Is Java Card ready for hash-based signatures?

Ebo van der Laan[1], Erik Poll[2], **Joost Rijneveld**[2],
Joeri de Ruiter[2], Peter Schwabe[2] and Jan Verschuren[1]

[1] Netherlands National Communication Security Agency (NLNCSA)
[2] Radboud University, Nijmegen, The Netherlands

2018-09-04
IWSEC 2018

Not really, no.

Not really, no.

Reviewer 1: *"an ill-fated attempt"*

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms
  - .. need new digital signature algorithms

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms
  - .. need new digital signature algorithms
  - .. still have symmetric crypto

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms
  - .. need new digital signature algorithms
  - .. still have symmetric crypto
- No DLP or factoring, but various new (and old) problems
  - Lattices, codes, $\mathcal{MQ}$, isogenies, hashes, ...
  - Ongoing NIST not-a-competition

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms
  - .. need new digital signature algorithms $\Leftarrow$
  - .. still have symmetric crypto $\Leftarrow$
- No DLP or factoring, but various new (and old) problems
  - Lattices, codes, $\mathcal{MQ}$, isogenies, hashes, ...
  - Ongoing NIST not-a-competition
- This talk: hash-based signatures

Post-quantum cryptography

- In the event of a large, practical quantum computer, we ..
  - .. need new key exchange algorithms
  - .. need new digital signature algorithms $\Leftarrow$
  - .. still have symmetric crypto $\Leftarrow$
- No DLP or factoring, but various new (and old) problems
  - Lattices, codes, $\mathcal{MQ}$, isogenies, hashes, . . .
  - Ongoing NIST not-a-competition
- This talk: hash-based signatures
  - Pre-image resistance: $\mathcal{H}(x) = y \not\Rightarrow x$
  - *The* conservative choice
  - RFC 8391: XMSS and XMSS^{MT}

Authenticating a single bit

- Preparation step:
  - Generate $\boxed{s_{YES}}$ and $\boxed{s_{NO}}$ (large random values)

Authenticating a single bit

- Preparation step:
  - Generate $\boxed{s_{YES}}$ and $\boxed{s_{NO}}$ (large random values)
  - Publish $h(\boxed{s_{YES}})$ and $h(\boxed{s_{NO}})$

Authenticating a single bit

- Preparation step:
  - Generate $\boxed{s_{YES}}$ and $\boxed{s_{NO}}$ (large random values)
  - Publish $h(\boxed{s_{YES}})$ and $h(\boxed{s_{NO}})$

time passes

Authenticating a single bit

- Preparation step:
  - Generate $\boxed{s_{YES}}$ and $\boxed{s_{NO}}$ (large random values)
  - Publish $h(\boxed{s_{YES}})$ and $h(\boxed{s_{NO}})$

time passes

- Authentication step:
  - Publish $\boxed{s_{YES}}$ or $\boxed{s_{NO}}$ to authenticate ‘YES’ or ‘NO’

Authenticating a single bit

- Preparation step:
  - Generate $\boxed{s_{YES}}$ and $\boxed{s_{NO}}$ (large random values)
  - Publish $h(\boxed{s_{YES}})$ and $h(\boxed{s_{NO}})$

time passes

- Authentication step:
  - Publish $\boxed{s_{YES}}$ or $\boxed{s_{NO}}$ to authenticate 'YES' or 'NO'
- Anyone can check and compare to hashes
- Can never re-use!

Lamport signatures

- ‘Classic example’ of hash-based signatures

Lamport signatures

- ‘Classic example’ of hash-based signatures
- Private key: N pairs of random numbers

$$\begin{array}{ccccccc} \boxed{s_{0,0}} & \boxed{s_{1,0}} & \boxed{s_{2,0}} & \cdots & \boxed{s_{N-3,0}} & \boxed{s_{N-2,0}} & \boxed{s_{N-1,0}} \\ \boxed{s_{0,1}} & \boxed{s_{1,1}} & \boxed{s_{2,1}} & \cdots & \boxed{s_{N-3,1}} & \boxed{s_{N-2,1}} & \boxed{s_{N-1,1}} \end{array}$$

Lamport signatures

- ‘Classic example’ of hash-based signatures
- Private key: N pairs of random numbers

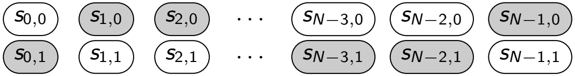
$$\begin{array}{ccccccc}
\boxed{s_{0,0}} & \boxed{s_{1,0}} & \boxed{s_{2,0}} & \cdots & \boxed{s_{N-3,0}} & \boxed{s_{N-2,0}} & \boxed{s_{N-1,0}} \\
\boxed{s_{0,1}} & \boxed{s_{1,1}} & \boxed{s_{2,1}} & \cdots & \boxed{s_{N-3,1}} & \boxed{s_{N-2,1}} & \boxed{s_{N-1,1}}
\end{array}$$

- Public key: hashes of these random numbers

$$\begin{array}{ccccccc}
\mathrm{h}(\boxed{s_{0,0}}) & \mathrm{h}(\boxed{s_{1,0}}) & \mathrm{h}(\boxed{s_{2,0}}) & \cdots & \mathrm{h}(\boxed{s_{N-3,0}}) & \mathrm{h}(\boxed{s_{N-2,0}}) & \mathrm{h}(\boxed{s_{N-1,0}}) \\
\mathrm{h}(\boxed{s_{0,1}}) & \mathrm{h}(\boxed{s_{1,1}}) & \mathrm{h}(\boxed{s_{2,1}}) & \cdots & \mathrm{h}(\boxed{s_{N-3,1}}) & \mathrm{h}(\boxed{s_{N-2,1}}) & \mathrm{h}(\boxed{s_{N-1,1}})
\end{array}$$

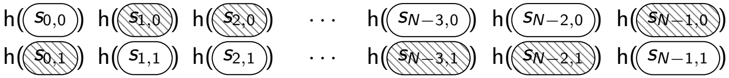
Lamport signatures

- 'Classic example' of hash-based signatures
- Private key: N pairs of random numbers



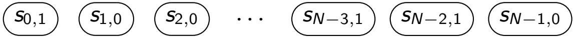
$s_{0,0}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,0}$ $s_{N-2,0}$ $s_{N-1,0}$
$s_{0,1}$ $s_{1,1}$ $s_{2,1}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,1}$

- Public key: hashes of these random numbers



$h(s_{0,0})$ $h(s_{1,0})$ $h(s_{2,0})$ $\cdots$ $h(s_{N-3,0})$ $h(s_{N-2,0})$ $h(s_{N-1,0})$
$h(s_{0,1})$ $h(s_{1,1})$ $h(s_{2,1})$ $\cdots$ $h(s_{N-3,1})$ $h(s_{N-2,1})$ $h(s_{N-1,1})$

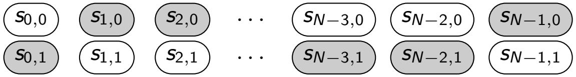
- Signature on N-bit value, e.g. 100...110



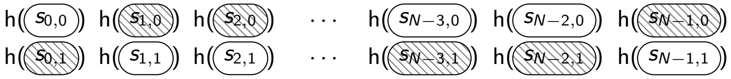
$s_{0,1}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,0}$

Lamport signatures

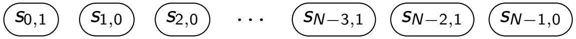
- ‘Classic example’ of hash-based signatures
- Private key: N pairs of random numbers


$s_{0,0}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,0}$ $s_{N-2,0}$ $s_{N-1,0}$
$s_{0,1}$ $s_{1,1}$ $s_{2,1}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,1}$

- Public key: hashes of these random numbers


$h(s_{0,0})$ $h(s_{1,0})$ $h(s_{2,0})$ $\cdots$ $h(s_{N-3,0})$ $h(s_{N-2,0})$ $h(s_{N-1,0})$
$h(s_{0,1})$ $h(s_{1,1})$ $h(s_{2,1})$ $\cdots$ $h(s_{N-3,1})$ $h(s_{N-2,1})$ $h(s_{N-1,1})$

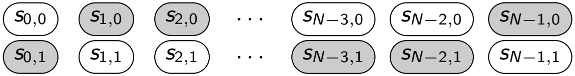
- Signature on N-bit value, e.g. 100...110


$s_{0,1}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,0}$

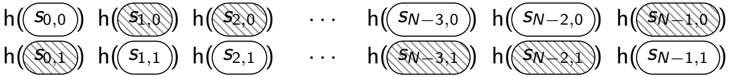
- Verification: hash, compare to public key

Lamport signatures

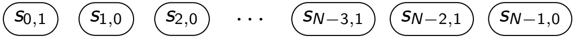
- ‘Classic example’ of hash-based signatures
- Private key: N pairs of random numbers


$s_{0,0}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,0}$ $s_{N-2,0}$ $s_{N-1,0}$
$s_{0,1}$ $s_{1,1}$ $s_{2,1}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,1}$

- Public key: hashes of these random numbers


$h(s_{0,0})$ $h(s_{1,0})$ $h(s_{2,0})$ $\cdots$ $h(s_{N-3,0})$ $h(s_{N-2,0})$ $h(s_{N-1,0})$
$h(s_{0,1})$ $h(s_{1,1})$ $h(s_{2,1})$ $\cdots$ $h(s_{N-3,1})$ $h(s_{N-2,1})$ $h(s_{N-1,1})$

- Signature on N-bit value, e.g. 100. . .110


$s_{0,1}$ $s_{1,0}$ $s_{2,0}$ $\cdots$ $s_{N-3,1}$ $s_{N-2,1}$ $s_{N-1,0}$

- Verification: hash, compare to public key
- Can still only do this **once**!

The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size

The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`

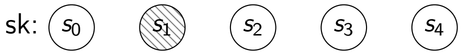
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`

sk:
s_0
s_1
s_2
s_3
s_4

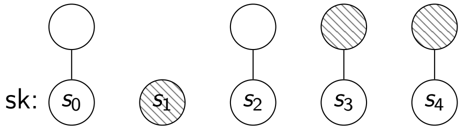
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



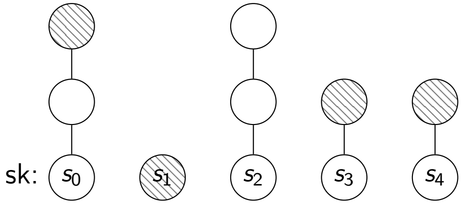
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



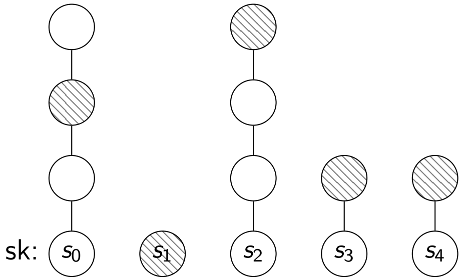
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign 10 00 11 01 01



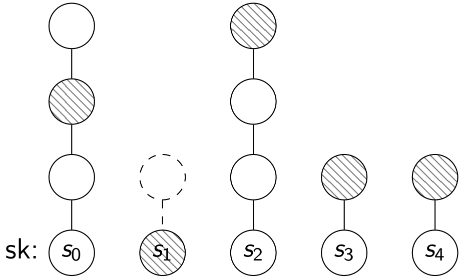
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



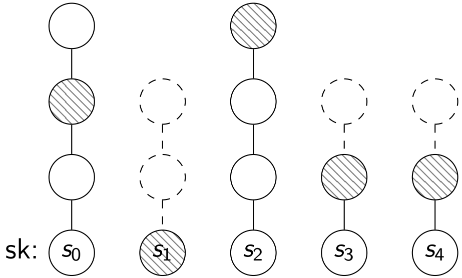
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



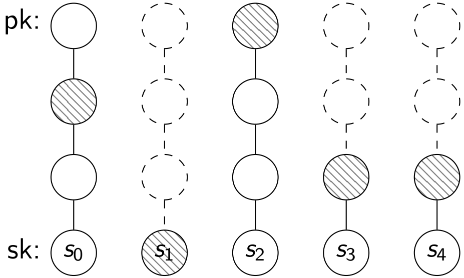
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



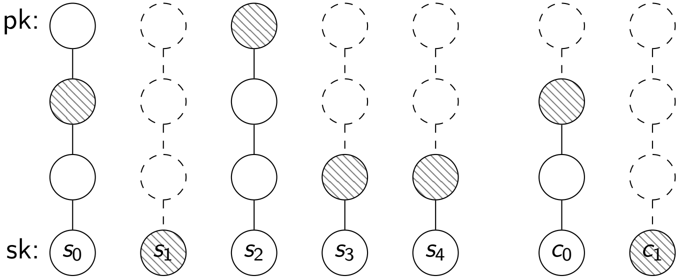
The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign 10 00 11 01 01



The Winternitz improvement

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let's sign `10 00 11 01 01`



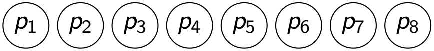
- Checksum: $\Sigma_{i=1}^{\ell_1}(w - 1 - m_i)$, convert to base w

Merkle trees

- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys

Merkle trees

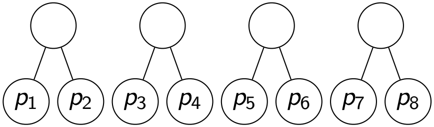
- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys
- Merkle: build 'authentication tree' on top



- Leaf p_i = OTS public key i

Merkle trees

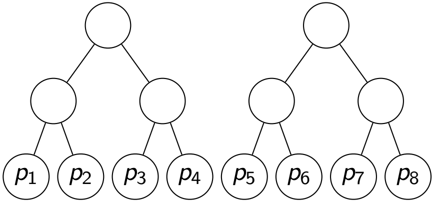
- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys
- Merkle: build ‘authentication tree’ on top



- Leaf p_i = OTS public key i
- Parent = h(LeftChild $\|$ RightChild)

Merkle trees

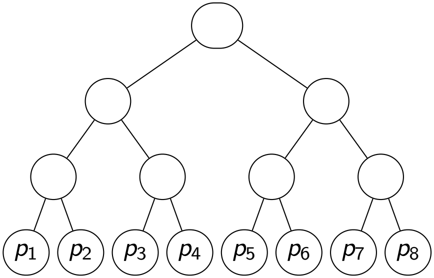
- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys
- Merkle: build ‘authentication tree’ on top



- Leaf p_i = OTS public key i
- Parent = h(LeftChild $\|$ RightChild)

Merkle trees

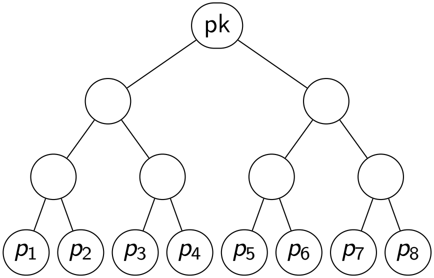
- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys
- Merkle: build 'authentication tree' on top



- Leaf p_i = OTS public key i
- Parent = h(LeftChild $\|$ RightChild)

Merkle trees

- One public key, multiple signatures?
  - OTS, so multiple signatures $\rightarrow$ multiple private keys
- Merkle: build 'authentication tree' on top



- Leaf p_i = OTS public key i
- Parent = h(LeftChild $\|$ RightChild)
- New public key: root node

Merkle trees

- Signature must now include:
  - OTS signature

Merkle trees

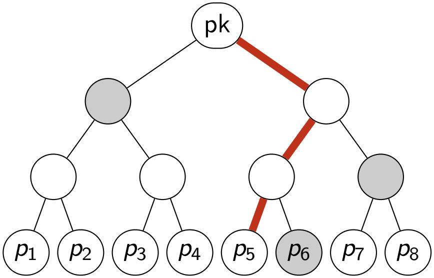
- Signature must now include:
  - OTS signature
  - OTS public key

Merkle trees

- Signature must now include:
  - OTS signature
  - OTS public key
  - Index in the Merkle tree, e.g. 5

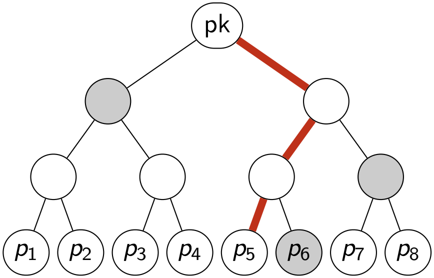
Merkle trees

- Signature must now include:
  - OTS signature
  - OTS public key
  - Index in the Merkle tree, e.g. 5
  - Nodes along the *authentication path*



Merkle trees

- Signature must now include:
  - OTS signature
  - OTS public key
  - Index in the Merkle tree, e.g. 5
  - Nodes along the *authentication path*



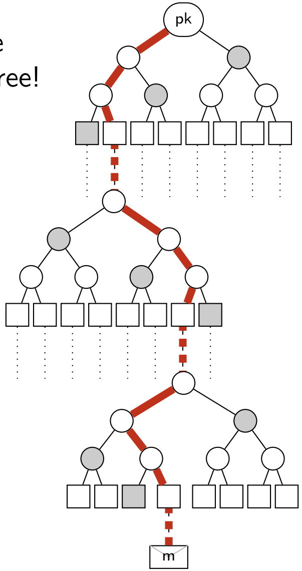
- Verification
  1. Implicitly verify OTS signature (reconstruct OTS public key)
  2. Reconstruct root node (using authentication path)

XMSS^{MT}

- Number of signatures $\Rightarrow$ size of tree
- Cannot reasonably generate entire tree!

XMSS^{MT}

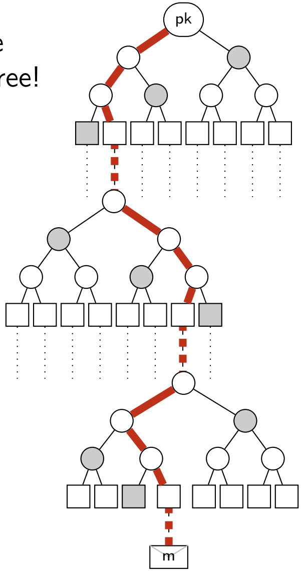
- Number of signatures $\Rightarrow$ size of tree
- Cannot reasonably generate entire tree!
  - 'Tree of trees'
  - Only generate needed subtrees
  - Link trees with OTS



XMSS^{MT}

- Number of signatures $\Rightarrow$ size of tree
- Cannot reasonably generate entire tree!
  - 'Tree of trees'
  - Only generate needed subtrees
  - Link trees with OTS

- Remember partial tree
  - Tree traversal

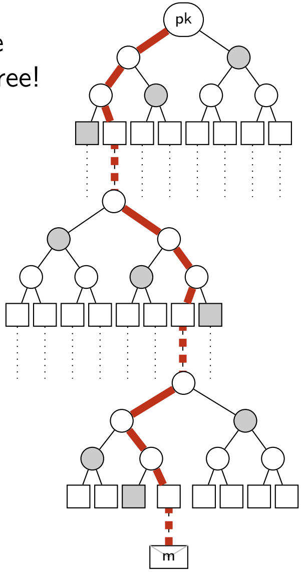


XMSS^{MT}

- Number of signatures $\Rightarrow$ size of tree
- Cannot reasonably generate entire tree!
  - 'Tree of trees'
  - Only generate needed subtrees
  - Link trees with OTS

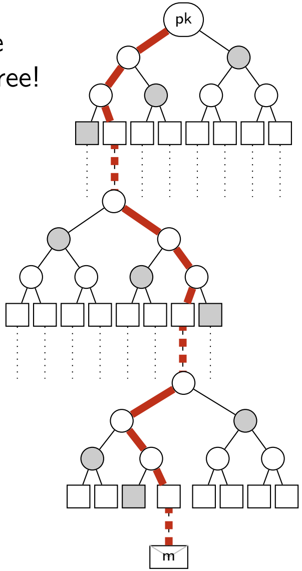
- Remember partial tree
  - Tree traversal

- Speed / size trade-offs



XMSS^{MT}

- Number of signatures ⇒ size of tree
- Cannot reasonably generate entire tree!
  - ‘Tree of trees’
  - Only generate needed subtrees
  - Link trees with OTS
- Remember partial tree
  - Tree traversal
- Speed / size trade-offs
- In practice:
  - Prevent multi-target attacks
  - 64 byte public keys, 2-20 KiB sig.
  - Standardized as RFC 8391



Java Card

- Smartcard platform standard
  - Oracle, ‘Java Card forum’
  - 20 billion cards solds (2016)
  - Java-based, but: no GC, 16-bit shorts, ...

Java Card

- Smartcard platform standard
  - Oracle, ‘Java Card forum’
  - 20 billion cards solds (2016)
  - Java-based, but: no GC, 16-bit shorts, ...
- APIs for cryptographic primitives
  - ‘Flexible’

Java Card

- Smartcard platform standard
  - Oracle, ‘Java Card forum’
  - 20 billion cards solds (2016)
  - Java-based, but: no GC, 16-bit shorts, ...
- APIs for cryptographic primitives
  - ‘Flexible’
- $\sim$10-100 KiB ROM, $\sim$1-10 KiB RAM

Java Card

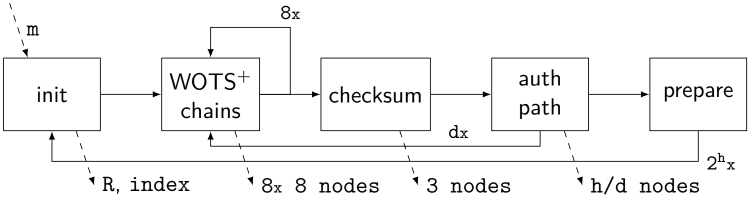
- Smartcard platform standard
  - Oracle, ‘Java Card forum’
  - 20 billion cards solds (2016)
  - Java-based, but: no GC, 16-bit shorts, ...
- APIs for cryptographic primitives
  - ‘Flexible’
- $\sim$10-100 KiB ROM, $\sim$1-10 KiB RAM
- XMSS on ‘bare metal’ [HBB13]
- We focus on JC 2.2.2 to 3.0.4

Java Card

- Smartcard platform standard
  - Oracle, ‘Java Card forum’
  - 20 billion cards solds (2016)
  - Java-based, but: no GC, 16-bit shorts, ...
- APIs for cryptographic primitives
  - ‘Flexible’
- ~10-100 KiB ROM, ~1-10 KiB RAM
- XMSS on ‘bare metal’ [HBB13]
- We focus on JC 2.2.2 to 3.0.4
  - Context: already-deployed Java Cards, to authenticate VPN

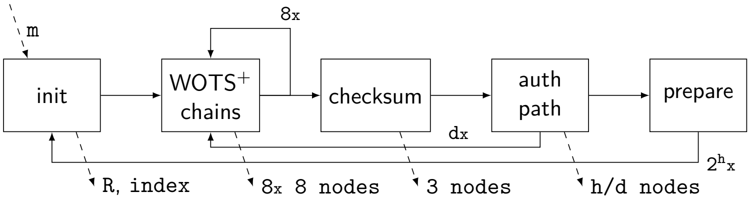
Implementation

- APDU-centered design
  - 256 bytes per output block



Implementation

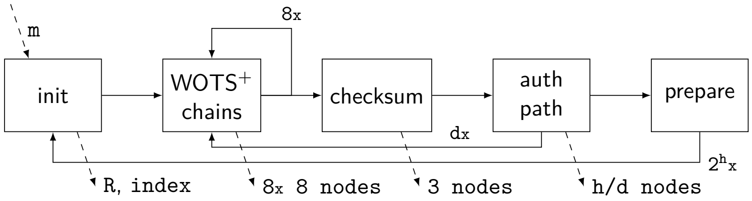
- APDU-centered design
    - 256 bytes per output block



- Retain leafs, compute in next tree when consumed
    - Computationally most expensive part
    - Slight unbalance
    - Prepare *after* signing

Implementation

- APDU-centered design
  - 256 bytes per output block



- Retain leafs, compute in next tree when consumed
  - Computationally most expensive part
  - Slight unbalance
  - Prepare *after* signing
- Treehash algorithm for WOTS+ leafs

Hash functions

- SHA-256
- Many hash calls on small input
- `MessageDigest` API

Hash functions

- SHA-256
- Many hash calls on small input
- `MessageDigest` API
- AES-based hashing?
  - More accessible hardware support?
  - Davies-Meyer? Matyas-Meyer-Oseas?
  - Parallelism using ECB mode?

Hash functions

- SHA-256
- Many hash calls on small input
- `MessageDigest` API
- AES-based hashing?
  - More accessible hardware support?
  - Davies-Meyer? Matyas-Meyer-Oseas?
  - Parallelism using ECB mode?

- Java stack is the bottleneck!

- $h = 20, d = 5$, 13 KiB signatures; 50 sec. signing!

Java Card API considerations

- Usability vs. flexibility vs. performance

Java Card API considerations

- Usability vs. flexibility vs. performance

- 'Parallel' hashing
  - duals of existing methods, i.e. `updateParallel`
  - Very flexible, requires expert developers

Java Card API considerations

- Usability vs. flexibility vs. performance

- ‘Parallel’ hashing
  - duals of existing methods, i.e. `updateParallel`
  - Very flexible, requires expert developers

- Complete WOTS$^+$ chains
  - Combine $16 \cdot 67$ calls; overlapping inputs
  - Less flexible across schemes

Java Card API considerations

- Usability vs. flexibility vs. performance

- ‘Parallel’ hashing
  - duals of existing methods, i.e. `updateParallel`
  - Very flexible, requires expert developers

- Complete WOTS$^+$ chains
  - Combine $16 \cdot 67$ calls; overlapping inputs
  - Less flexible across schemes

- WOTS$^+$ nodes and hash trees
  - Focus on output; abstract traversal algorithms
  - Complex parameter trade-offs

Java Card API considerations

- Usability vs. flexibility vs. performance
- ‘Parallel’ hashing
  - duals of existing methods, i.e. `updateParallel`
  - Very flexible, requires expert developers
- Complete WOTS$^+$ chains
  - Combine $16 \cdot 67$ calls; overlapping inputs
  - Less flexible across schemes
- WOTS$^+$ nodes and hash trees
  - Focus on output; abstract traversal algorithms
  - Complex parameter trade-offs
- ‘Complete signature’ API?

Java Card API considerations

- Usability vs. flexibility vs. performance
- ‘Parallel’ hashing
  - duals of existing methods, i.e. `updateParallel`
  - Very flexible, requires expert developers
- Complete WOTS$^+$ chains
  - Combine $16 \cdot 67$ calls; overlapping inputs
  - Less flexible across schemes
- WOTS$^+$ nodes and hash trees
  - Focus on output; abstract traversal algorithms
  - Complex parameter trade-offs
- ‘Complete signature’ API?
- Side-channel countermeasures?

Discussion

- Relevant and suitable use-case!
  - Long-term security
  - State management comes naturally

Discussion

- Relevant and suitable use-case!
  - Long-term security
  - State management comes naturally
- The theory is ready!

Discussion

- Relevant and suitable use-case!
  - Long-term security
  - State management comes naturally
- The theory is ready!
- The software is ready!
  - (Modulo use-case specific trade-offs)

Discussion

- Relevant and suitable use-case!
  - Long-term security
  - State management comes naturally
- The theory is ready!
- The software is ready!
  - (Modulo use-case specific trade-offs)
- The platform.. is not ☹

Discussion

- Relevant and suitable use-case!
    - Long-term security
    - State management comes naturally
- The theory is ready!
- The software is ready!
    - (Modulo use-case specific trade-offs)
- The platform.. is not ☹
- Call to action for manufacturers
    - .. let's talk!

Discussion

- Relevant and suitable use-case!
  - Long-term security
  - State management comes naturally
- The theory is ready!
- The software is ready!
  - (Modulo use-case specific trade-offs)
- The platform.. is not ☹
- Call to action for manufacturers
  - .. let's talk!
- Code is available (public domain): `https://joostrijneveld.nl/papers/javacard-xmss`