Post-Quantum

Cryptography Conference

LMS: Lighter, faster key generation

Francisco José Vial-Prado

Senior Cryptography Engineer at Fortanix

ENTRUST HID PKI Consortium

LMS: Faster key generation, lighter keys

Francisco José Vial-Prado

Post-Quantum Cryptography Conference

7-8 November, 2023, Amsterdam (NL)

Fortanix

This talk

- **Introduction to LMS** *(Also see Volker Krummel's talk before lunch "Stateful Hash-Based Signature Schemes")*
- **Faster key generation** (Remarks on SIMD versions of RFC8554 algorithms)
- **Key size/signature speed trade-offs** (Recalling the "treehash" algorithms)

This talk

- **Introduction to LMS** *(Also see Volker Krummel's talk before lunch "Stateful Hash-Based Signature Schemes")*
- **Faster key generation** (Remarks on SIMD versions of RFC8554 algorithms)
- **Key size/signature speed trade-offs** (Recalling the "treehash" algorithms)

NOT This talk

State management, interoperability, export restrictions …

LMS = LM-OTS/LMS/HSS...

LMS is a *stateful hash-based signature scheme*

- Key generation requires hashing
- Signing a message requires hashing
- Verifying a signature requires hashing

LMS = LM-OTS/LMS/HSS...

LMS is a *stateful hash-based signature scheme*

- Key generation requires hashing
- Signing a message requires hashing
- Verifying a signature requires hashing

Also...

- There is an internal state that MUST evolve upon signing (typically, one counter).
- LMS keys can be organized into HSS keys, augmenting capacity

LMS = LM-OTS/LMS/HSS…

$$\boxed{\text{LM-OTS key}} \xrightarrow{\times 2^{h} \ \ (\leq 2^{25})} \boxed{\text{LMS key}} \xrightarrow{\times k \ \ (\leq 8)} \boxed{\text{HSS key}}$$

LMS = LM-OTS/LMS/HSS...

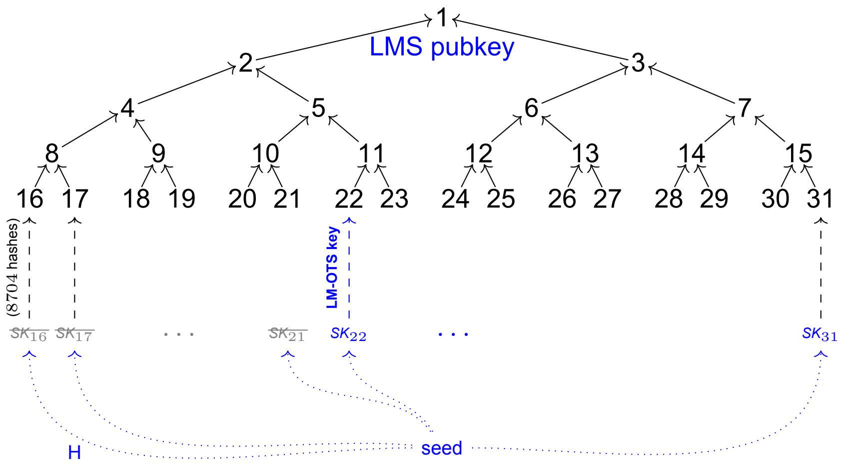
$$\begin{tikzcd} \boxed{\text{LM-OTS key}} \arrow[r, "\times 2^{h} \ (\leq 2^{25})"] & \boxed{\text{LMS key}} \arrow[r, "\times k \ (\leq 8)"] & \boxed{\text{HSS key}} \\ \text{(1 sig)} & (2^{h} \text{ sigs}) & (2^{h_1+\cdots+h_k} \text{ sigs}) \end{tikzcd}$$

LMS = LM-OTS/LMS/HSS...

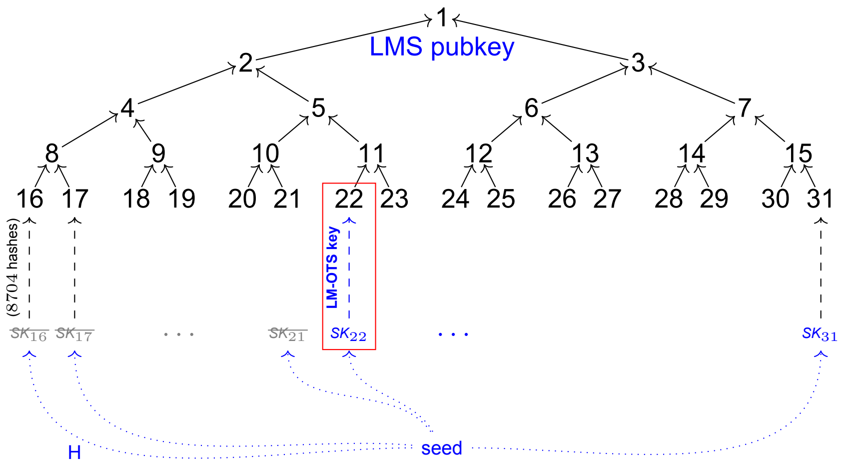
$$\boxed{\text{LM-OTS key}} \xrightarrow{\times 2^{h}\ \ (\leq 2^{25})} \boxed{\text{LMS key}} \xrightarrow{\times k\ \ (\leq 8)} \boxed{\text{HSS key}}$$

(1 sig)	(2^{h} sigs)	($2^{h_1+\cdots+h_k}$ sigs)
Keygen:		
$34 \times 255 + 34$ hashes	2^{h+1} hashes	$k-1$ sigs
(use once)	(maintain state)	(rotate exhausted LMS keys)

LMS = LM-OTS/LMS/HSS...



LMS = LM-OTS/LMS/HSS...



Single leaf calculation

$SK_{22} \xrightarrow{\textbf{LM-OTS key}} 22$

Single leaf calculation

$$
\begin{tikzcd}
SK_{22} \arrow[rrrr, dashed, "\textbf{LM-OTS key}"] \arrow[d, dotted] \arrow[dd, dotted, bend right] \arrow[dddd, dotted, bend right] & & & & 22 \\
x[0] \arrow[r] & h^1(x[0]) \arrow[r] & h^2(x[0]) \arrow[r] & \cdots \arrow[r] & h^{255}(x[0]) \arrow[d, dotted] \\
x[1] \arrow[r] & h^1(x[1]) \arrow[r] & h^2(x[1]) \arrow[r] & \cdots \arrow[r] & h^{255}(x[1]) \arrow[d, dotted] \\
\vdots & & & & \vdots \arrow[d, dotted] \\
x[33] \arrow[r] & h^1(x[33]) \arrow[r] & h^2(x[33]) \arrow[r] & \cdots \arrow[r] & h^{255}(x[33]) \arrow[uuuu, dotted, bend right]
\end{tikzcd}
$$

SHA-256 in SIMD is Easy™

- SHA-256 operates on 32-bit words
- Only uses bit shifts, rotation, and wrapping addition

SHA-256 in SIMD is Easy™

- SHA-256 operates on 32-bit words
- Only uses bit shifts, rotation, and wrapping addition
- Can compute `LANES` hash values In One Go!

Single leaf calculation

$$SK_{22} \dashrightarrow{} 22 \quad \textbf{LM-OTS key}$$

$$\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} \xrightarrow{\text{SIMD}_4} \begin{bmatrix} h^1(x[0]) \\ h^1(x[1]) \\ h^1(x[2]) \\ h^1(x[3]) \end{bmatrix} \longrightarrow \begin{bmatrix} h^2(x[0]) \\ h^2(x[1]) \\ h^2(x[2]) \\ h^2(x[3]) \end{bmatrix} \longrightarrow \cdots \longrightarrow \begin{bmatrix} h^{255}(x[0]) \\ h^{255}(x[1]) \\ h^{255}(x[2]) \\ h^{255}(x[3]) \end{bmatrix}$$

Single leaf calculation

$$\dashrightarrow SK_{22} \xrightarrow{\textbf{LM-OTS key}} 22$$

$$\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} \xrightarrow{\text{SIMD}_4} \begin{bmatrix} h^1(x[0]) \\ h^1(x[1]) \\ h^1(x[2]) \\ h^1(x[3]) \end{bmatrix} \longrightarrow \begin{bmatrix} h^2(x[0]) \\ h^2(x[1]) \\ h^2(x[2]) \\ h^2(x[3]) \end{bmatrix} \longrightarrow \cdots \longrightarrow \begin{bmatrix} h^{255}(x[0]) \\ h^{255}(x[1]) \\ h^{255}(x[2]) \\ h^{255}(x[3]) \end{bmatrix}$$

$$\Rightarrow \left\lceil \frac{34}{\mathtt{LANES} \times \mathtt{THREADS}} \right\rceil \times 255 + 34 \text{ calls}$$

LM-OTS signing

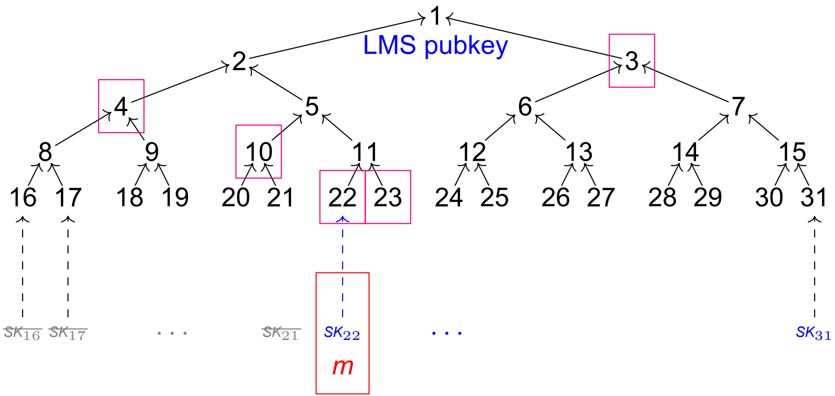
$$\begin{tikzcd}
SK_{22} \arrow[rrrrr, dashed, "\textbf{LM-OTS key}"] & & & & & 22 \\
& & \boxed{h^2(x[0])} \arrow[r] & \cdots \arrow[r] & h^{255}(x[0]) \arrow[d, dotted] & \\
& \boxed{h^1(x[1])} \arrow[r] & h^2(x[1]) \arrow[r] & \cdots \arrow[r] & h^{255}(x[1]) \arrow[d, dotted] & \\
\vdots & & & & \vdots \arrow[d, dotted] & \\
& & \boxed{h^{42}(x[33])} \arrow[r] & \cdots \arrow[r] & h^{255}(x[33]) \arrow[uuuur, dotted, bend right] &
\end{tikzcd}$$

Signer reveals intermediate values

Verifier hashes again

(Message dependency ends here)

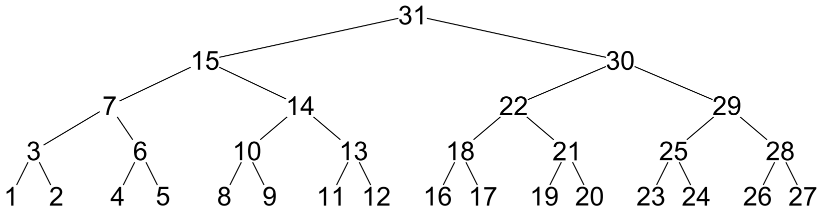
LMS signing



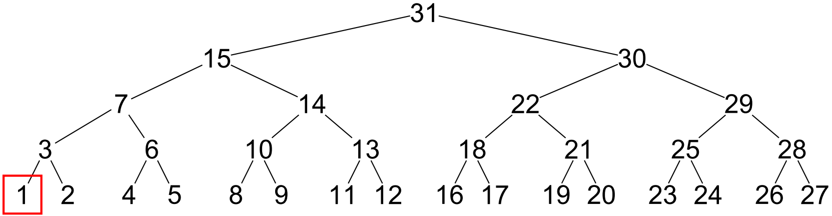
Signer needs to provide $\{23, 10, 4, 3\}$

Verifier hashes again

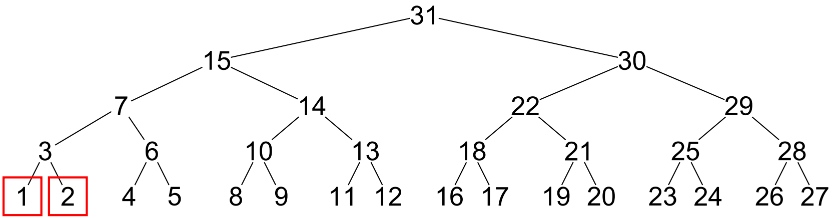
LMS root (RFC8554 app. C)



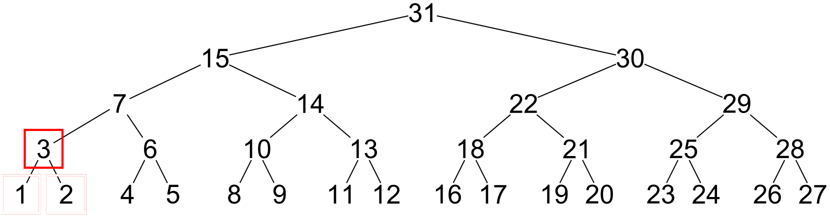
LMS root (RFC8554 app. C)



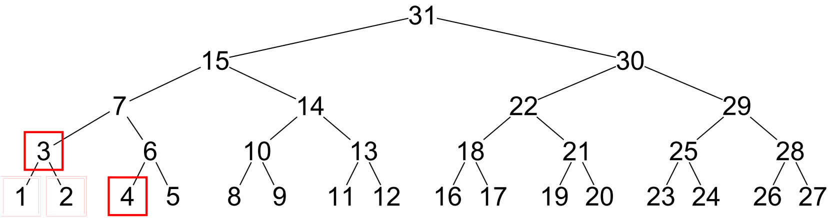
LMS root (RFC8554 app. C)



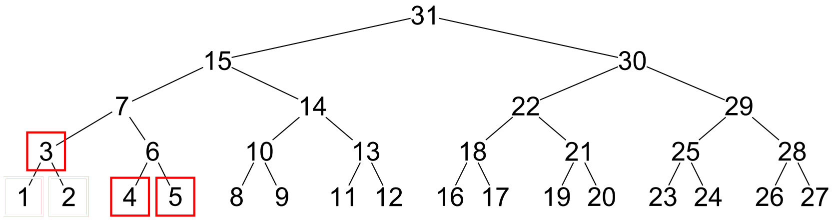
LMS root (RFC8554 app. C)



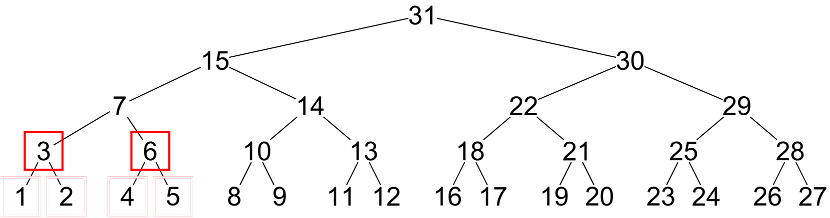
LMS root (RFC8554 app. C)



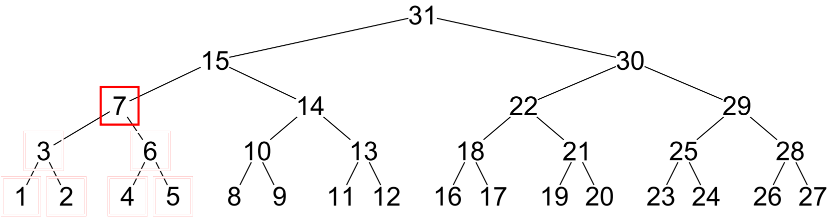
LMS root (RFC8554 app. C)



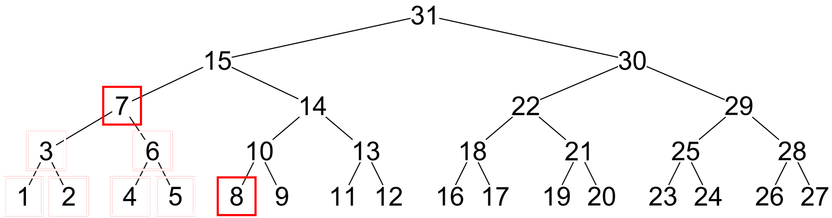
LMS root (RFC8554 app. C)



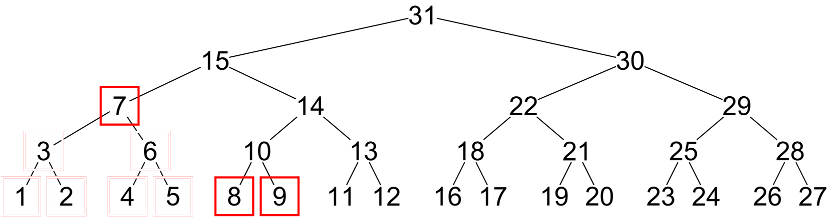
LMS root (RFC8554 app. C)



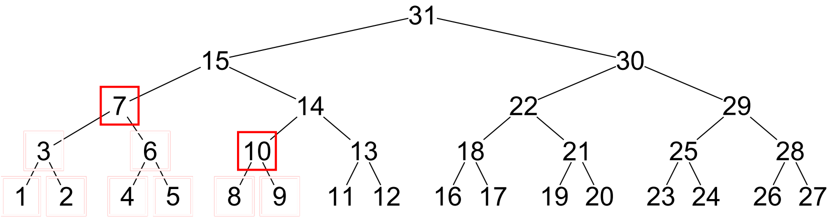
LMS root (RFC8554 app. C)



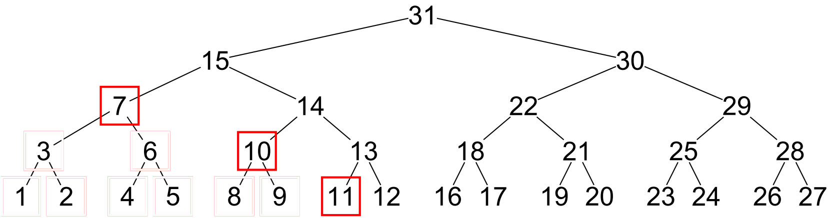
LMS root (RFC8554 app. C)



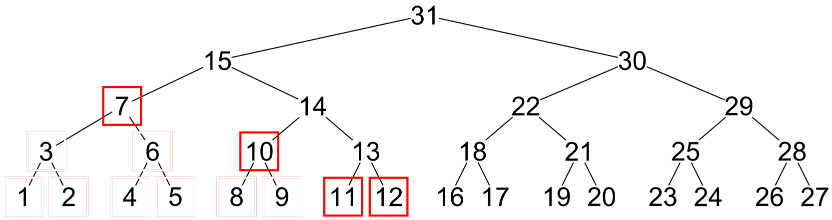
LMS root (RFC8554 app. C)



LMS root (RFC8554 app. C)



LMS root (RFC8554 app. C)



Get to the root with a stack of h -1 hashes! (≤ 768 bytes)

LMS root (RFC8554 app. C)

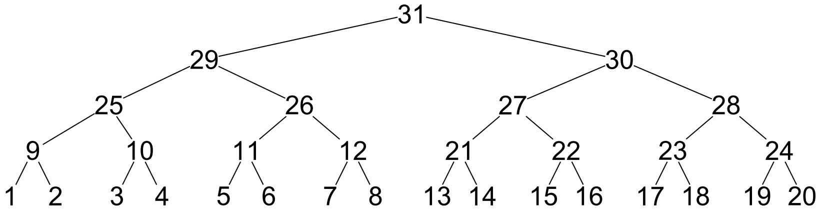
```
// Generating an LMS Public Key from an LMS Private Key

for ( i = 0; i < 2^h; i = i + 1 ) {
  r = i + 2^h;
  temp = H(I || r || "D_LEAF" || OTS_PUB_HASH[i])  // Compute leaf
  j = i;
  while (j % 2 == 1) {
    r = (r - 1) / 2;
    j = (j - 1) / 2;
    left = pop(data stack);
    temp = H(I || r || "D_INTR" || left || temp)  // Compute branch
  }
  push temp onto the data stack
}
public_key = pop(data stack)
```

SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

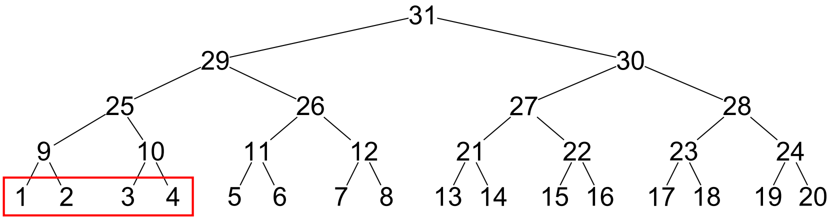
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

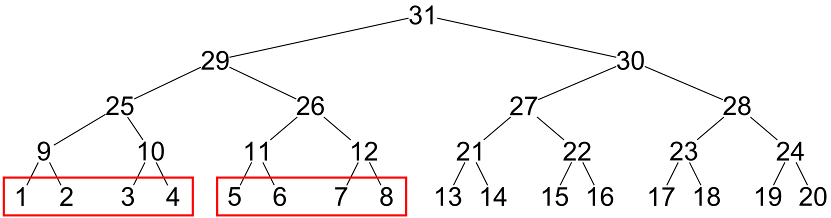
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

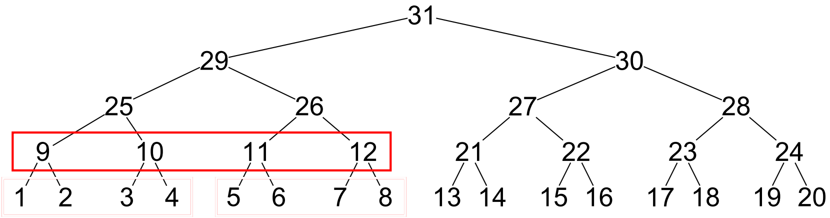
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

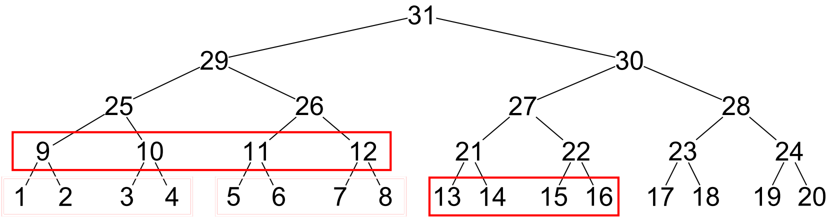
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

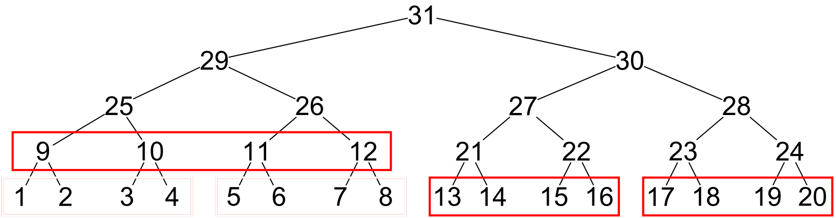
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

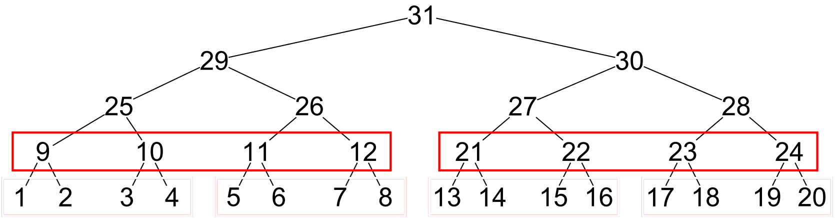
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

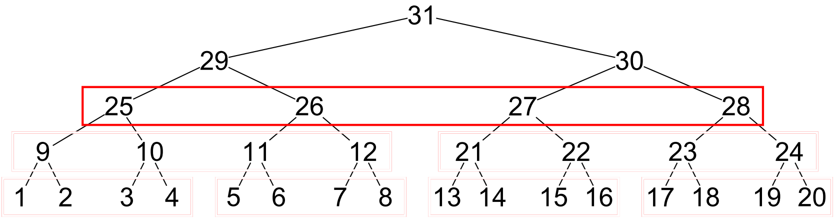
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

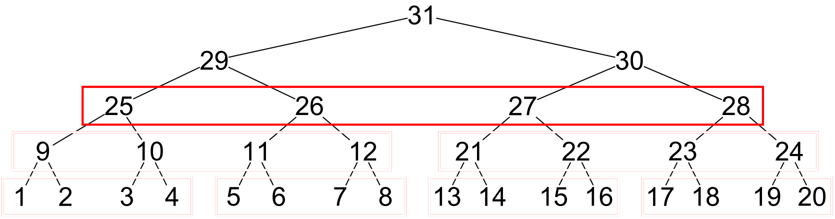
As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



SIMD LMS root (`LANES = 4`)

Stack = vector of arrays of `LANES` nodes.

As soon as $2 \times$ `LANES` neighbour nodes are available, hash them into `LANES` nodes.



Get to level log($\mathtt{LANES}$) with a stack of $(h-1) \times$ LANES SIMD calls

```
for ( i = 0; i < 2^h; i = i + LANES ) {
  r = i + 2^h;
  temp = H(
    I || r + 0..LANES || "D_LEAF" || OTS_PUB_HASH[0..LANES]
  )
  j = i / LANES;
  while (j % 2 == 1) {
    r = (r - LANES) / 2;
    j = (j - LANES) / 2;
    left = pop(data stack);
    temp = H(
      I || r + [0..LANES] || "D_INTR" || left || temp[0..LANES]
    )
  }
  push temp onto the data stack
}
// Compute levels [0..log(LANES)]
```

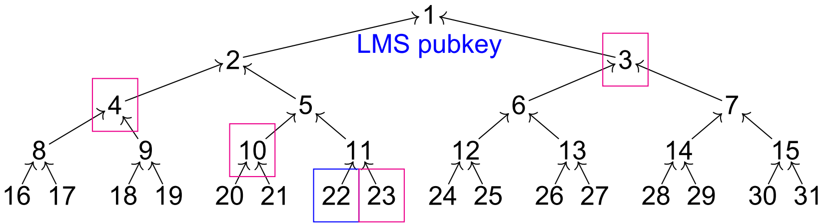
SIMD LMS KeyGen

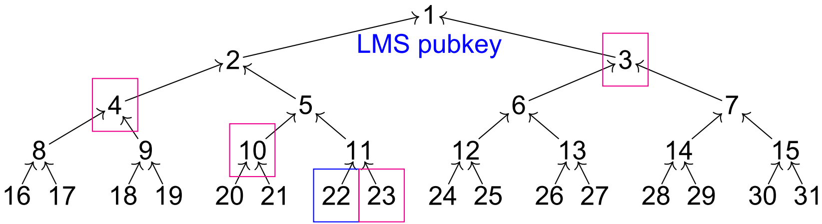
$$\boxed{\text{LM-OTS key}} \xrightarrow{\times 2^{h} \ \ (\leq 2^{25})} \boxed{\text{LMS key}} \xrightarrow{\times k \ \ (\leq 8)} \boxed{\text{HSS key}}$$

SIMD LMS KeyGen

$$
\begin{tikzcd}
\boxed{\text{LM-OTS key}} \arrow[r, "\times 2^{h} \quad (\leq 2^{25})"] & \boxed{\text{LMS key}} \arrow[r, "\times k \quad (\leq 8)"] & \boxed{\text{HSS key}}
\end{tikzcd}
$$

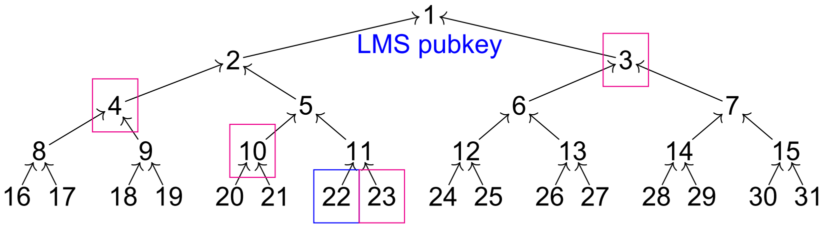
$$\left\lceil \frac{34}{\mathtt{LANES}\cdot\mathtt{THREADS}} \right\rceil \cdot 255 + 34 \qquad \left\lceil \frac{2^{h+1}}{\mathtt{LANES}\cdot\mathtt{THREADS}} \right\rceil \qquad \left\lceil \frac{k-1}{\mathtt{THREADS}} \right\rceil \text{ sigs}$$


1
LMS pubkey
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31



Light, slow

Remember the state and seed

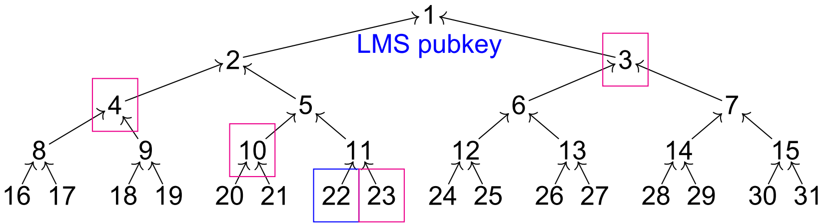


Light, slow

Remember the state and seed

Heavy, fast

Remember **everything**



Light, slow

Remember the state and seed

Heavy, fast

Remember **everything**

`Everything` **=** $2^{h+1} \times 32$ **bytes (**≤ 2.14 **GB)**

Node lifetime

$$
\begin{aligned}
\texttt{life}(h, 0) &= [0, 2) & \texttt{(leftmost leaf)} \\
\texttt{life}(h, 2) &= [2, 4) \\
\texttt{life}(h-1, 0) &= [0, 4) \\
\texttt{life}(1, 0) &= [0, 2^h) & \texttt{(left child of root)} \\
\texttt{life}(1, 1) &= [0, 2^h) & \texttt{(right child of root)} \\
&\vdots
\end{aligned}
$$

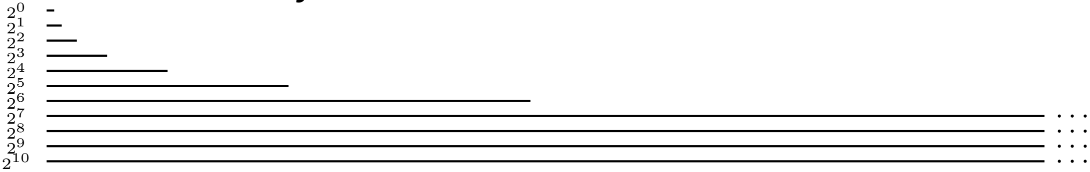
Node lifetime

$$
\begin{aligned}
\texttt{life}(h,0) &= [0,2) && \texttt{(leftmost leaf)} \\
\texttt{life}(h,2) &= [2,4) \\
\texttt{life}(h-1,0) &= [0,4) \\
\texttt{life}(1,0) &= [0,2^h) && \texttt{(left child of root)} \\
\texttt{life}(1,1) &= [0,2^h) && \texttt{(right child of root)} \\
&\vdots
\end{aligned}
$$

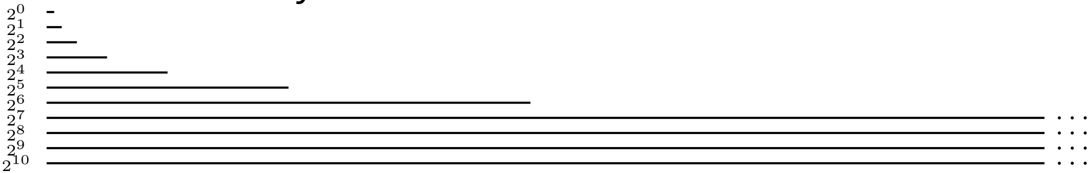
$$
\boxed{\texttt{life}(l,i) = \left[2^{h-l+1}\left\lfloor i/2 \right\rfloor, 2^{h-l+1}\left\lceil (i+1)/2 \right\rceil\right)}
$$

At level $l \in \{1, \ldots, h\}$, node $i \in \{0, \ldots, 2^l - 1\}$ lives during $2^{h-l+1} - 1$ signatures

Small-Memory LM Schemes



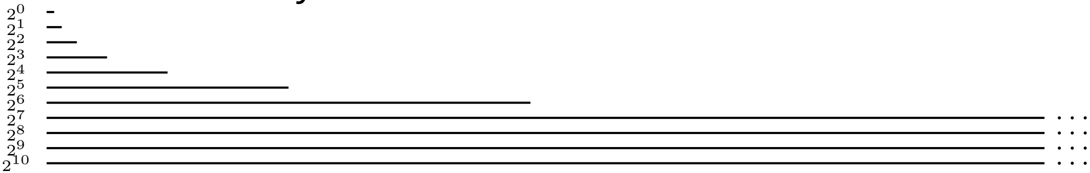
Small-Memory LM Schemes



$N^{1/2}$ algorithm

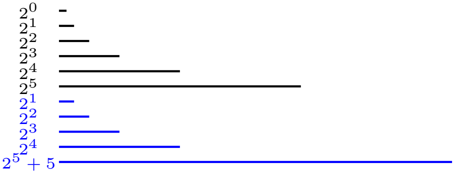
Remember top $h/2$ levels entirely. On level $l > h/2$, remember nodes $\{0, \ldots, 2^{l-h/2}\}$. Remember leaves $\{0, \ldots, 2^{h/2} + h/2\}$.

Small-Memory LM Schemes



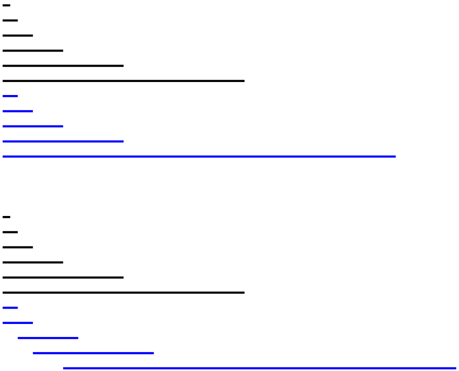
$N^{1/2}$ algorithm

Remember top $h/2$ levels entirely. On level $l > h/2$, remember nodes $\{0, \ldots, 2^{l-h/2}\}$.
Remember leaves $\{0, \ldots, 2^{h/2} + h/2\}$.



Small-Memory LM Schemes

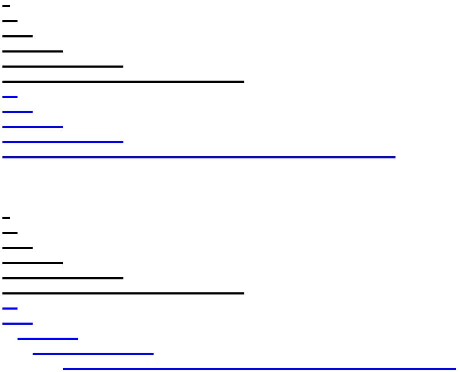
Slide windows after signing



At signature k, compute one leaf and upper branches as possible. Forget leaves left of $L := k - 2^{h/2} - h/2$ and nodes left of $L/2^{h-l}$.

Small-Memory LM Schemes

Slide windows after signing



At signature k, compute one leaf and upper branches as possible. Forget leaves left of $L := k - 2^{h/2} - h/2$ and nodes left of $L/2^{h-l}$.

State

"State" = counter $+$ cached nodes.

2.14 GB $\rightarrow$ **1 MB** for $h = 25$ and SHA-256

All together!

SHA-256 in SIMD is easy!

KeyGen

- Use SIMD/multithreading to compute leaves
- Use SIMD/multithreading to get to the root faster
- Remember node windows according to $N^{1/2}$ algorithm

All together!

SHA-256 in SIMD is easy!

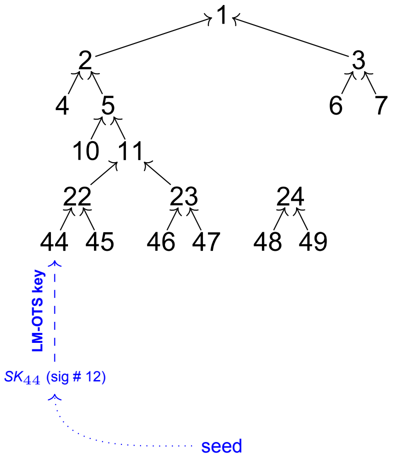
KeyGen

- Use SIMD/multithreading to compute leaves
- Use SIMD/multithreading to get to the root faster
- Remember node windows according to $N^{1/2}$ algorithm

Sign

- Use SIMD/multithreading to compute one leaf ($\lceil 34/(L \cdot T) \rceil \cdot 255 + 34$ calls)
- Compute at most $h/2$ branches
- Forget nodes left nodes past their lifetime
- Release signature AFTER

Thank you!


1
2
3
4
5
6
7
10
11
22
23
24
44
45
46
47
48
49
LM-OTS key
SK_{44} (sig # 12)
seed

LMS: Faster key generation, lighter keys

Francisco José Vial-Prado

Post-Quantum Cryptography Conference

7-8 November, 2023, Amsterdam (NL)

Fortanix

Post-Quantum Cryptography Conference

PKI Consortium