Selected Areas in Cryptology
Cryptanalysis
Week 5

Marc Stevens

stevens@cwi.nl

https://homepages.cwi.nl/~stevens/mastermath/

Asymmetric from symmetric cryptography

- Can we build asymmetric cryptography from symmetric cryptography?

- Benefits:
  - Symmetric cryptography seems generally to resist quantum cryptanalysis
  - No number-theoretic assumptions that might

- This week:
  - Hash-based signatures

Hash-Based Signatures (HBS)

- Entirely based on cryptographic hash functions
  - Single assumption for signatures as we need hash functions anyway!

- Core concept:
  - private key contains hash pre-images
  - signatures selectively reveal these pre-images

- Most HBS are *one-time*: reusing a private key breaks security!
- Few-time HBS: can reuse private key a few times
- Build many-time HBS from many one-time/few-time HBS!

Cryptographic Hash Functions

Hash function standards $H: \{0,1\}^* \rightarrow \{0,1\}^n$:

- MD5: 128-bits hash function published in 1992
  - Widely used till ~~2010
  - Broken in 2004: first collision found [WY05],

- SHA-1: 160-bit hash function published in 1995
  - Widely used even today (TLS1.2, Git, …)
  - 'Broken' in 2005: first theoretical collision attack [WYY05]
    practical attack in 2017: first collision [SBKAM17]

- SHA-2 family: 224/256/384/512-bit hash functions published in 2001

- SHA-3 family: 224/256/384/512-bit hash functions published in 2015

Cryptographic hash functions

Fixed n-bit hash functions: $f: \{0,1\}^* \rightarrow \{0,1\}^n$

- aPre: always pre-image resistance:
  - Given random $h \leftarrow \{0,1\}^n$ find M s/t $f(M) = h$
- aSec: always second pre-image resistance:
  - Given random $M \leftarrow \{0,1\}^{\leq n}$ find $M' \neq M$ s/t $f(M) = f(M')$

- Secure if there is no attack faster than a generic attack
  - Classically:
    - aPre / aSec: brute force search: $O(2^n)$
    - Coll: birthday search: $O(2^{n/2})$
  - Quantum:
    - aPre / aSec: Grover: $O(2^{(n+k)/2})$ on 2^k quantum computers
    - Coll: [CNS'17]: $O(2^{2(n+k)/5})$ on 2^k quantum computers
  - Cryptographic hash functions can be quantum-safe for sufficiently large output size: 256 bits

Preliminaries

- Consider input distributions X_1, X_2,…, X_k, and an index B over $\{1,2, \dots, k\}$
- For a PPT algorithm A, a given input distribution implies an output distribution
  - $Y_i = A(X_i)$
- Consider X_i and X_j being sampled in different ways but their distributions are identical
  - Then $Y_i = A(X_i) = A(X_j) = Y_j$
  - But then $Y_B = A(X_B)$ also independent from B
  - Thus can even view B as being chosen *after* A is run

Preliminaries

- If X_i and X_j have different distributions then Y_i and Y_j may or may not be identical distributed
  - Always be careful in changing an algorithm's input distribution: it's success probability might change!
  - But also: if output distribution changes significantly then it can thus distinguish between X_i and X_j

  - Distinguishing advantage: $\mathrm{Adv}^{A}(\mathrm{X_i}, \mathrm{X_j}) = |\Pr[A(X_i) = 1] - \Pr[A(X_j) = 1]|$

- Some security properties are defined as distinguishing games:
  - $X_{n,1}$ represents a cryptographic scheme: say the hash output $f_n(x)$ of a randomly chosen message x
  - $X_{n,2}$ a simplified idealized scheme: say a random bitstring of the same length as $f_n(x)$

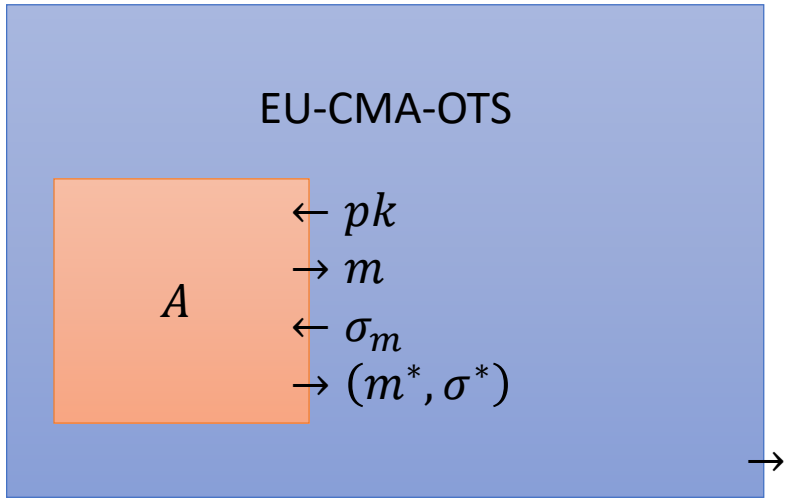
  - The formal definition that the cryptographic scheme behaves as the simplified idealized scheme:
    - for all PPT algorithms A: $\mathrm{Adv}^{A}(\mathrm{X_{n,1}}, \mathrm{X}_{n,2}) \in \mathrm{negl}(n)$ (viewing it as a function in $n \in \mathbb{N}$)

  - E.g.: formal definition that hash outputs "should look like random bitstrings"
  - But be careful of more information: knowing the hash function and preimage, distinguishing is easy!

One-Time Signatures

- Signature Scheme consists of 3 algorithms
  - $(sk, pk) \leftarrow \text{keygen}(1^\lambda)$: generates private/public key pair for security parameter λ
  - $\sigma \leftarrow \text{sign}(sk, m)$: generates signature σ for message m with private key sk
  - $b \leftarrow \text{verify}(pk, m, \sigma)$: verifies signature σ for message m with public key pk
- A *One-Time Signature* (OTS) Scheme only allows to call sign once
- EU-CMA: Existential Unforgeability under adaptive Chosen Message Attack
  - Attacker succeeds when it generates any forgery
    - (m^*, σ^*) for which $\text{verify}(pk, m^*, \sigma^*) = 1$
    - But m^* must not have been signed by user before
  - Adaptive Chosen Message Attack:
    - Allowed to query k signatures adaptively
    - General: $k(n) \in \text{poly}(n)$
    - OTS: $k = 1$
      (wlog attacker queries exactly 1 signature)
  - Scheme is EU-CMA secure i.f.f. for all PPT attackers A: $\Pr[A \text{ succeeds}] \in \text{negl}(\lambda)$

EU-CMA Game



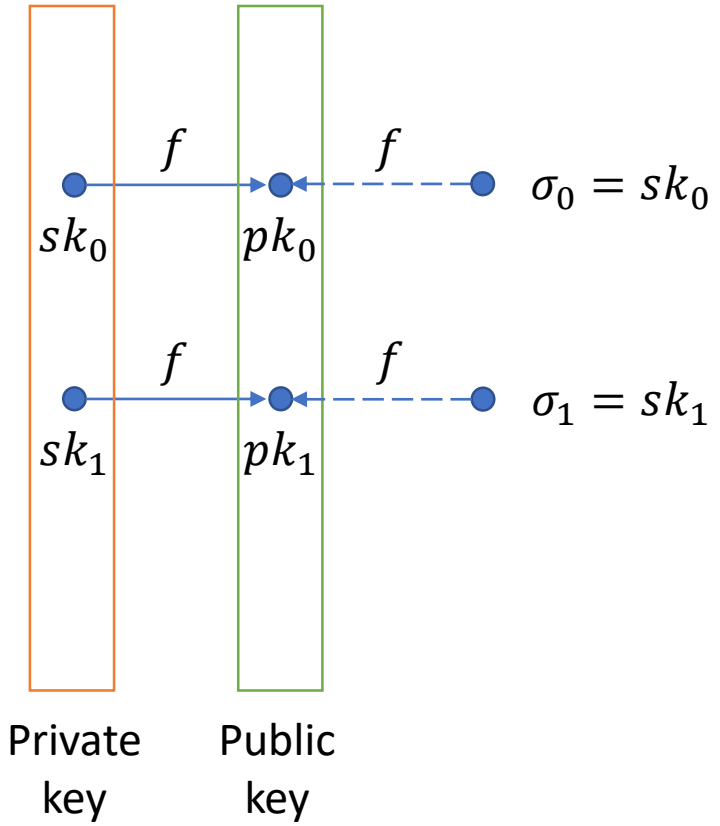
→ 1 if $m^* \neq m$ and
$\text{verify}(pk, m^*, \sigma^*) = 1$
0 otherwise

Game $G_{\text{EU-CMA-OTS}}(1^n, S, A)$:

1. $(sk, pk) \leftarrow \text{S.keygen}(1^n)$
2. $m \leftarrow A(1^n, pk)$
3. $\sigma_m \leftarrow \text{S.sign}(sk, m)$
4. $(m^*, \sigma^*) \leftarrow A(1^n, \sigma_m)$
5. Return 1 if
   $m^* \neq m \ \wedge \ \text{S.verify}(pk, m^*, \sigma^*) = 1$
6. Return 0 otherwise

Lamport OTS
1-bit messages

Lamport OTS 1-bit message



Make *either* sk_0 or sk_1 public as signature for 0 resp. 1.

Lamport OTS – 1-bit message

- Lamport OTS (One-Time Signature) for 1-bit message
  - Private key: sample $(r_0, r_1) \leftarrow \{0,1\}^{n \times 2}$
  - Public key: $(pk_0, pk_1) = (f(r_0), f(r_1))$
  - Signing: sign $m \in \{0,1\}$: output $\sigma_m = r_m$
  - Verification: Verify $f(\sigma_m) = pk_m$
  - Limited to signing 1 message only of 1 bit only

- Security of Lamport OTS is based on security of underlying hash function f:
  - (always) pre-image resistance: hard to find preimage for a randomly chosen hash
  - Undetectability: hash outputs $f(x)$ are indistinguishable from randomly chosen bitstrings (for randomly generated x which is unknown to attacker)

Lamport OTS – 1-bit message

- (always) pre-image resistance:
  - A computational puzzle game
  - Outputs 1 only if A finds solution
  - This may only happen with negligible probability:
  - Pre-secure iff $\Pr[G_{\mathrm{Pre}}(1^n, f_n, A) = 1] \in \mathrm{negl}(n)$

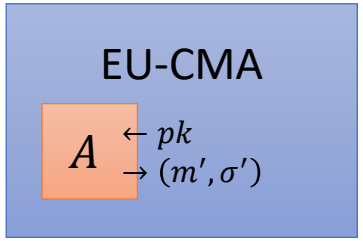
Game $G_{\mathrm{Pre}}(1^n, f_n, A)$:
1. $h \leftarrow \{0,1\}^n$
2. $x \leftarrow A(1^n, f_n, h)$
3. Return 1 if $f_n(x) = h$
4. Return 0 otherwise

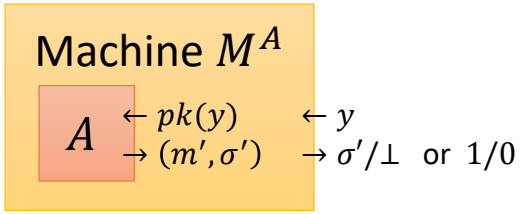
- Undetectability:
  - A distinguishability / guessing game
  - A must not be able to distinguish with non-negligibly probability
  - UD-secure iff $|\Pr[G^0_{\mathrm{UD}}(1^n, f_n, A) = 1] - \Pr[G^1_{UD}(1^n, f_n, A) = 1]| \in \mathrm{negl}(n)$

Game $G^g_{\mathrm{UD}}(1^n, f_n, A)$:
1. $x \leftarrow \{0,1\}^n$
2. If $g = 1$ then: $h = x$
3. Else: $h = f_n(x)$
4. Return $A(1^n, f_n, h)$

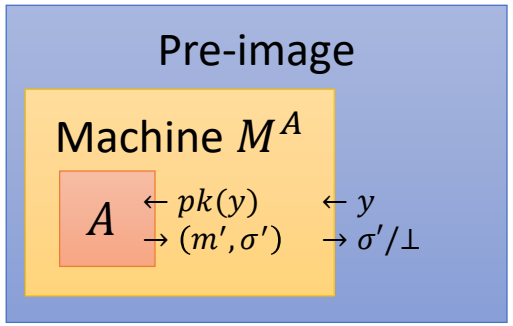
Proof structure

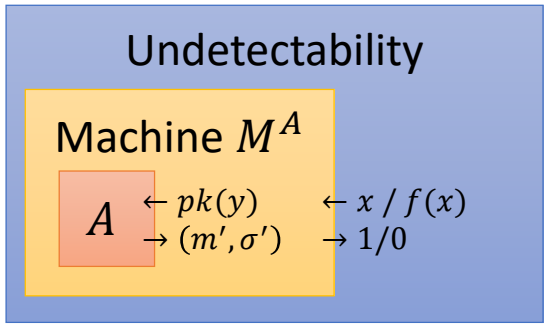
- So given PPT A for Game EU-CMA we're going to embed it in a machine M^A





- This machine will embed a given hash value y in the public key $pk(y)$:
  - it will try to use A to find a preimage σ' for y: $f(\sigma') = y$ or otherwise it returns $\perp$
- We can use this machine M^A as adversary for Pre-image & Undetectability





- By Pre-image resistance assumption it will have negligible success probability for Pre-image
- For Undetectability there are 2 possible inputs:
  - $y = x$: now A's input distribution is identical to Pre-image ⇒ has negligible success prob.
  - $y = f(x)$: now A's input distribution is identical to EU-CMA ⇒ contradiction if non-negligible prob.

Lamport OTS – 1-bit message

- Security proof by contradiction
  - Given PPT A that solves EU-CMA forgery game with probability $p_{\text{forg}} \notin \text{negl}(n)$
  - Build Machine M^A
  - Idea: embedded pre-image challenge y for message $m = b$ in public key pk
    - A successful forgery for $m = b$ should give a valid pre-image
    - If A asks for signature of $m = b$ then we need to abort: we don't know a valid signature

Game $G_{\text{EU-CMA-OTS}}(1^n, f_n, A)$:
1. $(sk, pk) \leftarrow \text{keygen}(1^n)$
2. $m \in \{0,1\} \leftarrow A(1^n, f_n, pk)$
3. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
4. Return 1 if $m' \neq m \wedge f_n(\sigma^*) = pk_{m^*}$
5. Return 0 otherwise

Machine $M^A(1^n, f_n, y)$:
1. $(sk, pk) \leftarrow \text{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, pk_b = y$
3. $m \in \{0,1\} \leftarrow A(1^n, f_n, pk)$
4. If $m = b$ then return $\perp$
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If $f(\sigma^*) = y$ then return σ^*
7. Else return $\perp$

Lamport OTS – 1-bit message

M^A is designed as an adversary against Pre:

- $p_{\mathrm{Pre}} := \Pr[G_{\mathrm{Pre}}(1^n, f_n, M^A) = 1] = \Pr[M^A \neq \perp]$
- By Pre assumption: $p_{\mathrm{Pre}} \in \mathrm{negl}(n)$

Machine $M^A(1^n, f_n, y)$:

1. $(sk, pk) \leftarrow \mathrm{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, pk_b = y$
3. $m \in \{0,1\} \leftarrow A(1^n, f_n, pk)$
4. If $m = b$ then return $\perp$
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If $f(\sigma^*) = y$ then return σ^*
7. Else return $\perp$

Note that σ^* is a preimage of $y = pk_b$
iff σ^* is a valid forgery for $m^* = b$

Game $G_{\mathrm{Pre}}(1^n, f_n, A)$:

1. $h \leftarrow \{0,1\}^n$
2. $x \leftarrow A(1^n, h)$
3. Return 1 if $f_n(x) = h$
4. Return 0 otherwise

Lamport OTS – 1-bit message

But M^A is also an adversary for UD
(replace output $\perp \rightarrow 0, \sigma^* \rightarrow 1$)

Game $G^g_{\mathrm{UD}}(1^n, f_n, A)$:
1. $x \leftarrow \{0,1\}^n$
2. If $g = 1$ then: $h = x$
3. Else: $h = f_n(x)$
4. Return $A(1^n, f_n, h)$

Machine $M^A(1^n, f_n, y)$:
1. $(sk, pk) \leftarrow \mathrm{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, pk_b = y$
3. $m \in \{0,1\} \leftarrow A(1^n, f_n, pk)$
4. If $m = b$ then return 0 (was $\perp$)
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If $f(\sigma^*) = y$ then return 1 (was σ^*)
7. Else return 0 (was $\perp$)

- $\Pr\left[G^1_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] = p_{\mathrm{Pre}} \in \mathrm{negl}(n)$
  - Returns 1 exactly when it wins game G_{Pre} ⇒ has negligible probability!

- $\Pr\left[G^0_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] = \frac{1}{2} p_{\mathrm{forg}} \notin \mathrm{negl}(n)$
  - Case of $y = f_n(x)$, thus pk is properly formed public-key: essentially EU-CMA-OTS game
  - b does not change input distribution of A in step 3, so m independent of b
  - Thus probability of not having early abort is $\Pr[b \neq m] = 1/2$.
  - Input distribution in step 5 also unchanged, as sk_m distribution is also independent of b
    ⇒ Unchanged forgery probability, conditioned on reaching step 5

- Now $\Pr\left[G^0_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] - \Pr\left[G^1_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] \notin \mathrm{negl}(n)$ contradiction for UD!!

Lamport OTS
k-bit messages

Lamport OTS – k-bit message

- Lamport OTS for k-bit messages
  - Private key: sample $\left(r_{j,0}, r_{j,1}\right)_{j=1}^{k} \leftarrow \{0,1\}^{n\times 2\times k}$
  - Public key: $\left(pk_{j,0}, pk_{j,1}\right)_{j=1}^{k} = \left(f_n\left(r_{j,0}\right), f_n\left(r_{j,1}\right)\right)_{j=1}^{k}$
  - Signing: sign $m = (m_1, \dots, m_k) \in \{0,1\}^k$: output $\sigma_m = \left(r_{j,m_j}\right)_{j=1}^{k}$
  - Verification: Verify $\left(h(\sigma_m[j])\right)_{j=1}^{k} = \left(pk_{j,m_j}\right)_{j=1}^{k}$

- Security proof goes analogously
  - Except we now need to pick random $pk_{j,b}$ to replace with y
  - Already had loss factor of ½ of abort on sign query with $m_j = b$
  - Also means additional loss factor of $\Pr\left[m_j' = b\right]$ since $m' \neq m$, but we actually need $m_j' \neq m_j$

- Note that using hash-then-sign with $k = n$ gives an OTS for arbitrary length messages

Lamport OTS – k-bit message

- Security proof by contradiction
  - Given PPT A that solves forgery game with probability $p_{\text{forg}} \notin \text{negl}(n)$
  - Build Machine M^A
  - Idea: embedded pre-image challenge y in public key entry $pk_{i,b}$
    - A successful forgery with $m_i = b$ should give a valid pre-image: $f(\sigma_i^*) = y$
    - If A asks for signature with $m_i = b$ then we need to abort: we don't know a valid signature

Game $G_{\text{EU-CMA-OTS}}(1^n, f_n, A)$:
1. $(sk, pk) \leftarrow \text{keygen}(1^n)$
2. $m \in \{0,1\}^k \leftarrow A(1^n, f_n, pk)$
3. $(m^*, \sigma^*) \leftarrow A\left(1^n, f_n, \sigma_m = \left(sk_{m_i}\right)_{i=1}^k\right)$
4. Return 1 if $m' \neq m \wedge \text{verify}(pk, m^*, \sigma^*) = 1$
5. Return 0 otherwise

Machine $M^A(1^n, f_n, y)$:
1. $(sk, pk) \leftarrow \text{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, i \leftarrow \{1, \dots, k\}, pk_{i,b} = y$
3. $m \leftarrow A(1^n, f_n, pk)$
4. If $m_i = b$ then return $\perp$
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If not $(m^* \neq m \wedge \text{verify}(pk, m^*, \sigma^*) = 1)$ return $\perp$
7. If $m_i^* = b$ then return σ^* else return $\perp$

Lamport OTS – k-bit message

M^A is designed as an adversary against Pre:

- $p_{\mathrm{Pre}} := \Pr[G_{\mathrm{Pre}}(1^n, f_n, M^A) = 1] = \Pr[M^A \neq \perp]$
- By Pre assumption: $p_{\mathrm{Pre}} \in \mathrm{negl}(n)$

Machine $M^A(1^n, f_n, y)$:

1. $(sk, pk) \leftarrow \mathrm{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, i \leftarrow \{1, \dots, k\}, pk_{i,b} = y$
3. $m \leftarrow A(1^n, f_n, pk)$
4. If $m_i = b$ then return $\perp$
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If not $(m^* \neq m \wedge \mathrm{verify}(pk, m^*, \sigma^*) = 1)$ return $\perp$
7. If $m_i^* = b$ then return σ_i^* else return $\perp$

Note that σ_i^* is a preimage of $y = pk_{i,b}$
iff σ^* is a valid forgery for m^* with $m_i^* = b$

Game $G_{\mathrm{Pre}}(1^n, f_n, A)$:

1. $h \leftarrow \{0,1\}^n$
2. $x \leftarrow A(1^n, h)$
3. Return 1 if $f_n(x) = h$
4. Return 0 otherwise

Lamport OTS – k-bit message

But M^A is also an adversary for UD
(replace output $\perp \rightarrow 0, \sigma^* \rightarrow 1$)

Game $G^g_{\mathrm{UD}}(1^n, f_n, A)$:
1. $x \leftarrow \{0,1\}^n$
2. If $g = 1$ then: $h = x$
3. Else: $h = f_n(x)$
4. Return $A(1^n, f_n, h)$

Machine $M^A(1^n, f_n, y)$:
1. $(sk, pk) \leftarrow \mathrm{keygen}(1^n)$
2. $b \leftarrow \{0,1\}, i \leftarrow \{1, \dots, k\}, pk_{i,b} = y$
3. $m \leftarrow A(1^n, f_n, pk)$
4. If $m_i = b$ then return 0
5. $(m^*, \sigma^*) \leftarrow A(1^n, f_n, \sigma_m = sk_m)$
6. If not $(m^* \neq m \wedge \mathrm{verify}(pk, m^*, \sigma^*) = 1)$ return 0
7. If $m^*_i = b$ then return 1 else return 0

- $\Pr\left[G^1_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] = p_{\mathrm{Pre}} \in \mathrm{negl}(n)$
  - Returns 1 exactly when it wins game G_{Pre} ⇒ has negligible probability!

- $\Pr\left[G^0_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] \geq \frac{1}{2}\frac{1}{k} p_{\mathrm{forg}} \notin negl(n)$
  - Case of $y = f_n(x)$, thus pk is properly formed public-key: essentially EU-CMA-OTS game
  - i, b do not change input (and thus output) distribution of A
  - ⇒ Unchanged forgery probability, except prob ½ of not having early abort
  - And prob $\geq 1/k$ of having solved the challenge, namely when $m^*_i = b$ (conditioned on $m^* \neq m$ and $m_i \neq b$)

- Now $\Pr\left[G^0_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] - \Pr\left[G^1_{\mathrm{UD}}(1^n, f_n, M^A) = 1\right] \notin \mathrm{negl}(n)$ contradiction for UD!!

Lamport OTS – k-bit message

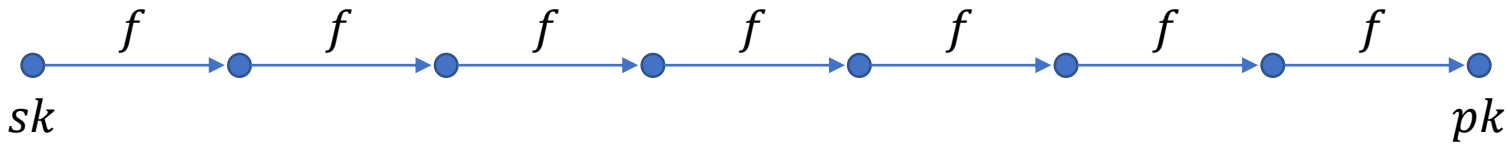
- Lamport has large private/public keys
- For practical purposes typically $k = 256$ then:
  - Private key : 2*256*256 bits = 16KiB
  - Public key : 2*256*256 bits = 16KiB
  - Signature : 1*256*256 bits = 8KiB
- That's for a single OTS key, we want to sign many messages
- Can we do better?

Winternitz OTS
(k-bit messages)

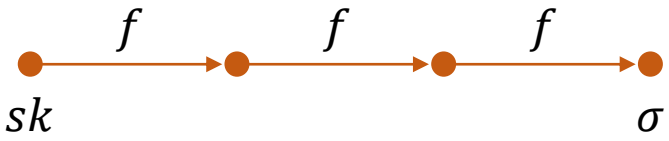
Winternitz OTS (WOTS)

Sign multiple bits at once: $m \in \{0,1,2,3,4,5,6,7\}$

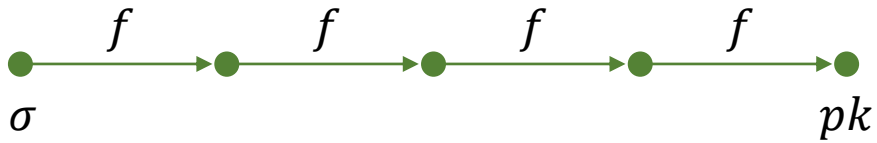
Then need hash chain with 8 points, or 7 calls to f.



$\text{Sign}(sk, 3)$



$\text{Verify}(pk, 3, \sigma)$



Winternitz OTS (WOTS)

- Idea: tradeoff speed for size using *hash chains* of length w
  - Key generation:
    - Choose random $sk \in \{0,1\}^n$
    - Then $pk = f^{w-1}(sk)$ (hash chain of length w)
  - Sign:
    - Given message $m \in \{0, \dots, w-1\}$
    - Output $\sigma = f^m(sk)$
  - Verify:
    - Given message m and signature σ
    - Verify that $f^{w-1-m}(\sigma) \stackrel{?}{=} pk$, since $f^{w-1-m}\left(f^m(sk)\right) = f^{w-1}(sk) = pk$

- To sign k-bit message $m \in \{0,1\}^k$:
  - First interpret as integer: $m \in \mathbb{Z}, 0 \leq m < 2^k$
  - Then write in radix w:

$$m = \sum_j m_j w^j \quad \text{with each} \quad m_j \in \{0, \dots, w-1\}$$

- We need to sign each m_j, thus we need $l_1 = \lceil k / \log_2 w \rceil$ hash chains

Winternitz OTS (WOTS)

- Problem: any signature $\sigma_m = f^m(sk)$ can be modified into signature for larger m:
  - $\sigma_{m+a} = f^{m+a}(sk) = f^a\left(f^m(sk)\right) = f^a(\sigma_m)$

- Winternitz's solution:
  - Add checksum hashchains that necessarily *decrease* for *larger* m_j

$$c = l_1(w-1) - \sum_j m_j$$

then

$$0 \le c < l_1(w-1) + 1$$

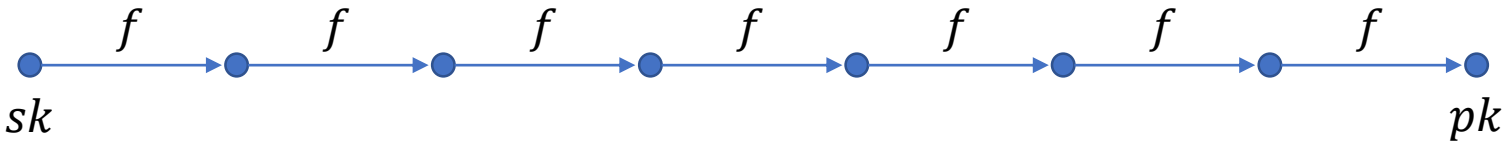
  - Also write c in radix w: $c = \sum_j c_j w^j$ with each $c_j \in \{0, \dots, w-1\}$
  - Need $l_2 = \lceil \log_2(l_1(w-1)+1) / \log_2 w \rceil$ additional hash chains

  - Define function $\mathrm{split(m)} = \left(m_1, \dots, m_{l_1}, c_1, \dots, c_{l_2}\right)$

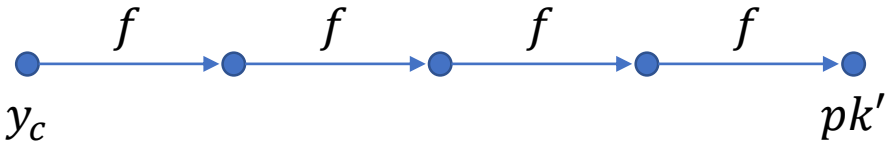
Winternitz OTS (WOTS)

- Security proof:
  - A successful forgery must do a (second-)preimage attack on at least 1 chain
  - Can embed a pre-image challenge in a random chain at a random position, similar to Lamport
  - But now also need hybrid argument for undetectability for hash chains!:
    - In each game replace one hash function call at the beginning of a chain with random bitstring
    - In the first game, all pk_i are properly generated hash chains
    - In the last game, all pk_i are simply randomly chosen bitstrings
    - If the adversary can distinguish between the first and last game with non-negligible probability
    - Then there is at least one game hop, from game i to game $i+1$,
      for which it also has non-negligible probability
    - But distinguishing between game i and game $i+1$ implies breaking UD:
      as one hash function output is replaced by random bitstring ⇒ contradiction!
  - However, a successful forgery can also be created from second pre-images
  - Unfortunately WOTS does not allow to embed a second-preimage challenge
  - And thus instead reduces to Collision resistance

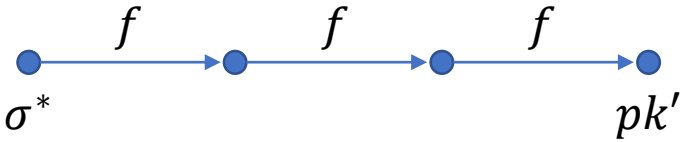
Winternitz OTS (WOTS)



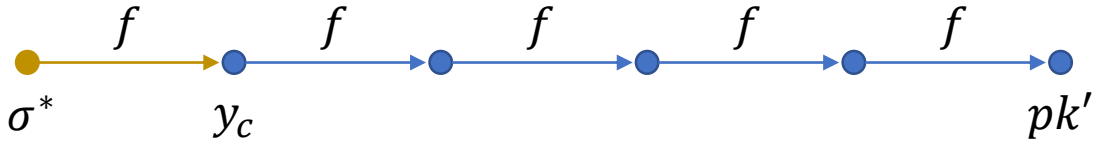
Embedding pre-image challenge y_c



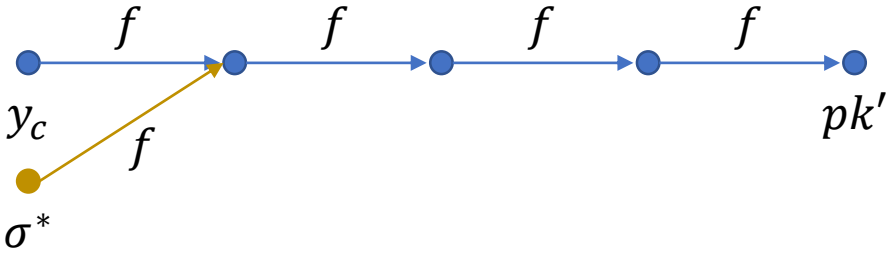
Forgery σ^* that misses embedded challenge



Forgery σ^* with good preimage



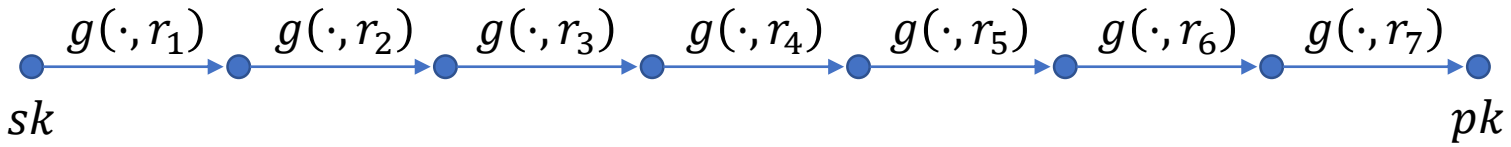
Forgery σ^* with second-preimage
But there was no such challenge!
Instead the reduction can output
collision pair (y_c, σ^*)



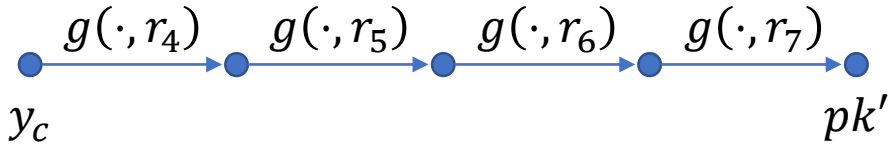
Winternitz OTS+ (WOTS+)

- WOTS+ is a strengthened version of WOTS that does reduce to preimage & second-preimage security

$g(x, r) = f(x \oplus r)$

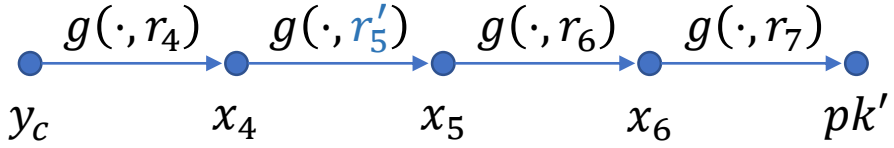


Embedding pre-image challenge y_c



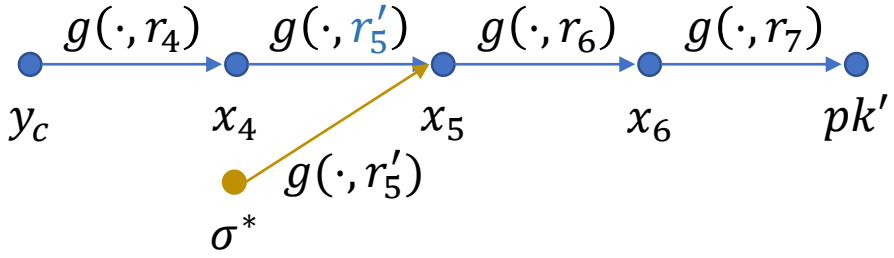
Embedding second pre-image challenge x_c

$r'_5 = x_4 \oplus x_c \;\Rightarrow\; x_5 = f(x_4 \oplus r'_5) = f(x_c)$



Forgery σ^* with good second pre-image

$g(\sigma^*, r'_5) = f(\sigma^* \oplus r'_5) = x_5 = f(x_c)$



Can now output $\sigma^* \oplus r'_5$ as second preimage!

Security proof similar to Lamport, now extended with second pre-image.
But also need hybrid argument for undetectability at each hash chain depth!
For more details see: https://eprint.iacr.org/2017/965.pdf

Winternitz OTS+ (WOTS+)

- WOTS+ is a strengthened version of WOTS that does reduce to preimage & second-preimage security
- Idea:
  - Add random bitstrings $r = (r_1, \dots, r_{w-1}) \in \{0,1\}^{n\times(w-1)}$ to public key
  - Those random bitstrings are used in hash chain in such a way to allow programming second preimage challenges

- Chain definition:
  - $c^i(x, r) = f\left(c^{i-1}(x, r) \oplus r_i\right)$
  - $c^0(x, r) = x$

- Key generation ($l = l_1 + l_2$):
  - $r = (r_1, \dots, r_{w-1}) \leftarrow \{0,1\}^{n\times(w-1)}$
  - $(sk_1, \dots, sk_l) \leftarrow \{0,1\}^{n\times l}$
  - $sk = \left(r, (sk_1, \dots, sk_l)\right)$
  - $pk = \left(r, (c^{w-1}(sk_1, r), \dots, c^{w-1}(sk_l, r))\right)$

- $\mathrm{Sign}(sk, m)$:
  - Let $\left(b_j\right)_{j=1}^{l} \leftarrow \mathrm{split}(m)$
  - $\sigma = \left(c^{b_1}(sk_1, r), \dots, c^{b_l}(sk_l, r)\right)$

- $\mathrm{Verify}\ (pk, m, \sigma)$:
  - Let $\left(b_j\right)_{j=1}^{l} \leftarrow \mathrm{split}(m)$
  - Verify that $c^{w-1-b_i}\left(\sigma_i, r_{[b_i+1,w-1]}\right) = pk_i$ for all $i = 1, \dots, l$

Winternitz OTS+ (WOTS+)

- Winternitz OTS+ can trade more work for smaller keys

- Let $w = 2^4$ (and $k = 256$)
  - Then $l_1 = \frac{256}{4} = 64$ and $l_2 = \lceil \log_2(l_1 \cdot 15 + 1)/4 \rceil = 3$
  - Total number of chains: $l = 64 + 3 = 67$
  - Then private/public key/signature size: $l \cdot 256$ bits = 2144 B ~ 2 KiB
  - Total work: $l \cdot w = 64 \cdot 16 = 1024$

- Let $w = 2^8$ (and $k = 256$)
  - Then $l_1 = \frac{256}{8} = 32$ and $l_2 = \lceil \log_2(l_1 \cdot 255 + 1)/8 \rceil = 2$
  - Total number of chains: $l = 32 + 2 = 34$
  - Then private/public key/signature size: $l \cdot 256$ bits = 1088 B ~ 1 KiB
  - Total work: $l \cdot w = 34 \cdot 256 = 8704$

- Compare to Lamport OTS : Priv/Pub Key: 16 KiB, Signature: 8 KiB

Merkle Trees:
From one-time to many-time

Many-time HBS

- To sign R messages with one-time HBS's, we need R one-time HBS key pairs
- Typically we want $R = 2^{20}, 2^{40}, 2^{60}$

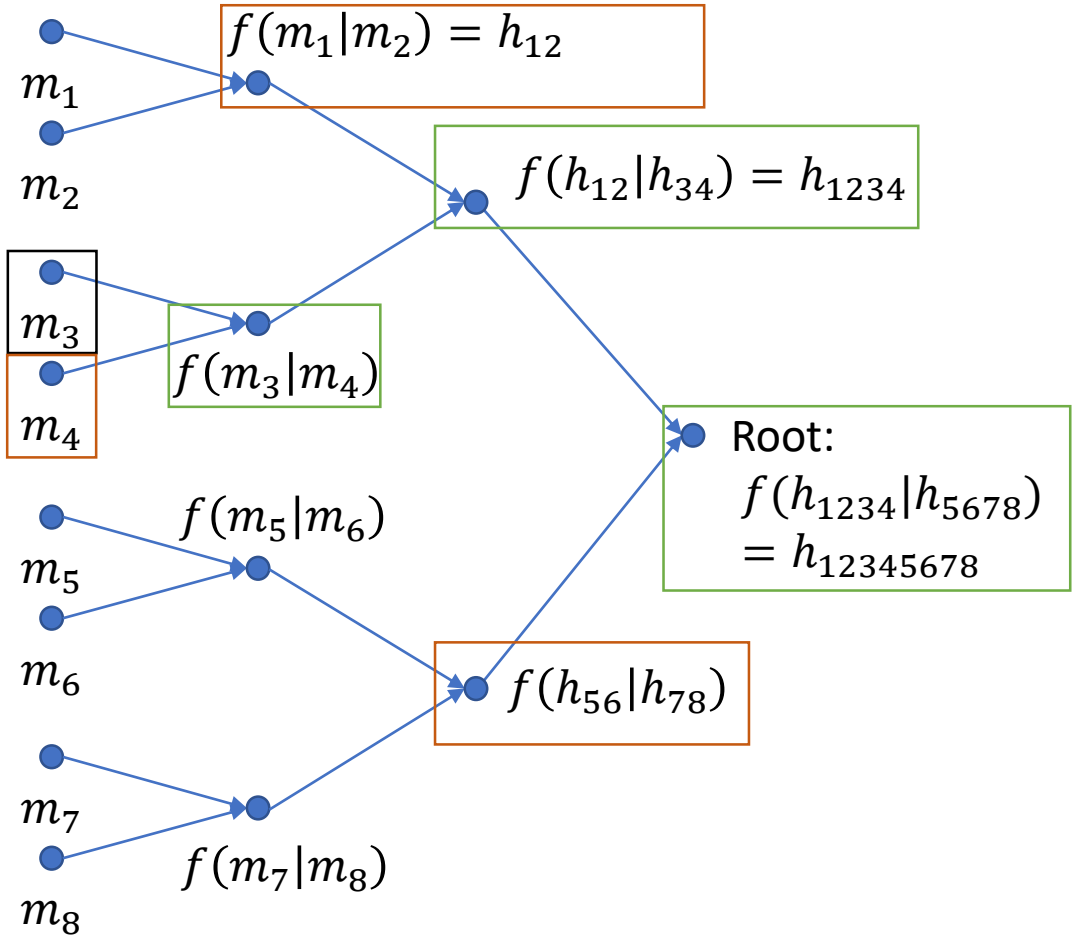
- For WOTS+-256-2^8 that would mean 1MiB, 1TiB, 1EiB of data!

- One solution is to expand private keys from a private seed
  - E.g.: $sk_i^{(r)} \leftarrow f(seed|r|i)$
  - This reduces private key storage to only the seed
  - Individual WOTS+-256-2^8 private keys can generated on demand during signing

- Still need to publish all public keys for public verification!
- Can we do better? Yes: Merkle Trees

Merkle Tree

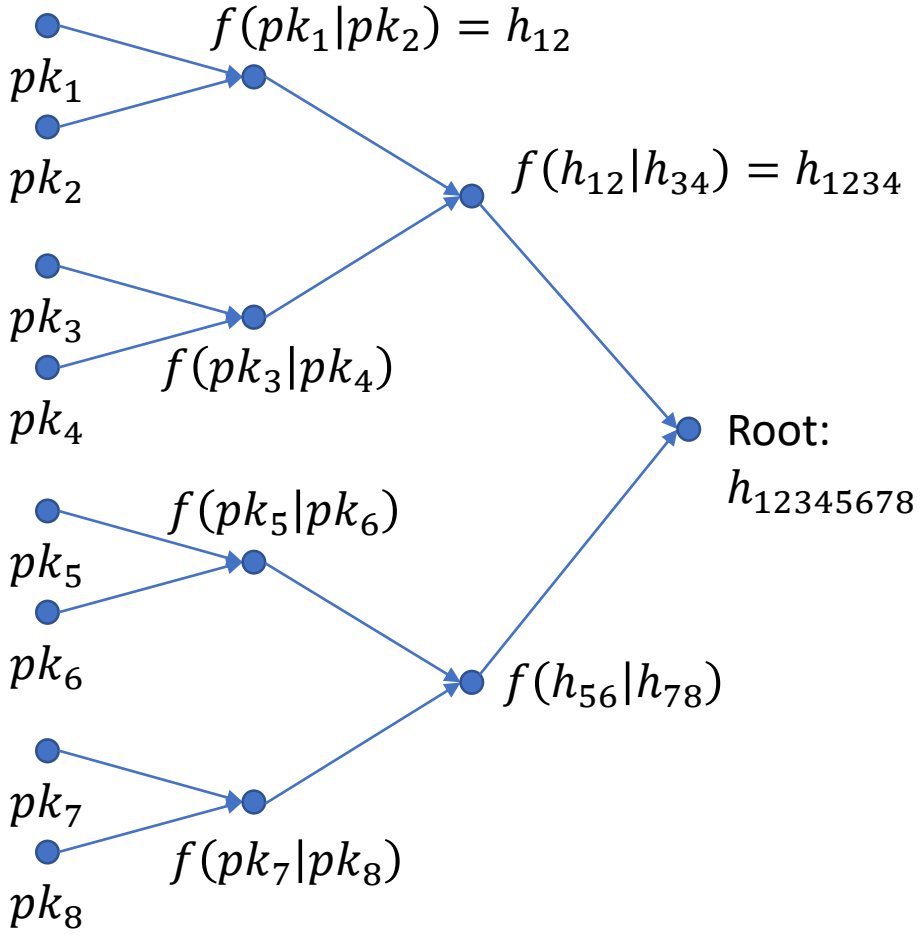
- Merkle Tree is a membership proof using hash functions in a tree structure
- First compute & publish root



- To prove m_3 is in set:
  - Outputs $m_3, (3, m_4, h_{12}, h_{5678})$
  - Note the position 3 defines Path and in which order to concatenate elements
- Verifier computes:
  - Path using the position 3
  - $h'_{34} = f(m_3|m_4)$
  - $h'_{1234} = f(h_{12}, h'_{34})$
  - $h'_{12345678} = f(h'_{1234}, h_{5678})$
  - Verifies $h'_{12345678} =? \ root$
- Security reduces to Pre or Coll (but WOTS+ trick applicable: Coll → SecPre)

Merkle Tree

- *Composite* signature scheme using e.g. WOTS-256-2^8
- Key generation:
  - Generate random $seed$
  - Generate R private keys $sk_1, \dots, sk_R$ from $seed$
  - Compute public keys $pk_1, \dots, pk_R$
  - Secret key: $seed, counter = 1$
  - Public key: Merkle tree root PK over $pk_1, \dots, pk_R$
- Each signature needs to be extended with corresponding public key & membership proof
- Signing a message m
  - Increase $counter$, say now $counter = 3$
  - Generate (sk_3, pk_3) from $seed$
  - Generate signature σ_3
  - Output $\sigma = (\sigma_3, pk_3, (3, pk_4, h_{12}, h_{56789}))$
- Verifying a signature
  - Verify σ_3 for message m under pk_3
  - Verify membership proof of pk_3 to root PK



Merkle Tree

- Let's use WOTS+-256-2^8 then

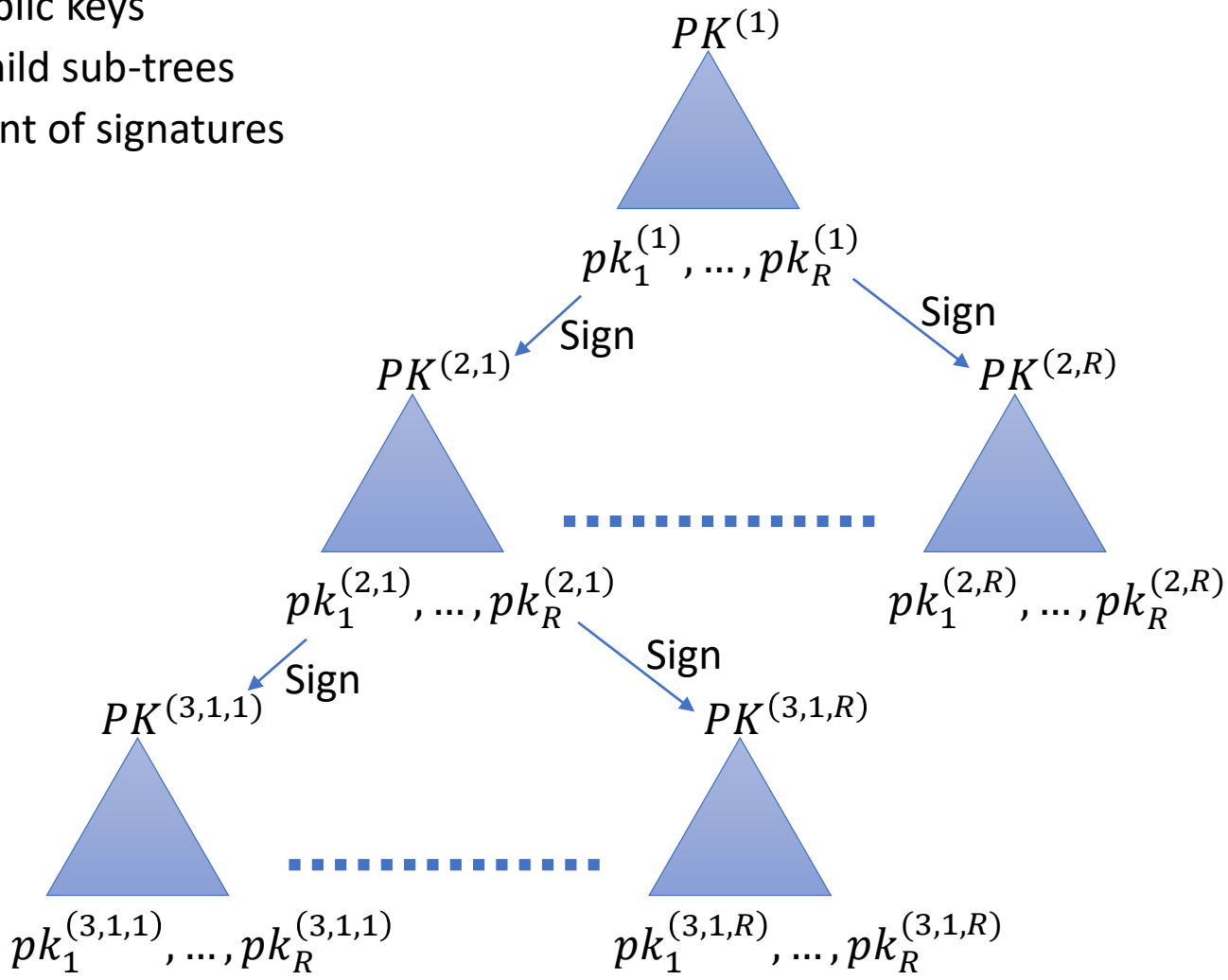
- Private key: 256 + $\lceil \log_2 R \rceil$ bits (seed + index)
- Public key: 256 bits (root)
- Signature: (WOTS+ signature & public key, membership proof)
  - Total: 8704 + 8704 + $\lceil \log_2 R \rceil$ + $\lceil \log_2 R \rceil \cdot 256$ bits
  - $R = 2^{10}$: 2498 B ~ 2.4 KiB
  - $R = 2^{20}$: 2819 B ~ 2.8 KiB

- Total keygen work: $R \cdot 34 + R \cdot 8704 + R$ (sk gen + pk gen + root)
  - Thus total work ~ $2^{13+\log_2 R}$
  - $R = 2^{10} \Rightarrow \text{work} = 2^{23}$ costs less than a second on single CPU
  - $R = 2^{20} \Rightarrow \text{work} = 2^{33}$ costs minutes on single CPU
  - $R = 2^{40} \Rightarrow \text{work} = 2^{53}$ costs decades on single CPU, not usable!
  - $R = 2^{60} \Rightarrow \text{work} = 2^{73}$ really not usable!
- Note that signature work is the same as keygen, but can be optimized by storing hashes on current path.
  - $\Rightarrow$ On average only a few WOTS+ keys need to be computed per signature.

Trees of Trees

Trees of Trees

- Composite signature scheme of composite signature scheme
  - Each sub-tree is 1 Merkle Tree of R WOTS+ public keys
  - Sub-trees are used to sign root public key of child sub-trees
  - Parameter: D depth of tree ⇒ R^D total amount of signatures
- Keygen:
  - Generate $seed$
  - Generate top subtree root $PK^{(1)}$ from $seed$
  - Private key: $seed, counter = 0$
  - Public key: $PK^{(1)}$
- Sign message m:
  - Increase counter
  - Compute path, e.g. $(3,1,5,2)$
  - Generate subtrees on path from $seed$
  - Output Sig. m with subtree leaf $pk_2^{(3,1,5)}$
  - Output Sig. of each child subtree with parent
- Verify:
  - Verify entire path of signatures to $PK^{(1)}$



Trees of Trees

- Let's use MerkleTree-WOTS+-256-2^8 and $R = 2^r$

- Private key: $256 + D \cdot r$ bits (seed + index)
- Public key: 256 bits (root $PK^{(1)}$)
- Signature: $D \cdot (2 \cdot 8704 + r \cdot 257)$ (D MerkleTree-WOTS+-256-2^8 signatures)

- Examples for 2^{60} signatures (private key = 316 bits, public key = 256 bits):
  - $r = 20, D = 3$: signature size: 8456 B ~ 8.3 KiB
  - $r = 10, D = 6$: signature size: 14984 B ~ 14.6 KiB
  - $r = 5, D = 12$: signature size: 28040 B ~ 27.4 KiB

Real World Schemes

- Stateful HBS: need to be really careful maintaining state!
  - XMSS: Based on MerkleTree using WOTS+ (NIST standard)
  - XMSS-MT: Based on Tree of XMSS (NIST standard)
  - LMS: Based on MerkleTree using WOTS (NIST standard)
  - HSS: Based on Tree of LMS (NIST standard)
  - Note these have various tweaks including:
    - Optimized TreeHash algorithm to maintain internal state of current path to prevent signature calls with a lot of update work
    - Extra prefix/suffix per hash call to avoid various attacks

- Stateless HBS: avoid keeping track of state by enabling random paths!
  - SPHINCS+: Based on Trees of MerkleTrees of WOTS+ (NIST standard)
    - Uses FORS instead of WOTS+ at leaf MerkleTree
    - FORS is a few-time signature scheme
    - Number of potential signatures is so large, one can randomly choose path to a HORST instance
    - Even with many signatures, the probability a HORST instance is used too often is negligible

Summary

- Lamport 1-bit and k-bit message OTS
  - $2k$ pre-images as private key, reveal k pre-images selectively based on message
  - Hash function needs to be Pre-secure and UD-secure
- Winternitz(+) OTS
  - Use hashchains to trade-off speed for size by signing multiple bits at once
  - Use extra checksum hashchains to prevent trivial manipulation
  - WOTS+: Hash function needs to be Pre-, Sec-, and UD-secure
- MerkleTree
  - Compact composite public key for many OTS public keys
  - Each signature includes membership proof for used OTS public key
- Trees of Trees
  - Tree of MerkleTrees, Parent Tree sign public key of Child Trees
- These are all Stateful: need to keep track of state or break security!