Hash-based Signatures

Andreas Hülsing

Summer School on Post-Quantum Cryptography
June 2017, TU Eindhoven
Post-Quantum Signatures

Lattice, MQ, Coding

⚠️ Signature and/or key sizes

⚠️ Runtimes

⚠️ Secure parameters

\[
y_1 = x_1^2 + x_1 x_2 + x_1 x_4 + x_3 \\
y_2 = x_3^2 + x_2 x_3 + x_2 x_4 + x_1 + 1 \\
y_3 = ... 
\]
Hash-based Signature Schemes

[Mer89]

- Post quantum
- Only secure hash function
- Security well understood
- Fast
RSA — DSA — EC-DSA...

- RSA, DH, SVP, MQ, ...
- Intractability Assumption
- Cryptographic hash function
- Digital signature scheme
Hash function families
(Hash) function families

- $H_n := \{ h_k : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n \}$
- $m(n) \geq n$
- „efficient“
One-wayness

\[ H_n := \{ h_k : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n \} \]

\[
\begin{align*}
    & h_k \leftarrow H_n \\
    & x \leftarrow \{0,1\}^{m(n)} \\
    & y_c \leftarrow h_k(x)
\end{align*}
\]

Success if \( h_k(x^*) = y_c \)
Collision resistance

\[ H_n := \{ h_k : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n \} \]

\[ h_k \leftarrow H_n \]

Success if

\[ h_k(x_1^*) = h_k(x_2^*) \] and
\[ x_1^* \neq x_2^* \]
Second-preimage resistance

$$H_n := \{h_k : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n\}$$

$\$ $\$ $\$ $\$

$h_k \leftarrow H_n$

$x_c \leftarrow \{0,1\}^{m(n)}$

Success if

$h_k(x_c) = h_k(x^*)$ and

$x_c \neq x^*$
Undetectability

\[ H_n := \{ h_k : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n \} \]

\[ h_k \leftarrow H_n \]

\[ b \leftarrow \{0,1\} \]

\textbf{If} \( b = 1 \)

\[ x \leftarrow \{0,1\}^{m(n)} \]

\[ y_c \leftarrow h_k(x) \]

\textbf{else}

\[ y_c \leftarrow \{0,1\}^n \]
Pseudorandomness

\[ H_n := \{ h_k: \{0,1\}^{m(n)} \to \{0,1\}^n \} \]
Generic security

• „Black Box“ security (best we can do without looking at internals)
  • For hash functions: Security of random function family

• (Often) expressed in #queries (query complexity)

• Hash functions not meeting generic security considered insecure
Generic Security - OWF

Classically:

• No query: Output random guess
  
  \( Succ_A^{OW} = \frac{1}{2^n} \)

• One query: Guess, check, output new guess
  
  \( Succ_A^{OW} = \frac{2}{2^n} \)

• q-queries: Guess, check, repeat q-times, output new guess
  
  \( Succ_A^{OW} = \frac{q+1}{2^n} \)

• Query bound: \( \Theta(2^n) \)
Generic Security - OWF

Quantum:
• More complex
• Reduction from quantum search for random $H$
• Know lower & upper bounds for quantum search!

• Query bound: $\Theta(2^{n/2})$

• Upper bound uses variant of Grover

(Disclaimer: Currently only proof for $2^m \gg 2^n$)
# Generic Security

<table>
<thead>
<tr>
<th></th>
<th>OW</th>
<th>SPR</th>
<th>CR</th>
<th>UD*</th>
<th>PRF*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>$\Theta(2^n)$</td>
<td>$\Theta(2^n)$</td>
<td>$\Theta(2^{n/2})$</td>
<td>$\Theta(2^n)$</td>
<td>$\Theta(2^n)$</td>
</tr>
<tr>
<td>Quantum</td>
<td>$\Theta(2^{n/2})$</td>
<td>$\Theta(2^{n/2})$</td>
<td>$\Theta(2^{n/3})$</td>
<td>$\Theta(2^{n/2})$</td>
<td>$\Theta(2^{n/2})$</td>
</tr>
</tbody>
</table>

* conjectured, no proof
Hash-function properties

- Collision-Resistance
- 2nd-Preimage-Resistance
- One-way
- Pseudorandom

Assumptions / Attacks:

- Stronger / easier to break
- Weaker / harder to break
Attacks on Hash Functions

- **MD5 Collisions (theo.)**
  - 2004

- **SHA1 Collisions (theo.)**
  - 2005

- **MD5 Collisions (practical!)**
  - 2008

- **SHA1 Collisions (practical!)**
  - 2017

**MD5 & SHA-1**
No (Second-) Preimage Attacks!

*The timeline indicates the years when attacks were discovered.*
Basic Construction
Lamport-Diffie OTS [Lam79]

Message $M = b_1, \ldots, b_m$, OWF $H$, $\ast = n$ bit
EU-CMA for OTS

Success if $M^* \neq M$ and $\text{Verify}(pk, \sigma^*, M^*) = \text{Accept}$
Security

Theorem:
If $H$ is one-way then LD-OTS is one-time eu-cma-secure.
Reduction

Input: $y_c, k$

Set $H \leftarrow h_k$

Replace random $\text{pk}_{i,b}$
Reduction

Input: $y_c, k$

Set $H \leftarrow h_k$

Replace random $pk_{i,b}$

Adv. Message: $M = b_1, \ldots, b_m$
If $b_i = b$ return fail
else return $\text{Sign}(M)$
Reduction

Input: $y_c, k$
Set $H \leftarrow h_k$
Choose random $pk_{i,b}$

Forgery: $M^* = b_1^*, ..., b_m^*$, $\sigma = \sigma_1, ..., \sigma_m$
If $b_i \neq b$ return fail
Else return $\sigma_i^*$
Reduction - Analysis

Abort in two cases:

1. $b_i = b$
   
   probability $\frac{1}{2} : b$ is a random bit

2. $b_i^* \neq b$
   
   probability $1 - \frac{1}{m}$: At least one bit has to flip as $M^* \neq M$

Reduction succeeds with A’s success probability times $\frac{1}{2m}$. 
Merkle’s Hash-based Signatures

\[ \text{SIG} = (i=2, \text{OTS}, \circ, \circ, \circ) \]
Security

Theorem:
MSS is eu-cma-secure if OTS is a one-time eu-cma secure signature scheme and H is a random element from a family of collision resistant hash functions.
Reduction

Input: $k, pk_{OTS}$

1. Choose random $0 \leq i < 2^h$
2. Generate key pair using $pk_{OTS}$ as $i$th OTS public key and $H \leftarrow h_k$
3. Answer all signature queries using $sk$ or sign oracle (for index $i$)
4. Extract OTS-forgery or collision from forgery
Reduction (Step 4, Extraction)

Forgery: \((i^*, \sigma_{OTS}^*, pk_{OTS}^*, \text{AUTH})\)

1. If \(pk_{OTS}^*\) equals OTS pk we used for \(i^*\) OTS, we got an OTS forgery.
   • Can only be used if \(i^* = i\).

2. Else adversary used different OTS pk.
   • Hence, different leaves.
   • Still same root!
   • Pigeon-hole principle: Collision on path to root.
Winternitz-OTS
Recap LD-OTS [Lam79]

Message $M = b_1, \ldots, b_m$, $\text{OWF} \ H$  

$\begin{align*}
&\text{SK} \\
&\begin{array}{c}
\text{sk}_{1,0} \\
\text{sk}_{1,1} \\
\vdots \\
\text{sk}_{m,0} \\
\text{sk}_{m,1}
\end{array} \\
&\begin{array}{c}
\text{H} \\
\text{H} \\
\vdots \\
\text{H} \\
\text{H}
\end{array} \\
&\begin{array}{c}
\text{PK} \\
\begin{array}{c}
\text{pk}_{1,0} \\
\text{pk}_{1,1} \\
\vdots \\
\text{pk}_{m,0} \\
\text{pk}_{m,1}
\end{array}
\end{array} \\
&\begin{array}{c}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{array} \\
&\begin{array}{c}
\text{Mux} \\
\text{Mux} \\
\vdots \\
\text{Mux}
\end{array} \\
&\begin{array}{c}
\text{Sig} \\
\text{sk}_{1,b_1} \\
\vdots \\
\text{sk}_{m,b_m}
\end{array}
\end{align*}$

$*$ = n bit
LD-OTS in MSS

\[ \text{SIG} = (i=2, \text{文档}, \text{钥匙}, \text{文件}, \text{文件}, \text{钥匙} ) \]

Verification:

1. Verify 📄
2. Verify authenticity of 🔍

We can do better!
Trivial Optimization

Message $M = b_1, \ldots, b_m$, OWF $H$

$\ast = n$ bit
Optimized LD-OTS in MSS

\[ \text{SIG} = (i=2, X, \bigcirc, \bigcirc, \bigcirc) \]

Verification:

1. Compute from
2. Verify authenticity of

Steps 1 + 2 together verify
Let’s sort this

**Message** \( M = b_1, ..., b_m \), OWF \( H \)

**SK:** \( sk_1, ..., sk_m, sk_{m+1}, ..., sk_{2m} \)

**PK:** \( H(sk_1), ..., H(sk_m), H(sk_{m+1}), ..., H(sk_{2m}) \)

**Encode \( M \):** \( M' = M \| \neg M = b_1, ..., b_m, \neg b_1, ..., \neg b_m \)

(instead of \( b_1, \neg b_1, ..., b_m, \neg b_m \))

**Sig:** \( \text{sig}_i = \begin{cases} 
    sk_i & , \text{if } b_i = 1 \\
    H(sk_i) & , \text{otherwise}
\end{cases} \)

Checksum with bad performance!
Optimized LD-OTS

Message $M = b_1, \ldots, b_m$, OWF $H$

SK: $sk_1, \ldots, sk_m, sk_{m+1}, \ldots, sk_{m+1+\log m}$

PK: $H(sk_1), \ldots, H(sk_m), H(sk_{m+1}), \ldots, H(sk_{m+1+\log m})$

Encode $M$: $M' = b_1, \ldots, b_m, \neg \sum_{1}^{m} b_i$

Sig: $\text{sig}_i = \begin{cases} 
  sk_i, & \text{if } b_i = 1 \\
  H(sk_i), & \text{otherwise}
\end{cases}$

IF one $b_i$ is flipped from 1 to 0, another $b_j$ will flip from 0 to 1
Function chains

Function family: \( H_n := \{ h_k : \{0,1\}^n \rightarrow \{0,1\}^n \} \)

\( h_k \leftarrow H_n \)

Parameter \( w \)

Chain: \( c^i(x) = h_k(c^{i-1}(x)) = h_k \circ h_k \circ \ldots \circ h_k(x) \)

\( i \)-times

\( c^0(x) = x \)

\( c^1(x) = h_k(x) \)

\( c^{w-1}(x) \)
WOTS

Winternitz parameter $w$, security parameter $n$, message length $m$, function family $H_n$

**Key Generation:** Compute $l$, sample $h_k$

$w$ 

$\text{pk}_0 = c^{w-1}(sk_0)$

$\text{pk}_1 = c^{w-1}(sk_1)$

$\text{pk}_r = c^{w-1}(sk_r)$

$\text{pk}_{r+1} = c^{w-1}(sk_{r+1})$

$\ldots$

$\text{pk}_{r+w} = c^{w-1}(sk_{r+w})$

$\text{pk}_{r+w+1} = c^{w-1}(sk_{r+w+1})$

$\ldots$

$\text{pk}_{2w-1} = c^{w-1}(sk_{2w-1})$

$\text{pk}_{2w} = c^{w-1}(sk_{2w})$

$\text{pk}_{2w+1} = c^{w-1}(sk_{2w+1})$

$\ldots$

$\text{pk}_{3w-1} = c^{w-1}(sk_{3w-1})$
WOTS Signature generation

\[
c^0(\text{sk}_1) = \text{sk}_1
\]

\[
\sigma_1 = c^{b_1(\text{sk}_1)}
\]

Signature:
\[
\sigma = (\sigma_1, \ldots, \sigma_\ell)
\]

\[
c^0(\text{sk}_\ell) = \text{sk}_\ell
\]

\[
\sigma_\ell = c^{b_\ell(\text{sk}_\ell)}
\]

\[
\text{pk}_\ell = c^{w-1}(\text{sk}_\ell)
\]

\[
\text{pk}_j = c^{w-1}(\text{sk}_j)
\]
WOTS Signature Verification

Verifier knows: $M, w$

Signature: $\sigma = (\sigma_1, \ldots, \sigma_\ell)$
WOTS Function Chains

For $x \in \{0,1\}^n$ define $c^0(x) = x$ and

- **WOTS**: $c^i(x) = h_k(c^{i-1}(x))$
- **WOTS$^\$:** $c^i(x) = h_{c^{i-1}(x)}(r)$
- **WOTS$^+$**: $c^i(x) = h_k(c^{i-1}(x) \oplus r_i)$
Theorem (informally):

**W-OTS** is strongly unforgeable under chosen message attacks if $H_n$ is a collision resistant family of undetectable one-way functions.

**WOTS$^s$** is existentially unforgeable under chosen message attacks if $H_n$ is a pseudorandom function family.

**WOTS$^+$** is strongly unforgeable under chosen message attacks if $H_n$ is a 2nd-preimage resistant family of undetectable one-way functions.
XMSS

Tree: Uses bitmasks

Leafs: Use binary tree with bitmasks

OTS: WOTS\(^+\)

Message digest: Randomized hashing

Collision-resilient

\(\rightarrow\) signature size halved
Multi-Tree XMSS

Uses multiple layers of trees

-> Key generation
   (= Building first tree on each layer)
   $\Theta(2^h) \rightarrow \Theta(d*2^{h/d})$

-> Allows to reduce
   worst-case signing times
   $\Theta(h/2) \rightarrow \Theta(h/2d)$
Authentication path computation
TreeHash
(Mer89)
TreeHash

- TreeHash(v,i): Computes node on level v with leftmost descendant L_i
- Public Key Generation: Run TreeHash(h,0)
TreeHash

<table>
<thead>
<tr>
<th>TreeHash(v,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Init Stack, N1, N2</td>
</tr>
<tr>
<td>2: For j = i to i+2^v-1 do</td>
</tr>
<tr>
<td>3: N1 = LeafCalc(j)</td>
</tr>
<tr>
<td>4: While N1.level() == Stack.top().level() do</td>
</tr>
<tr>
<td>5: N2 = Stack.pop()</td>
</tr>
<tr>
<td>6: N1 = ComputeParent( N2, N1 )</td>
</tr>
<tr>
<td>7: Stack.push(N1)</td>
</tr>
<tr>
<td>8: Return Stack.pop()</td>
</tr>
</tbody>
</table>
TreeHash

TreeHash(v,i)
Efficiency?

Key generation: Every node has to be computed once.
  cost = $2^h$ leaves + $2^{h-1}$ nodes
  => optimal

Signature: One node on each level $0 \leq v < h$.
  cost $2^{h-1}$ leaves + $2^{h-1} - h$ nodes.

Many nodes are computed many times!
(e.g. those on level $v=h-1$ are computed $2^{h-1}$ times)
  -> Not optimal if state allowed
The BDS Algorithm

[BDS08]
Motivation
(for all Tree Traversal Algorithms)

No Storage:
Signature: Compute one node on each level $0 \leq v < h$.
Costs: $2^h - 1$ leaf + $2^h - 1 - h$ node computations.

Example: XMSS with SHA2-256 and $h = 20$  -> approx. 15min

Store whole tree: $2^h n$ bits.

Example: $h=20$, $n=256$; storage: $2^{28}$bits = 32MB

Idea: Look for time-memory trade-off!
Use a State
Authentication Paths
Observation 1

Same node in authentication path is recomputed many times!

Node on level $v$ is recomputed for $2^v$ successive paths.

Idea: Keep authentication path in state.

$\Rightarrow$ Only have to update “new” nodes.

Result

Storage: $h$ nodes

Time: $\sim h$ leaf + $h$ node computations (average)

But: Worst case still $2^h - 1$ leaf + $2^h - 1 - h$ node computations!

$\Rightarrow$ Keep in mind. To be solved.
Observation 2

When new left node in authentication path is needed, its children have been part of previous authentication paths.
Computing Left Nodes

\[ v = 2 \]

\[ \in A(i - 1) \quad \in A(i - 1 - 2^{v-1}) \]
Result

Storing $\left\lfloor \frac{h}{2} \right\rfloor$ nodes

all left nodes can be computed with one node computation / node
Observation 3

Right child nodes on high levels are most costly.

Computing node on level $v$ requires $2^v$ leaf and $2^v-1$ node computations.

Idea: Store right nodes on top $k$ levels during key generation.

Result
Storage: $2^k-2$ n bit nodes
Time: $\sim h-k$ leaf + h-k node computations (average)

Still: Worst case $2^{h-k-1}$ leaf + $2^{h-k-1}-(h-k)$ node computations!
Distribute Computation
Intuition

Observation:
- For every second signature only one leaf computation
- Average runtime: $\sim h-k$ leaf + $h-k$ node computations

Idea: Distribute computation to achieve average runtime in worst case.

Focus on distributing computation of leaves
TreeHash with Updates

TreeHash.init(v,i)

1: Init Stack, N1, N2, j=i, j_max = i+2^v-1
2: Exit

TreeHash.update()

1: If j <= j_max
2: N1 = LeafCalc(j)
3: While N1.level() == Stack.top().level() do
5: N2 = Stack.pop()
6: N1 = ComputeParent( N2, N1 )
7: Stack.push(N1)
8: Set j = j+1
9: Exit

One leaf per update
Distribute Computation

Concept

- Run one TreeHash instance per level $0 \leq v < h-k$
- Start computation of next right node on level $v$ when current node becomes part of authentication path.
- Use scheduling strategy to guarantee that nodes are finished in time.
- Distribute $(h-k)/2$ updates per signature among all running TreeHash instances
Distribute Computation

Worst Case Runtime

Before:
$2^{h-k}-1$ leaf and $2^{h-k}-1-(h-k)$ node computations.

With distributed computation:
$(h-k)/2 + 1$ leaf and $3(h-k-1)/2 + 1$ node computations.

Add. Storage
- Single stack of size $h-k$ nodes for all TreeHash instances.
- + One node per TreeHash instance.
= $2(h-k)$ nodes
BDS Performance

Storage:

\[ 3h + \left\lfloor \frac{h}{2} \right\rfloor - 3k - 2 + 2^k \text{ } n \text{ bit nodes} \]

Runtime:

\( \frac{(h-k)}{2} + 1 \) leaf and

\( 3\frac{(h-k-1)}{2} + 1 \) node computations.
XMSS in practice
XMSS Internet-Draft
(draft-irtf-cfrg-xmss-hash-based-signatures)

• Protecting against multi-target attacks / tight security
  • n-bit hash => n bit security

• Small public key (2n bit)
  • At the cost of ROM for proving PK compression secure

• Function families based on SHA2

• Equal to XMSS-T [HRS16] up-to message digest
## XMSS / XMSS-T Implementation

C Implementation, using OpenSSL [HRS16]

<table>
<thead>
<tr>
<th></th>
<th>Sign (ms)</th>
<th>Signature (kB)</th>
<th>Public Key (kB)</th>
<th>Secret Key (kB)</th>
<th>Bit Security classical/quantum</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMSS</td>
<td>3.24</td>
<td>2.8</td>
<td>1.3</td>
<td>2.2</td>
<td>236 / 118</td>
<td>h = 20, d = 1,</td>
</tr>
<tr>
<td>XMSS-T</td>
<td>9.48</td>
<td>2.8</td>
<td><strong>0.064</strong></td>
<td>2.2</td>
<td><strong>256 / 128</strong></td>
<td>h = 20, d = 1</td>
</tr>
<tr>
<td>XMSS</td>
<td>3.59</td>
<td>8.3</td>
<td>1.3</td>
<td>14.6</td>
<td>196 / 98</td>
<td>h = 60, d = 3,</td>
</tr>
<tr>
<td>XMSS-T</td>
<td>10.54</td>
<td>8.3</td>
<td><strong>0.064</strong></td>
<td>14.6</td>
<td><strong>256 / 128</strong></td>
<td>h = 60, d = 3</td>
</tr>
</tbody>
</table>

Intel(R) Core(TM) i7 CPU @ 3.50GHz
XMSS-T uses message digest from Internet-Draft
All using SHA2-256, w = 16 and k = 2
Open research topics

1. Message compression which
   • is collision-resilient,
   • provides tight provable security,
   • especially resists multi-target attacks (=> no eTCR)
   • => Has applications outside hash-based crypto!

2. Quantum query complexity for further properties
   • E.g. PRF, UD, aSec, ...

3. Quantum security of existing hash function constructions.
   • E.g. can classical attacks be improved (e.g. differential cryptanalysis)
   • Formal proofs (see recent works on collapsing hashes)
SPHINCS
About the statefulness

• Works great for some settings

• However....
  ... back-up
  ... multi-threading
  ... load-balancing
ELIMINATE THE STATE
Few-Time Signature Schemes
Recap LD-OTS

Message $M = b_1, \ldots, b_n$, OWF $H$, $* = n$ bit
HORS [RR02]

Message M, OWF H, CRHF H’  
Parameters $t=2^a, k$, with $m = ka$ (typical $a=16, k=32$)
HORS mapping function

Message M, OWF H, CRHF H’ \[ * \] = n bit
Parameters \( t = 2^a, k \), with \( m = ka \) (typical \( a=16, k=32 \))
HORS

Message M, OWF H, CRHF H’ = n bit
Parameters t=2^a, k, with m = ka (typical a=16, k=32)
HORS Security

- $M$ mapped to $k$ element index set $M^i \in \{1, \ldots, t\}^k$
- Each signature publishes $k$ out of $t$ secrets
- Either break one-wayness or...

- $r$- Subset-Resilience: After seeing index sets $M^i_j$ for $r$ messages $msg_j$, $1 \leq j \leq r$, hard to find $msg_{r+1} \neq msg_j$ such that $M^i_{r+1} \in \bigcup_{1 \leq j \leq r} M^i_j$.

- Best generic attack: $\text{Succ}_{r\text{-SSR}}(A, q) = q \left( \frac{rk}{t} \right)^k$

$\rightarrow$ Security shrinks with each signature!
HORST

Using HORS with MSS requires adding PK (tn) to MSS signature.

HORST: Merkle Tree on top of HORS-PK
• New PK = Root
• Publish Authentication Paths for HORS signature values
• PK can be computed from Sig
• With optimizations: \( tn \rightarrow (k \log t - x + 1 + 2^x)n \)
  • E.g. SPHINCS-256: 2 MB \( \rightarrow \) 16 KB
• Use randomized message hash
SPHINCS

• Stateless Scheme
• XMSS$^{MT}$ + HORST + (pseudo-)random index
• Collision-resilient
• Deterministic signing

• SPHINCS-256:
  • 128-bit post-quantum secure
  • Hundrest of signatures / sec
  • 41 kb signature
  • 1 kb keys
Signatures via Non-Interactive Proofs: The Case of Fish & Picnic

Thanks to the Fish/Picnic team for slides
Interactive Proofs

Three move protocol:

- Important that $e$ unpredictable before sending $a$
- aka (Interactive) Honest-Verifier Zero-Knowledge Proofs

Non-interactive variant via Fiat-Shamir [FS86] transform
ZKBoo

Efficient $\Sigma$-protocols for arithmetic circuits

- generalization, simplification, + implementation of “MPC-in-the-head” [IKOS07]

Idea

1. (2,3)-decompose circuit into three shares
2. Revealing 2 parts reveals no information
3. Evaluate decomposed circuit per share
4. Commit to each evaluation
5. Challenger requests to open 2 of 3
6. Verifies consistency

Efficiency

- Heavily depends on #multiplications
High-Level Approach

• Use LowMC v2 to build dedicated hash function with low AND-gate-depth
• Use ZKBoo to proof knowledge of a preimage
• Use Fiat-Shamir to turn ZKP into Signature in ROM (Fish), or
• Use Unruh’s transform to turn ZKP into Signature in QROM (Picnic)
# Performance

| Scheme       | Gen | Sign | Verify | |sk| | pk| | |σ| | M       |
|--------------|-----|------|--------|--------|--------|--------|--------|--------|--------|---------|
| Fish-10-38   | 0.01| 29.73| 17.46  | 32     | 32/64  | 116K   | ROM    |
| Picnic-10-38 | 0.01| 31.31| 16.30  | 32     | 32/64  | 191K   | QROM   |
| MQ 5pass     | 1.0 | 7.2  | 5.0    | 32     | 74     | 40K    | ROM    |
| SPHINCS-256  | 0.8 | 1.0  | 0.6    | 1K     | 1K     | 40K    | SM     |
| BLISS-I      | 44  | 0.1  | 0.1    | 2K     | 7K     | 5.6K   | ROM    |
| Ring-TESLA   | 17K | 0.1  | 0.1    | 12K    | 8K     | 1.5K   | ROM    |
| TESLA-768    | 49K | 0.6  | 0.4    | 3.1M   | 4M     | 2.3K   | (Q)ROM |
| FS-Véron     | n/a | n/a  | n/a    | 32     | 160    | ≥126K  | ROM    |
| SIDHp751     | 16  | 7K   | 5K     | 48     | 768    | 138K   | QROM   |

**Table 2:** Timings (ms) and key/signature sizes (bytes)
Open research topics II

SPHINCS:
• More efficient few-time signatures
• Dedicated fast short, constant size input hash functions.

Fish / Picnic
• More efficient (size!!!) QROM transform
• Dedicated, more efficient proof for knowledge of preimage?
• Hash functions with lower AND-gate depth.
Thank you!
Questions?

For references & further literature see
https://huelsing.wordpress.com/hash-based-signature-schemes/literature/