



Post-quantum crypto on μC

Peter Schwabe

peter@cryptojedi.org

<https://cryptojedi.org>

December 12, 2017



Asymmetric crypto today

- Signatures today: RSA, DSA, ECDSA, EdDSA
- Key exchange and PKE today: RSA, DH, ECDH,
- All based on factoring or (EC)DL



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

Peter W. Shor[†]

Abstract

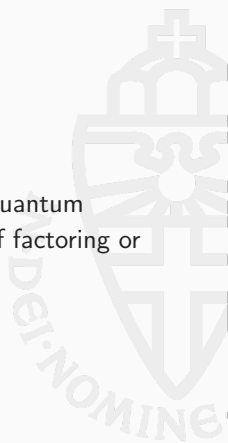
A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. 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.

Asymmetric crypto today

- Signatures today: RSA, DSA, ECDSA, EdDSA
- Key exchange and PKE today: RSA, DH, ECDH,
- All based on factoring or (EC)DL

Post-quantum crypto

(Asymmetric) cryptography that resists attacks by a large quantum computer, in particular, crypto not based on the hardness of factoring or (EC)DL.



Asymmetric crypto today

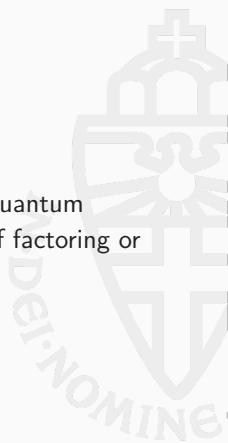
- Signatures today: RSA, DSA, ECDSA, EdDSA
- Key exchange and PKE today: RSA, DH, ECDH,
- All based on factoring or (EC)DL

Post-quantum crypto

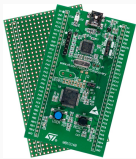
(Asymmetric) cryptography that resists attacks by a large quantum computer, in particular, crypto not based on the hardness of factoring or (EC)DL.

Today's talk

- Lattice-based (RLWE-based) key exchange
- Hash-based signatures

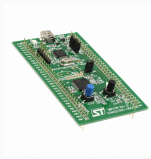


Cortex-M0



- STM32F0 development board
- Thumb + subset Thumb 2 (ARMv6-M)
- 8KB RAM
- 64KB Flash
- 8+8 registers

Cortex-M3



- STM32L100C development board
- Thumb2 instruction set (ARMv7-M)
- 16KB RAM
- 256KB Flash
- 16 registers (2 reserved)

Cortex-M4



- STM32F4 development board
- Thumb 2 instruction set (ARMv7-ME)
- 192KB RAM
- 1MB Flash
- 16 registers (2 reserved)

POST-QUANTUM KEY EXCHANGE



A NEW HOPE

ERDEM ALKIM

LÉO DUCAS

THOMAS PÖPPELMANN

PETER SCHWABE

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/>

Infineon

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>

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 $\mathbf{s} \in \mathcal{R}_q$ be secret
- Attacker is given pairs $(\mathbf{a}, \mathbf{as} + \mathbf{e})$ with
 - \mathbf{a} uniformly random from \mathcal{R}_q
 - \mathbf{e} sampled from χ
- Task for the attacker: find \mathbf{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 $\mathbf{s} \in \mathcal{R}_q$ be secret
- Attacker is given pairs $(\mathbf{a}, \mathbf{as} + \mathbf{e})$ with
 - \mathbf{a} uniformly random from \mathcal{R}_q
 - \mathbf{e} sampled from χ
- Task for the attacker: find \mathbf{s}
- Common choice for χ : discrete Gaussian
- Common optimization for protocols: fix \mathbf{a}



- 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

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

Alice has $\mathbf{t} = \mathbf{u}\mathbf{s} = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}'\mathbf{s}$

Bob has $\mathbf{t}' = \mathbf{b}\mathbf{s}' = \mathbf{a}\mathbf{s}\mathbf{s}' + \mathbf{e}\mathbf{s}'$

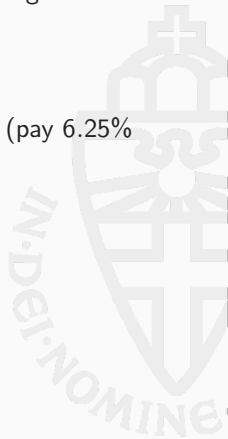
- Secret and noise polynomials $\mathbf{s}, \mathbf{s}', \mathbf{e}, \mathbf{e}'$ are small
- \mathbf{t} and \mathbf{t}' are *approximately* the same



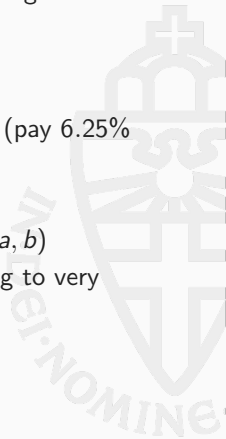
- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope



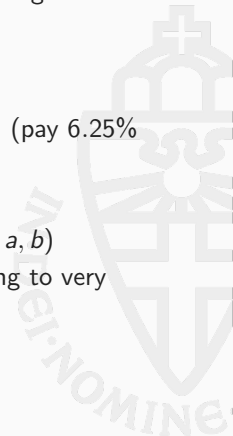
- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope
- NewHope-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)



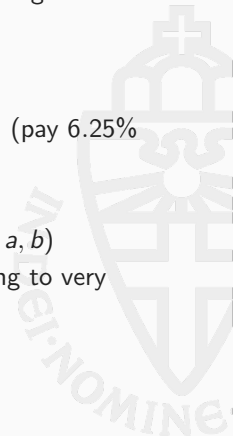
- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope
- NewHope-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Centered binomial noise ψ_k ($\text{HW}(a) - \text{HW}(b)$ for k -bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup



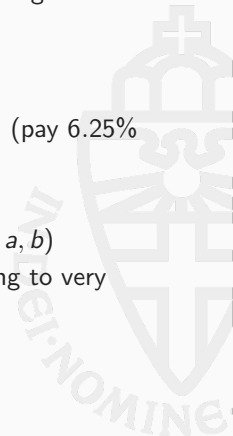
- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope
- NewHope-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Centered binomial noise ψ_k ($\text{HW}(a) - \text{HW}(b)$ for k -bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup
- Choose a fresh parameter \mathbf{a} for every protocol run



- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope
- NewHope-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Centered binomial noise ψ_k ($\text{HW}(a) - \text{HW}(b)$ for k -bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup
- Choose a fresh parameter \mathbf{a} for every protocol run
- Encode polynomials in NTT domain



- Improve IEEE S&P 2015 results by Bos, Costello, Naehrig, Stebila (BCNS)
- Use reconciliation to go from approximate agreement to agreement
 - Originally proposed by Ding (2012)
 - Improvements by Peikert (2014)
 - More improvements in NewHope
- NewHope-Simple (2016): Scrap complex reconciliation (pay 6.25% increase in ciphertext size)
- Very conservative parameters ($n = 1024, q = 12289$)
- Centered binomial noise ψ_k ($\text{HW}(a) - \text{HW}(b)$ for k -bit a, b)
- Achieve ≈ 256 bits of post-quantum security according to very conservative analysis
- Higher security, shorter messages, and $> 10\times$ speedup
- Choose a fresh parameter \mathbf{a} for every protocol run
- Encode polynomials in NTT domain
- C reference and AVX2 optimized implementation



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!

Performance on the M0

- All measurements at 48 MHz
- Keygen cycles: 1 170 892 cycles
- Encaps cycles: 1 760 837 cycles
- Decaps cycles: 298 877 cycles



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!

Performance on the M0

- All measurements at 48 MHz
- Keygen cycles: 1 170 892 cycles
- Encaps cycles: 1 760 837 cycles
- Decaps cycles: 298 877 cycles
- Curve25519: 3 589 850



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!

Performance on the M4

- All measurements at 48 MHz
- Keygen cycles: 781 518 cycles
- Encaps cycles: 1 140 594 cycles
- Decaps cycles: 174 798 cycles



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!

Performance on the M4

- All measurements at 48 MHz
- Keygen cycles: 781 518 cycles
- Encaps cycles: 1 140 594 cycles
- Decaps cycles: 174 798 cycles
- Curve25519: 907 240 cycles



- Joint work with Erdem Alkim and Philipp Jakubeit
- Hand-optimized NTT implementation
- New speed records for NTT on Cortex-M
- Most other routines also in assembly
- Fits into 8 KB of RAM on the M0!

Performance on the M4

- All measurements at 48 MHz
- Keygen cycles: 781 518 cycles
- Encaps cycles: 1 140 594 cycles
- Decaps cycles: 174 798 cycles
- Curve25519: 907 240 cycles

Public key and ciphertext each \approx 2 KB



- Submission by Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila



- Submission by Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila
- Start with NewHope-Simple
- Slightly modify noise \rightarrow negligible failure prob.



- Submission by Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila
- Start with NewHope-Simple
- Slightly modify noise \rightarrow negligible failure prob.
- Provide CPA-secure and CCA-secure KEM



- Submission by Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila
- Start with NewHope-Simple
- Slightly modify noise \rightarrow negligible failure prob.
- Provide CPA-secure and CCA-secure KEM
- Additional tweaks to improve speed
- Also provide low-security (“level 1”) variant



SPHINCS: practical stateless hash-based incredibly nice cryptographic signatures



Daniel J. Bernstein

Daira Hopwood

Andreas Hülsing

Tanja Lange

Ruben Niederhagen

Louiza Papachristodoulou

Michael Schneider

Peter Schwabe

Zooko Wilcox-O'Hearn



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)



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 $b = 1$: $\sigma = r_1$



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 $h(\sigma) = p_b$



Key generation

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



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}) :
 $\sigma = (\sigma_0, \dots, \sigma_{255}) = (r_{0,b_0}, \dots, r_{255,b_{255}})$



Key generation

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

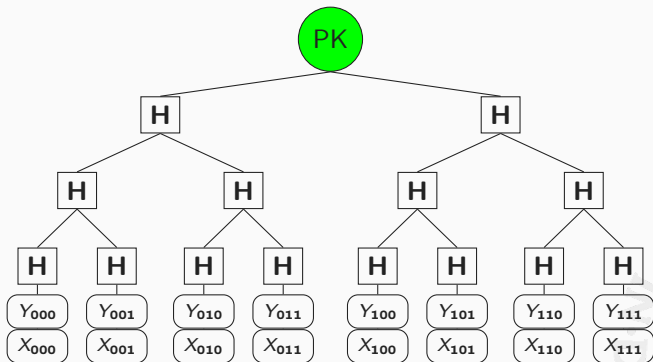
Signing

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

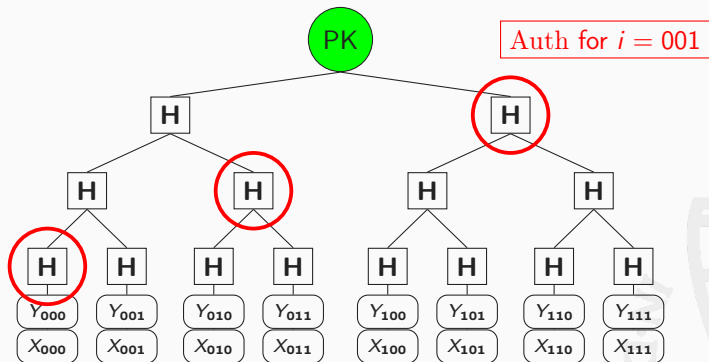
Verification

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



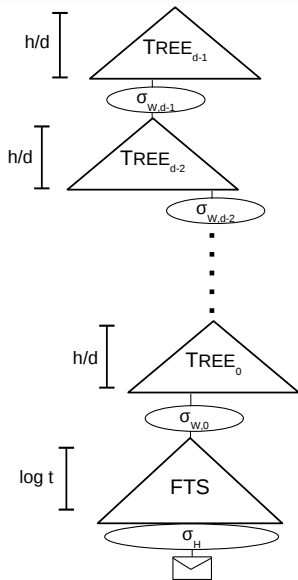


- Merkle, 1979: Leverage one-time signatures to multiple messages
- Binary hash tree on top of OTS public keys
- Use OTS keys sequentially

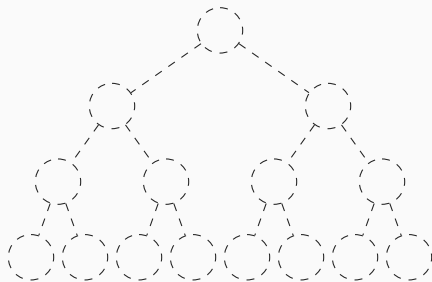


- $SIG = (i, \text{sign}(M, X_i), Y_i, \text{Auth})$
- Need to remember current *index* (\Rightarrow stateful scheme)
- State is a *“huge foot cannon”* (Langley, 2013)

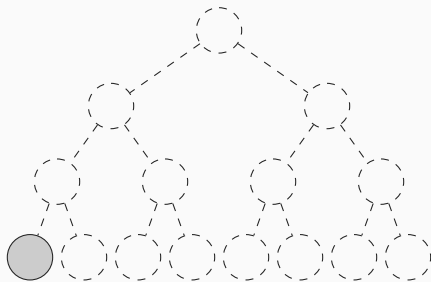
- Combine Merkle tree with “signing tree” by Goldreich
- Use a “hyper-tree” of total height h
- Pick index (pseudo-)randomly
- Messages signed with *few-time* signature scheme
- SPHINCS-256 for 128-bit post-quantum security (up to 2^{50} signatures under one key)
- Signature size of 41 KB



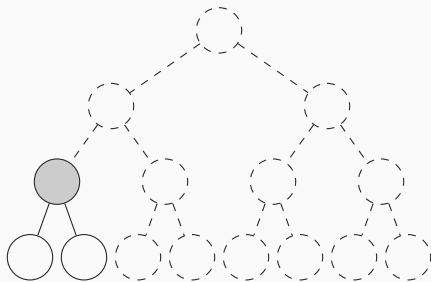
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



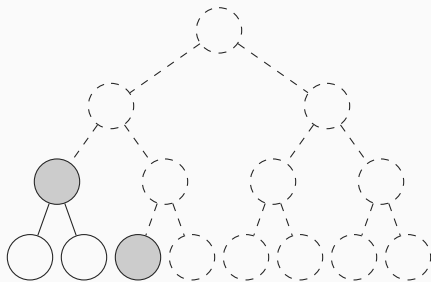
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



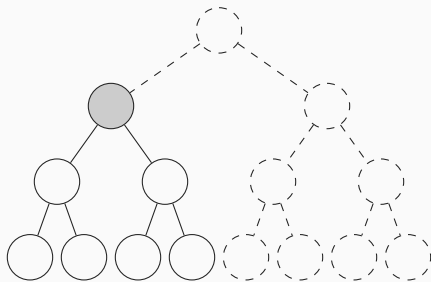
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



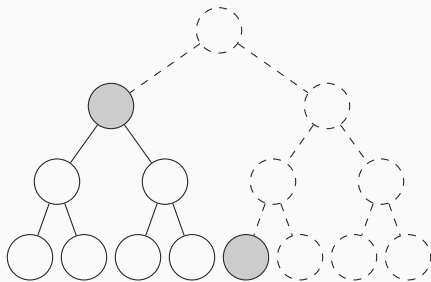
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



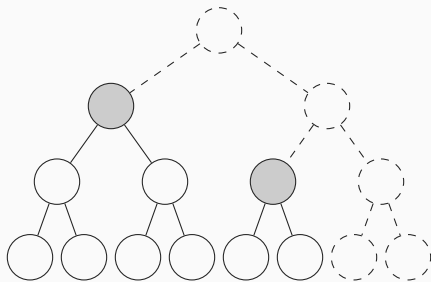
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



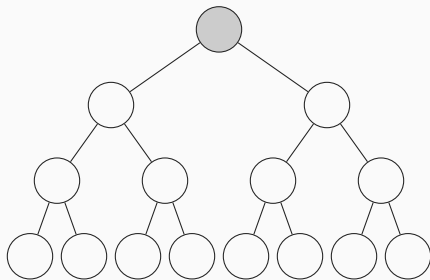
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



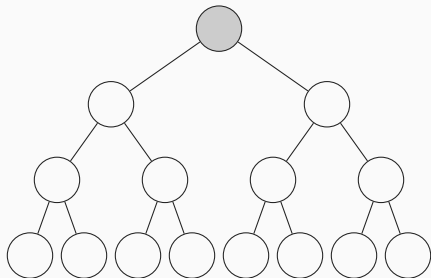
- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)



- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)
- Trace 32 'random' paths through tree



- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)
- Trace 32 'random' paths through tree
- Stream in message "piece by piece"



- Main challenge: Fit 40 KB signature into 16 KB of RAM
- Use Treehash (Merkle, 1990) inside FTS computation
- Maintain a stack: at most $\log(n) = 16$ nodes (or $\log(8) = 3$, in the example below)
- Trace 32 'random' paths through tree

- Stream in message "piece by piece"
- Stream out signature "piece by piece"
- Rearrange puzzle pieces on the hostside



- Works on 16KB RAM ✓
 - Uses less than 7 KB
- Benchmarks at 32 MHz
- Key generation: 0.88 seconds
- Signing: 18.41 seconds
- Verification: 0.51 seconds



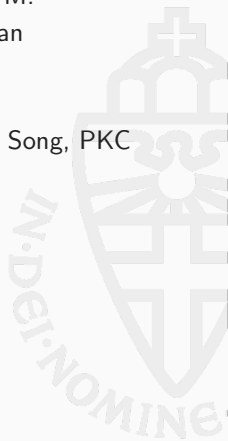
- Works on 16KB RAM ✓
 - Uses less than 7 KB
- Benchmarks at 32 MHz
- Key generation: 0.88 seconds
- Signing: 18.41 seconds
- Verification: 0.51 seconds
- Cost for eliminating the state: $30\times$ signing slowdown
- Typically better to use stateful XMSS
- (Verification) code for SPHINCS and XMSS has very large overlap



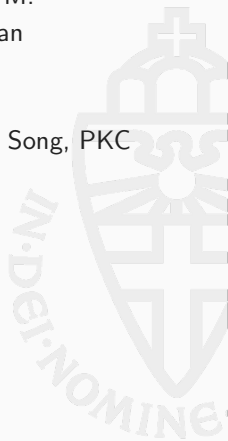
- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe



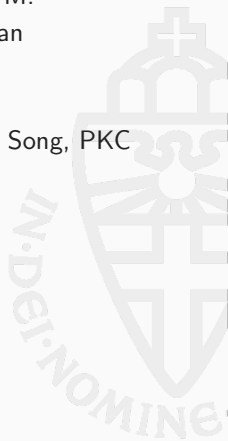
- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing



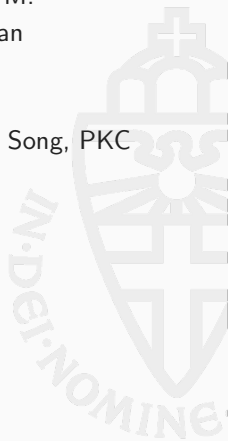
- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing
- Improvements to FTS → smaller signatures



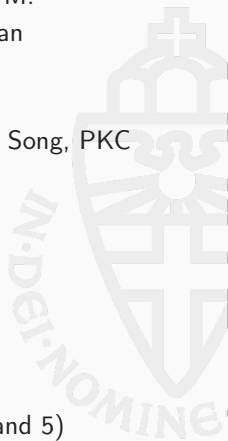
- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing
- Improvements to FTS → smaller signatures
- Support 2^{64} signatures under one key



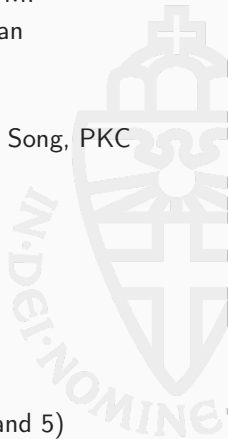
- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing
- Improvements to FTS → smaller signatures
- Support 2^{64} signatures under one key
- Framework supporting different hash functions



- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing
- Improvements to FTS → smaller signatures
- Support 2^{64} signatures under one key
- Framework supporting different hash functions
- Add lower-security variants (parameters for levels 1,3, and 5)



- Submission by Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe
- Various improvements to SPHINCS → SPHINCS⁺
- Incorporate multi-target protection (Hülsing, Rijneveld, Song, PKC 2016)
 - Better security properties
 - Shorter keys (64 bytes)
 - Slower signing
- Improvements to FTS → smaller signatures
- Support 2^{64} signatures under one key
- Framework supporting different hash functions
- Add lower-security variants (parameters for levels 1,3, and 5)
- Signature sizes between 8 KB and 49 KB



NewHope

<https://newhopecrypto.org> (soon)

SPHINCS⁺

<https://sphincs.org> (soon)

