM

LÉO DUCAS

THOMAS PÖPPELMANN

PETER SCHWABE

Faster and smaller post-quantum confidentiality?

Forward secrecy

- ▶ “Classical” public-key crypto (PKC): encrypt to long-term key
- ▶ Problem: key compromise breaks confidentiality of all past messages

Faster and smaller post-quantum confidentiality?

Forward secrecy

- ▶ “Classical” public-key crypto (PKC): encrypt to long-term key
- ▶ Problem: key compromise breaks confidentiality of all past messages
- ▶ Modern PKC: use **ephemeral keys** for confidentiality
- ▶ Use long-term keys only for authentication (e.g., via signatures)
- ▶ This is often called (Perfect) Forward Secrecy (PFS)

Faster and smaller post-quantum confidentiality?

Forward secrecy

- ▶ “Classical” public-key crypto (PKC): encrypt to long-term key
- ▶ Problem: key compromise breaks confidentiality of all past messages
- ▶ Modern PKC: use **ephemeral keys** for confidentiality
- ▶ Use long-term keys only for authentication (e.g., via signatures)
- ▶ This is often called (Perfect) Forward Secrecy (PFS)
- ▶ This needs **ephemeral key exchange**:
 - ▶ Require fast key generation and transmission
 - ▶ Require short public keys

Faster and smaller post-quantum confidentiality?

Forward secrecy

- ▶ “Classical” public-key crypto (PKC): encrypt to long-term key
- ▶ Problem: key compromise breaks confidentiality of all past messages
- ▶ Modern PKC: use **ephemeral keys** for confidentiality
- ▶ Use long-term keys only for authentication (e.g., via signatures)
- ▶ This is often called (Perfect) Forward Secrecy (PFS)
- ▶ This needs **ephemeral key exchange**:
 - ▶ Require fast key generation and transmission
 - ▶ Require short public keys
- ▶ Different security notions, different optimization possibilities

Faster and smaller post-quantum confidentiality?

Forward secrecy

- ▶ “Classical” public-key crypto (PKC): encrypt to long-term key
- ▶ Problem: key compromise breaks confidentiality of all past messages
- ▶ Modern PKC: use **ephemeral keys** for confidentiality
- ▶ Use long-term keys only for authentication (e.g., via signatures)
- ▶ This is often called (Perfect) Forward Secrecy (PFS)
- ▶ This needs **ephemeral key exchange**:
 - ▶ Require fast key generation and transmission
 - ▶ Require short public keys
- ▶ Different security notions, different optimization possibilities
- ▶ Note: PFS does **not** protect against cryptanalytical break!

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>





ISARA Radiate

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

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

<https://www.isara.com/isara-radiate/>



Products Applications Tools **About Infineon** Careers

Newsletter Contact Where to Buy English myinfineon login

Search

Press General Information **Press Releases** Market News Press Kits Media Pool Events Contacts

> Home > About Infineon > Press > Press Releases > Ready for tomorrow: Infineon demonstrates first post-quantum cryptography on a contactless security chip

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

May 30, 2017 | Business & Financial Press

Press Contact



Karin Braeckle
T +49 89 234 23424
[Send E-mail](#)

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

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



A bit of (R)LWE history

- ▶ Hoffstein, Pipher, Silverman, 1996: NTRU cryptosystem
- ▶ Regev, 2005: Introduce LWE-based encryption
- ▶ Lyubashevsky, Peikert, Regev, 2010: Ring-LWE and Ring-LWE encryption
- ▶ Ding, Xie, Lin, 2012: Transform to (R)LWE-based key exchange
- ▶ Peikert, 2014: Improved RLWE-based key exchange
- ▶ Bos, Costello, Naehrig, Stebila, 2015: Instantiate and implement Peikert's key exchange in TLS:
- ▶ Alkim, Ducas, Pöppelmann, Schwabe, Aug. 2016: NEWHOPE
- ▶ Alkim, Ducas, Pöppelmann, Schwabe, Dec. 2016: NEWHOPE-Simple



Ring-Learning-with-errors (RLWE)

- ▶ Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- ▶ Let χ be an *error distribution* on \mathcal{R}_q
- ▶ Let $s \in \mathcal{R}_q$ be secret
- ▶ Attacker is given pairs $(a, as + e)$ with
 - ▶ a uniformly random from \mathcal{R}_q
 - ▶ e sampled from χ
- ▶ Task for the attacker: find s

Ring-Learning-with-errors (RLWE)

- ▶ Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- ▶ Let χ be an *error distribution* on \mathcal{R}_q
- ▶ Let $s \in \mathcal{R}_q$ be secret
- ▶ Attacker is given pairs $(a, as + e)$ with
 - ▶ a uniformly random from \mathcal{R}_q
 - ▶ e sampled from χ
- ▶ Task for the attacker: find s
- ▶ Common choice for χ : discrete Gaussian
- ▶ Common optimization for protocols: fix a

RLWE-based Encryption, KEM, KEX

The basic idea

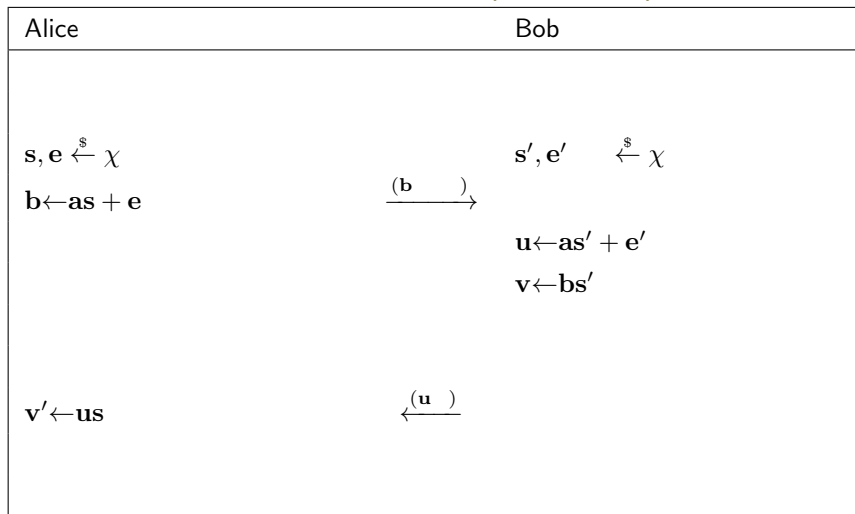
Alice (server)		Bob (client)
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e$	\xrightarrow{b}	$u \leftarrow as' + e'$
	\xleftarrow{u}	

Alice has $v = us = ass' + e's$

Bob has $v' = bs' = ass' + es'$

- ▶ Secret and noise polynomials s, s', e, e' are small
- ▶ v and v' are *approximately* the same

NEWHOPE-Simple key exchange (simplified)



NEWHOPE-Simple key exchange (simplified)

Alice	Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$ $\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$ $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$ $\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$ $\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$ $\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$ $\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
	$\xrightarrow{(\mathbf{b}, seed)}$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u} \quad)}$



NEWHOPE-Simple key exchange (simplified)

Alice		Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}'$
		$\mathbf{k} \xleftarrow{\$} \{0, 1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(\mathbf{k})$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$

NEWHOPE-Simple key exchange (simplified)

Alice		Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}' + \mathbf{e}''$
		$\mathbf{k} \xleftarrow{\$} \{0, 1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(\mathbf{k})$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$



NEWHOPE-Simple key exchange (simplified)

Alice		Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}' + \mathbf{e}''$
		$\mathbf{k} \xleftarrow{\$} \{0, 1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(\mathbf{k})$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		

NEWHOPE-Simple key exchange (simplified)

Alice		Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}' + \mathbf{e}''$
		$\mathbf{k} \xleftarrow{\$} \{0, 1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(\mathbf{k})$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

NEWHOPE-Simple key exchange (simplified)

Alice		Bob
$seed \xleftarrow{\$} \{0, 1\}^{256}$		
$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \chi$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \chi$
$\mathbf{b} \leftarrow \mathbf{a}\mathbf{s} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{a}\mathbf{s}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{b}\mathbf{s}' + \mathbf{e}''$
		$\mathbf{k} \xleftarrow{\$} \{0, 1\}^n$
		$\mathbf{k} \leftarrow \text{Encode}(\mathbf{k})$
$\mathbf{v}' \leftarrow \mathbf{u}\mathbf{s}$	$\xleftarrow{(\mathbf{u}, \mathbf{c})}$	$\mathbf{c} \leftarrow \mathbf{v} + \mathbf{k}$
$\mathbf{k}' \leftarrow \mathbf{c} - \mathbf{v}'$		$\mu \leftarrow \text{Extract}(\mathbf{k})$
$\mu \leftarrow \text{Extract}(\mathbf{k}')$		

This is LPR encryption, written as KEX (except for generation of \mathbf{a})



Against all authority

- Standard approach to choosing a :

“Let a be a uniformly random. . .”

Against all authority

- ▶ Standard approach to choosing a :
“Let a be a uniformly random. . .”
- ▶ Standard *real-world* approach: generate fixed a once

Against all authority

- ▶ Standard approach to choosing a :
“Let a be a uniformly random. . .”
- ▶ Standard *real-world* approach: generate fixed a once
- ▶ What if a is backdoored?
- ▶ Parameter-generating authority can break key exchange
- ▶ “Solution”: Nothing-up-my-sleeves (involves endless discussion!)

Against all authority

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

Against all authority

- ▶ Standard approach to choosing a :
“Let a be a uniformly random...”
- ▶ Standard *real-world* approach: generate fixed a once
- ▶ What if a is backdoored?
- ▶ Parameter-generating authority can break key exchange
- ▶ “Solution”: Nothing-up-my-sleeves (involves endless discussion!)
- ▶ Even without backdoor:
 - ▶ Perform massive precomputation based on a
 - ▶ Use precomputation to break *all* key exchanges
 - ▶ Infeasible today, but who knows...
 - ▶ Attack in the spirit of Logjam
- ▶ Solution in NewHope(-Simple): Choose a fresh a every time
- ▶ Server can cache a for some time (e.g., 1h)



Against all authority

- ▶ Standard approach to choosing a :
“Let a be a uniformly random...”
- ▶ Standard *real-world* approach: generate fixed a once
- ▶ What if a is backdoored?
- ▶ Parameter-generating authority can break key exchange
- ▶ “Solution”: Nothing-up-my-sleeves (involves endless discussion!)
- ▶ Even without backdoor:
 - ▶ Perform massive precomputation based on a
 - ▶ Use precomputation to break *all* key exchanges
 - ▶ Infeasible today, but who knows...
 - ▶ Attack in the spirit of Logjam
- ▶ Solution in NewHope(-Simple): Choose a fresh a every time
- ▶ Server can cache a for some time (e.g., 1h)
- ▶ **Must not reuse keys/noise!**



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 $[-1/4q, 1/4q]$): set bit to zero
 - ▶ Closer to $\pm q/2$: set bit to one

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 $[-1/4q, 1/4q]$): set bit to zero
 - ▶ Closer to $\pm q/2$: set bit to one
- ▶ NEWHOPE-Simple: map $n/4$ bits to n coefficients
- ▶ Set 4 coefficients to 0 or to $q/2$
- ▶ Decode: map coeffs into $[-q/2, q/2]$, sum up 4 absolute values
 - ▶ Closer to 0 (i.e., in $[0, q]$): set bit to zero
 - ▶ Closer to $\pm 2q$: set bit to one



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 $[-1/4q, 1/4q]$): set bit to zero
 - ▶ Closer to $\pm q/2$: set bit to one
- ▶ NEWHOPE-Simple: map $n/4$ bits to n coefficients
- ▶ Set 4 coefficients to 0 or to $q/2$
- ▶ Decode: map coeffs into $[-q/2, q/2]$, sum up 4 absolute values
 - ▶ Closer to 0 (i.e., in $[0, q]$): set bit to zero
 - ▶ Closer to $\pm 2q$: set bit to one
- ▶ First proposed by Pöppelmann and Güneysu in 2013.

Reducing the size of \mathbf{c}

- ▶ Remember that Bob sends $\mathbf{c} = \mathbf{v} + \mathbf{k}$
- ▶ Alice recovers $\mathbf{k}' = \mathbf{c} - \mathbf{v}' \approx \mathbf{k}$
- ▶ Information about \mathbf{k} sits in high bits of \mathbf{c} coefficients
- ▶ Noise sits in low bits of \mathbf{c} coefficients

Reducing the size of \mathbf{c}

- ▶ Remember that Bob sends $\mathbf{c} = \mathbf{v} + \mathbf{k}$
- ▶ Alice recovers $\mathbf{k}' = \mathbf{c} - \mathbf{v}' \approx \mathbf{k}$
- ▶ Information about \mathbf{k} sits in high bits of \mathbf{c} coefficients
- ▶ Noise sits in low bits of \mathbf{c} coefficients
- ▶ Idea: **Don't transmit low bits**
- ▶ This introduces additional (deterministic, uniform) noise
- ▶ In NEWHOPE-Simple compress coefficients to 3 bits:

$$\bar{c}_i \leftarrow \lceil (c_i \cdot 8) / q \rceil \mod 8$$

Reducing the size of \mathbf{c}

- ▶ Remember that Bob sends $\mathbf{c} = \mathbf{v} + \mathbf{k}$
- ▶ Alice recovers $\mathbf{k}' = \mathbf{c} - \mathbf{v}' \approx \mathbf{k}$
- ▶ Information about \mathbf{k} sits in high bits of \mathbf{c} coefficients
- ▶ Noise sits in low bits of \mathbf{c} coefficients
- ▶ Idea: **Don't transmit low bits**
- ▶ This introduces additional (deterministic, uniform) noise
- ▶ In NEWHOPE-Simple compress coefficients to 3 bits:

$$\bar{c}_i \leftarrow \lceil (c_i \cdot 8) / q \rceil \mod 8$$

- ▶ Recover $c'_i \approx c_i$ on the receiver side:

$$c'_i \leftarrow \lceil (\bar{c}_i \cdot q) / 8 \rceil$$

Reducing the size of \mathbf{c}

- ▶ Remember that Bob sends $\mathbf{c} = \mathbf{v} + \mathbf{k}$
- ▶ Alice recovers $\mathbf{k}' = \mathbf{c} - \mathbf{v}' \approx \mathbf{k}$
- ▶ Information about \mathbf{k} sits in high bits of \mathbf{c} coefficients
- ▶ Noise sits in low bits of \mathbf{c} coefficients
- ▶ Idea: **Don't transmit low bits**
- ▶ This introduces additional (deterministic, uniform) noise
- ▶ In NEWHOPE-Simple compress coefficients to 3 bits:

$$\bar{c}_i \leftarrow \lceil (c_i \cdot 8) / q \rceil \mod 8$$

- ▶ Recover $c'_i \approx c_i$ on the receiver side:

$$c'_i \leftarrow \lceil (\bar{c}_i \cdot q) / 8 \rceil$$

- ▶ Technique known at least since 2009 (Peikert), used in various other protocols

Parameter choices

BCNS key exchange

- ▶ Starting point: Bos, Costello, Naehrig, Stebila 2015:
- ▶ $n = 1024, q = 2^{32} - 1$
- ▶ Error distribution: discrete Gaussian
- ▶ Claim: 128-bit pre-quantum security

Parameter choices

BCNS key exchange

- ▶ Starting point: Bos, Costello, Naehrig, Stebila 2015:
- ▶ $n = 1024$, $q = 2^{32} - 1$
- ▶ Error distribution: discrete Gaussian
- ▶ Claim: 128-bit pre-quantum security

NewHope(-Simple) key exchange

- ▶ $n = 1024$, $q = 12289$ (14 bits)
- ▶ Error distribution: centered binomial:
 - ▶ Sample uniformly random k -bit integers a and b
 - ▶ Output $HW(a) - HW(b)$ (HW = Hamming weight)
 - ▶ In NEWHOPE we use $k = 16$
- ▶ Claim: \gg 128-bit post-quantum security



Post-quantum security

- ▶ Consider RLWE instance as LWE instance
- ▶ Attack using BKZ
- ▶ BKZ uses SVP oracle in smaller dimension
- ▶ Consider only the cost of one call to that oracle (“core-SVP hardness”)



Post-quantum security

- ▶ Consider RLWE instance as LWE instance
- ▶ Attack using BKZ
- ▶ BKZ uses SVP oracle in smaller dimension
- ▶ Consider only the cost of one call to that oracle (“core-SVP hardness”)
- ▶ Consider quantum sieve as SVP oracle
 - ▶ Best-known quantum cost (BKC): $2^{0.265n}$
 - ▶ Best-plausible quantum cost (BPC): $2^{0.2075n}$

Post-quantum security

- ▶ Consider RLWE instance as LWE instance
- ▶ Attack using BKZ
- ▶ BKZ uses SVP oracle in smaller dimension
- ▶ Consider only the cost of one call to that oracle (“core-SVP hardness”)
- ▶ Consider quantum sieve as SVP oracle
 - ▶ Best-known quantum cost (BKC): $2^{0.265n}$
 - ▶ Best-plausible quantum cost (BPC): $2^{0.2075n}$
- ▶ Obtain lower bounds on the bit security:

	Known Classical	Known Quantum	Best Plausible
BCNS	86	78	61
NEWHOPE	281	255	199

Performance

	BCNS	C ref	AVX2
Key generation (server)	$\approx 2\,477\,958$	258 246	88 920
Key gen + shared key (client)	$\approx 3\,995\,977$	384 994	110 986
Shared key (server)	$\approx 481\,937$	86 280	19 422

- ▶ Cycle counts for NEWHOPE on one core of an Intel i7-4770K (Haswell)
- ▶ BCNS benchmarks are derived from `openssl speed`
- ▶ Includes around $\approx 37\,000$ cycles for generation of `a` on each side
- ▶ Compare to X25519 elliptic-curve scalar mult: 156 092 cycles

Google's conclusions

See <https://www.imperialviolet.org/2016/11/28/cecpq1.html>

“[...] we did not find any unexpected impediment to deploying something like NewHope. There were no reported problems caused by enabling it.”

Google's conclusions

See <https://www.imperialviolet.org/2016/11/28/cecpq1.html>

“[...] if the need arose, it would be practical to quickly deploy NewHope in TLS 1.2. (TLS 1.3 makes things a little more complex and we did not test with CECPQ1 with it.)”



Google's conclusions

See <https://www.imperialviolet.org/2016/11/28/cecpq1.html>

“Although the median connection latency only increased by a millisecond, the latency for the slowest 5% increased by 20ms and, for the slowest 1%, by 150ms. Since NewHope is computationally inexpensive, we’re assuming that this is caused entirely by the increased message sizes. Since connection latencies compound on the web (because subresource discovery is delayed), the data requirement of NewHope is moderately expensive for people on slower connections.”



NewHope(-Simple) online

- NewHope Paper: <https://cryptojedi.org/papers/#newhope>
- NHS Paper: <https://cryptojedi.org/papers/#newhopesimple>
- Software: <https://cryptojedi.org/crypto/#newhope>
- Newhope for ARM: <https://github.com/newhopearm/newhopearm.git>
(by Erdem Alkim, Philipp Jakubeit, and Peter Schwabe)

NewHope(-Simple) online

NewHope Paper: <https://cryptojedi.org/papers/#newhope>

NHS Paper: <https://cryptojedi.org/papers/#newhopesimple>

Software: <https://cryptojedi.org/crypto/#newhope>

Newhope for ARM: <https://github.com/newhopearm/newhopearm.git>
(by Erdem Alkim, Philipp Jakubeit, and Peter Schwabe)

More Software:

<https://ianix.com/pqcrypto/pqcrypto-deployment.html>



Further resources

- ▶ <https://2017.pqcrypto.org/school>: PQCRYPTO summer school with 21 lectures on video + slides + exercises.
- ▶ <https://2017.pqcrypto.org/exec> Executive school (12 lectures), less math, more overview. So far slides, soon videos.
- ▶ <https://2017.pqcrypto.org/conference> PQCrypto 2017; the latest results on post-quantum crypto.
- ▶ <https://pqcrypto.org>: Our survey site.
 - ▶ Many pointers: e.g., to PQCrypto conferences;
 - ▶ Bibliography for 4 major PQC systems.
- ▶ <https://pqcrypto.eu.org>: PQCRYPTO EU project.
Coming soon:
 - ▶ Expert recommendations.
 - ▶ Free software libraries.
 - ▶ More benchmarking to compare cryptosystems.
 - ▶ 2017: workshop and spring/summer school.
- ▶ https://twitter.com/pqc_eu: PQCRYPTO Twitter feed.
- ▶ <https://twitter.com/PQCryptoConf> PQCrypto conference Twitter feed.

