CIS 5560

Cryptography
Lecture 19

Course website:
pratyushmishra.com/classes/cis-5560-s24/

Slides adapted from Dan Boneh and Vinod Vaikuntanathan

Announcements

- **HW 8 out Wednesday evening**
  - Due **Wednesday Apr 10** at 11:59PM on Gradescope
  - Covers
    - RSA
    - little bit of IND-CCA PKE
- Pratyush's Office Hours moved to 12-1PM tomorrow.
  - (Also will be on zoom)

Recap of last lecture

New primitive: Digital Signatures

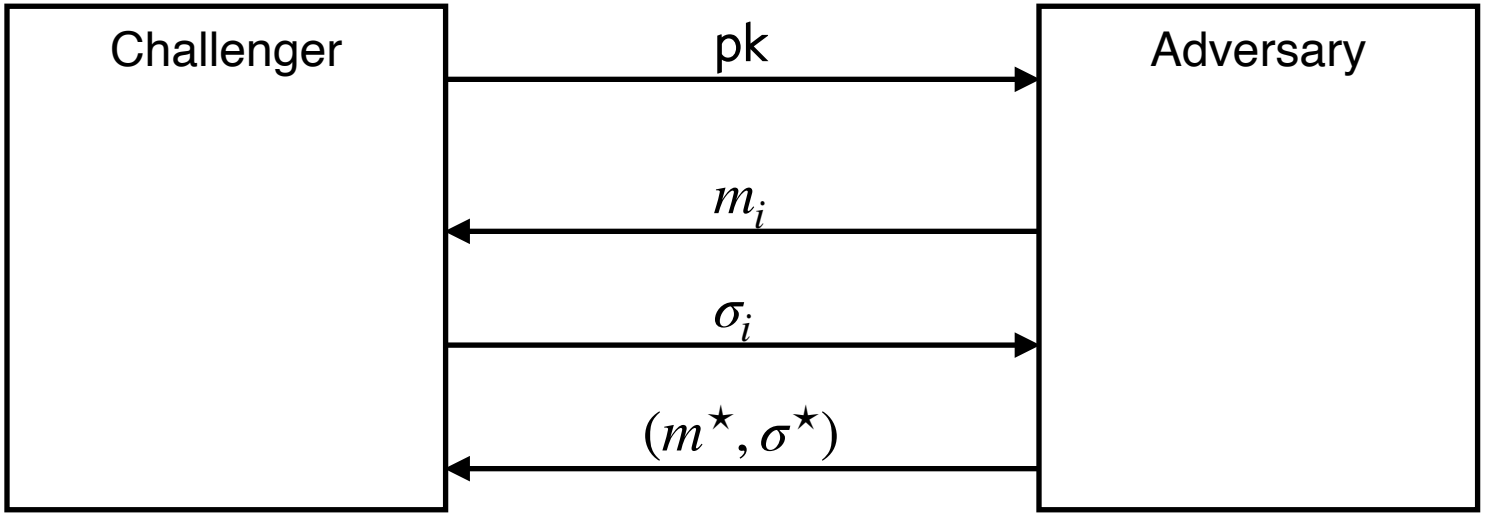
Digital Signatures: Definition

A triple of PPT algorithms (**Gen**, **Sign**, **Verify**) such that

- Key generation: $\mathsf{Gen}(1^n) \rightarrow (\mathsf{sk}, \mathsf{pk})$
- Message signing: $\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$
- Signature verification: $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \rightarrow b \in \{0,1\}$

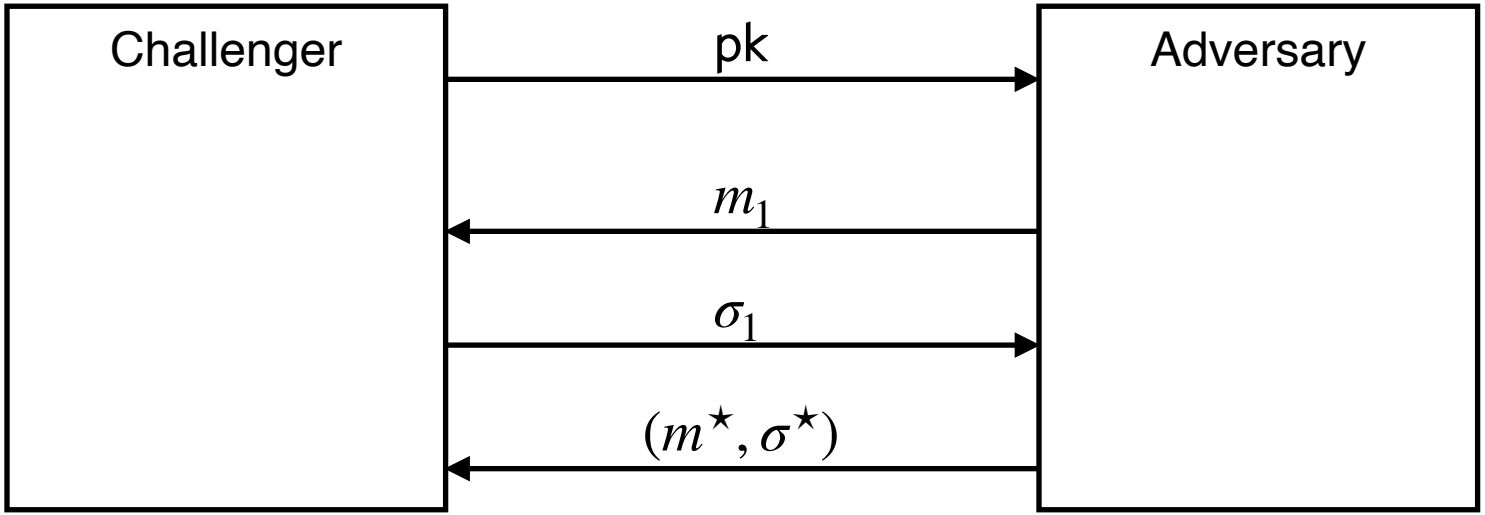
Correctness: For all vk, sk, m: $\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$

EUF-CMA for Signatures



$$\Pr\left[\begin{array}{c} m^\star \notin \{m_i\} \\ \text{and} \\ \mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1 \end{array}\right] = \mathsf{negl}(\lambda)$$

Simpler Goal: EUF-CMA for *1-time Signatures*



$$\Pr\left[\begin{array}{c} m^\star \neq m_1 \\ \text{and} \\ \text{Verify}(\text{pk}, m^\star, \sigma^\star) = 1 \end{array}\right] = \text{negl}(\lambda)$$

Lamport (One-time) Signatures from OWFs

Signing Key sk: $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$

Public Key pk: $\begin{pmatrix} y_0 = f(x_0) \\ y_1 = f(x_1) \end{pmatrix}$

Signing a bit b: The signature is $\sigma = x_b$

Verifying (b, σ): Check if $f(\sigma) = y_b$

Claim: Assuming f is a OWF, no PPT adversary can produce a signature of $\bar{b}$ given a signature of b.

Lamport (One-time) Signatures for n bits

Secret Key sk: $\begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ x_{1,1} & x_{1,1} & \cdots & x_{n,1} \end{pmatrix}$

Public Key pk: $\begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{n,1} \end{pmatrix}$ where $y_{i,b} = f(x_{i,b})$.

Signing $m = (m_1, \ldots, m_n)$: $\sigma = (x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n})$

Claim: Assuming f is a OWF, no PPT adv can produce a signature of m given a signature of a single $m' \neq m$.

Claim: Can forge signature on any message given the signatures on (some) two messages.

Lamport (One-time) Signatures for arbitrary bits

Secret Key sk: $\begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ x_{1,1} & x_{1,1} & \cdots & x_{n,1} \end{pmatrix}$

Public Key pk: $\begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{n,1} \end{pmatrix}$ where $y_{i,b} = f(x_{i,b})$.

Signing m:

1. $z := H(m)$
2. $\sigma = (x_{1,z_1}, x_{2,z_2}, \ldots, x_{n,z_n})$

Claim: Assuming H is CRH and f is a OWF, no PPT adv can produce a signature of m given a signature of a single $m' \neq m$.

Claim: Can forge signature on any message given the signatures on (some) two messages.

Today's Lecture

- **1-time signatures → many-time signatures**
- **Signatures from CDH**
- **Signatures from TDFs**

Constructing a Signature Scheme

Step 0. Still one-time, but arbitrarily long messages.

Step 1. Many-time: Stateful, Growing Signatures.

Step 2. How to Shrink the signatures.

Step 3. How to Shrink Alice's storage.

Step 4. How to make Alice stateless.

Step 5 (*optional*). How to make Alice stateless and deterministic.

So far, only one-time security…

Constructing a Signature Scheme

Theorem [Naor-Yung'89, Rompel'90]
(EUF-CMA-secure) Signature schemes exist assuming that one-way functions exist.

TODAY:
(EUF-CMA-secure) Signature schemes exist assuming that collision-resistant hash functions exist.

(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

Step 3. How to Shrink Alice's storage.
Idea: ***Pseudorandom Trees***

Step 4. How to make Alice stateless.
Idea: ***Randomization***

Step 5 (*optional*). How to make Alice stateless and deterministic. Idea: ***PRFs.***

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key sk_0

When signing a message m_1:

Generate a new pair $(\mathsf{sk}_1, \mathsf{pk}_1)$

Produce signature $\sigma_1' \leftarrow \mathsf{Sign}(\mathsf{sk}_0, m_1 || \mathsf{pk}_1)$

Output $\mathsf{pk}_1 || \sigma_1'$.

Remember $\mathsf{pk}_1 || m_1 || \sigma_1$ as well as sk_1.

To verify a signature $\mathsf{pk}_1 || \sigma_1$ for message m_1:

Run $\mathsf{Verify}(\mathsf{pk}_0, \mathsf{pk}_1 || m_1, \sigma_1') = 1$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key sk_0

When signing a message m_1:

Generate a new pair $(\mathsf{sk}_1, \mathsf{pk}_1)$

Produce signature $\sigma_1 \leftarrow \mathrm{Sign}(\mathsf{sk}_0, m_1 || \mathsf{pk}_1)$

Output $\mathsf{pk}_1 || \sigma_1$.

Remember $\mathsf{pk}_1 || m_1 || \sigma_1$ as well as sk_1.

$$\mathsf{pk}_0 \xrightarrow{\sigma_1 \quad m_1} \mathsf{pk}_1$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key sk_0

When signing the next message m_2

Generate a new pair $(\mathsf{sk}_2, \mathsf{pk}_2)$

Produce signature $\sigma_2 \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m_2 \,||\, \mathsf{pk}_2)$

Output ???

$$\mathsf{pk}_0 \xrightarrow{\sigma_1 \quad m_1} \mathsf{pk}_1$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key sk_0

When signing the next message m_2

Generate a new pair $(\mathsf{sk}_2, \mathsf{pk}_2)$

Produce signature $\sigma_2 \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m_2 || \mathsf{pk}_2)$

Output $\mathsf{pk}_2 || \sigma_2$??

$$\mathsf{pk}_0 \xrightarrow{\sigma_1 \quad m_1} \mathsf{pk}_1$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

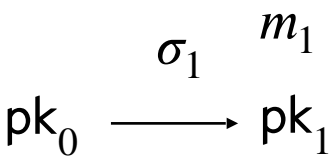
Alice starts with a secret signing Key sk_0

When signing the next message m_2

Generate a new pair $(\mathsf{sk}_2, \mathsf{pk}_2)$

Produce signature $\sigma_2 \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m_2 || \mathsf{pk}_2)$

Output $\mathsf{pk}_1 || \mathsf{pk}_2 || \sigma_2$??



Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice starts with a secret signing Key sk_0

When signing the next message m_2

Generate a new pair $(\mathsf{sk}_2, \mathsf{pk}_2)$

Produce signature $\sigma_2 \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m_2 || \mathsf{pk}_2)$

Output $(\mathsf{pk}_1 || m_1 || \sigma_1) || \mathsf{pk}_2 || \sigma_2$

(additionally) remember $\mathsf{pk}_2 || m_2 || \sigma_2$ as well as sk_2.

$$\mathsf{pk}_0 \xrightarrow{\sigma_1, m_1} \mathsf{pk}_1 \xrightarrow{\sigma_2, m_2} \mathsf{pk}_2$$

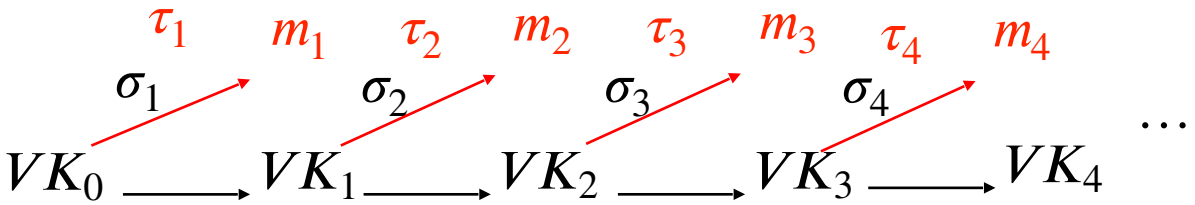
Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Two major problems:

1. Alice is *stateful*: Alice needs to remember a whole lot of things, $O(T)$ information after T steps.

2. The *signatures grow*: Length of the signature of the T-th message is $O(T)$.



(Many-time) Signature Scheme

In four+ steps

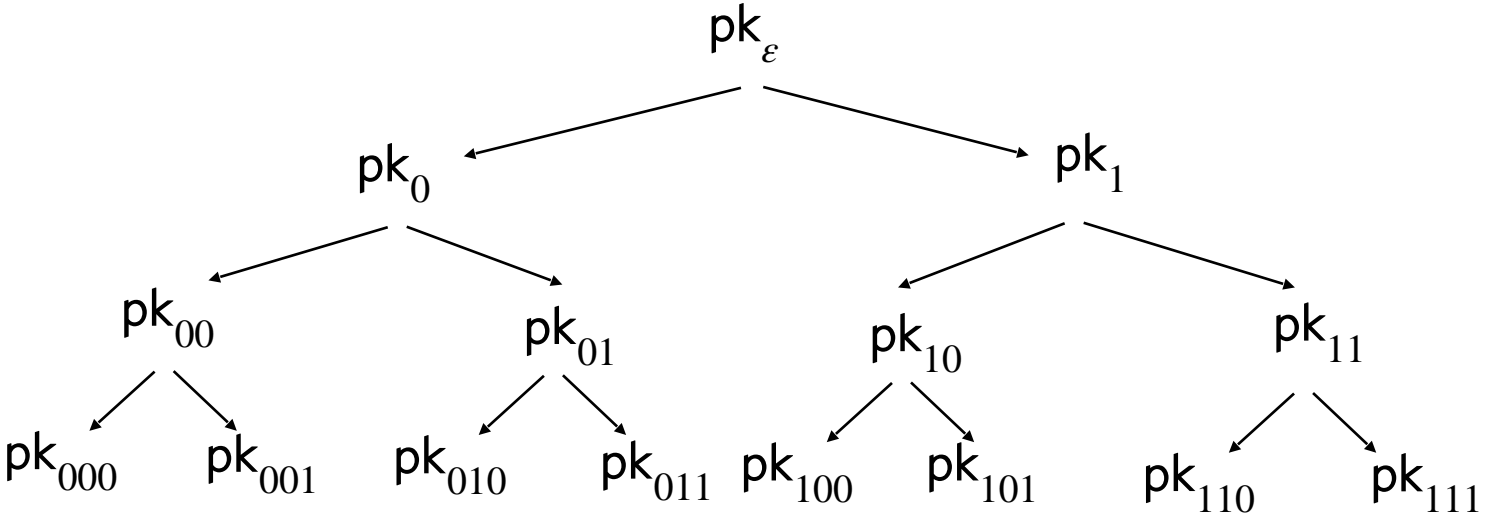
Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

Step 2. How to Shrink the signatures.

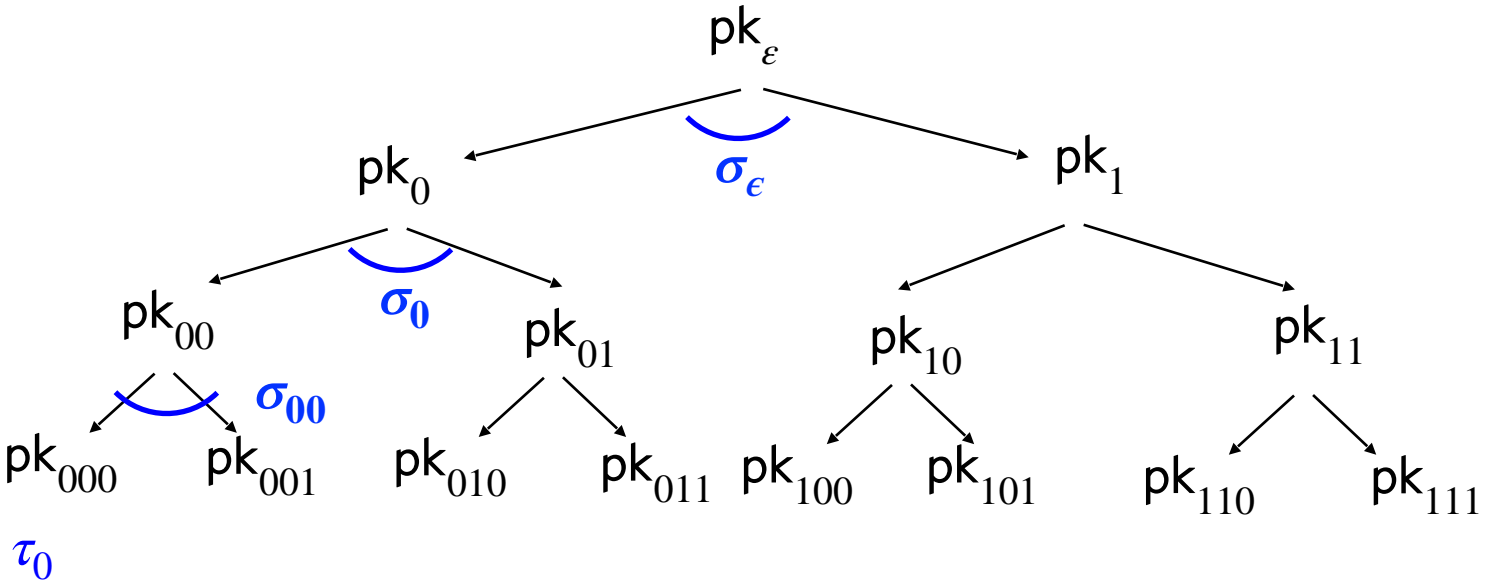
$\mathsf{pk}_{\varepsilon}$

Step 2. How to Shrink the signatures.



Alice (the *stateful* signer) computes many (pk, sk) pairs and arranges them in a tree of depth = sec. param. λ

Step 2. How to Shrink the signatures.

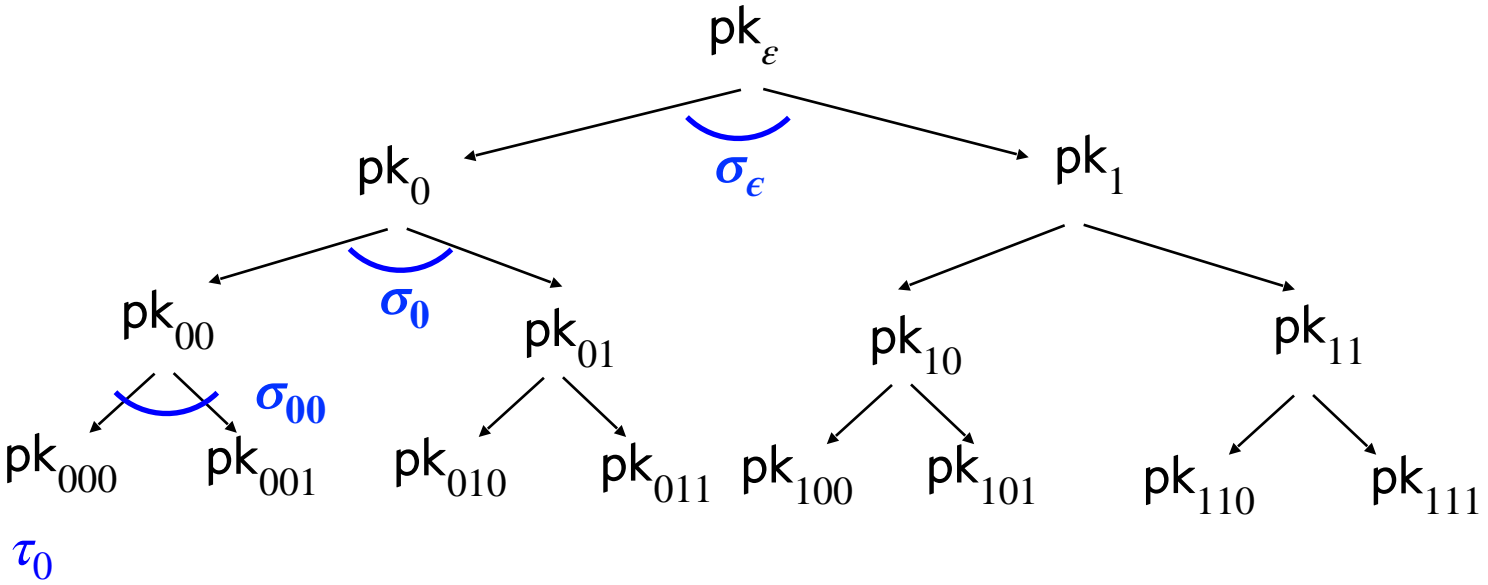


Signature of the zeroth message m_0:

Use sk_{000} to sign m_0.

"Authenticate" pk_{000} using the "signature path".

Step 2. How to Shrink the signatures.

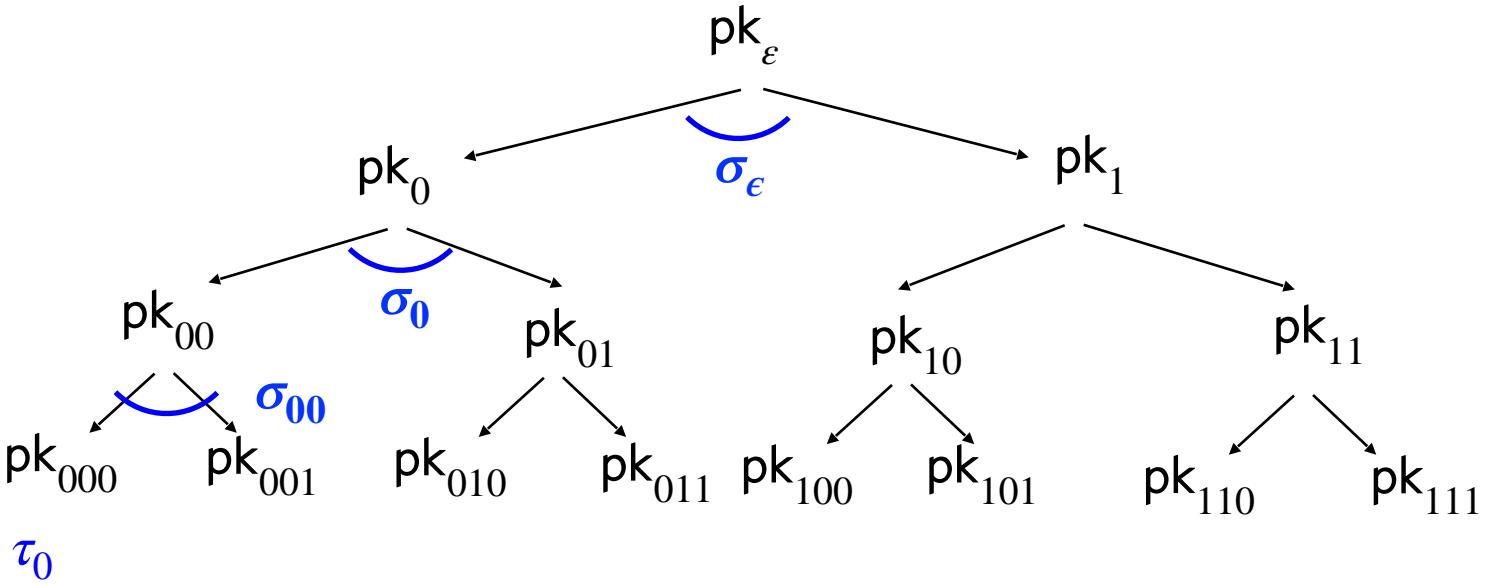


Signature of the zeroth message m_0:

$(\sigma_\varepsilon \leftarrow \mathsf{Sign}(\mathsf{sk}_\epsilon, \mathsf{pk}_0 \| \mathsf{pk}_1), \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{sk}_0, \mathsf{pk}_{00} \| \mathsf{pk}_{01}),$

$\sigma_{00} \leftarrow \mathsf{Sign}(\mathsf{sk}_{00}, \mathsf{pk}_{000} \| \mathsf{pk}_{001}), \tau_0 \leftarrow \mathsf{Sign}(\mathsf{sk}_{000}, m_0))$

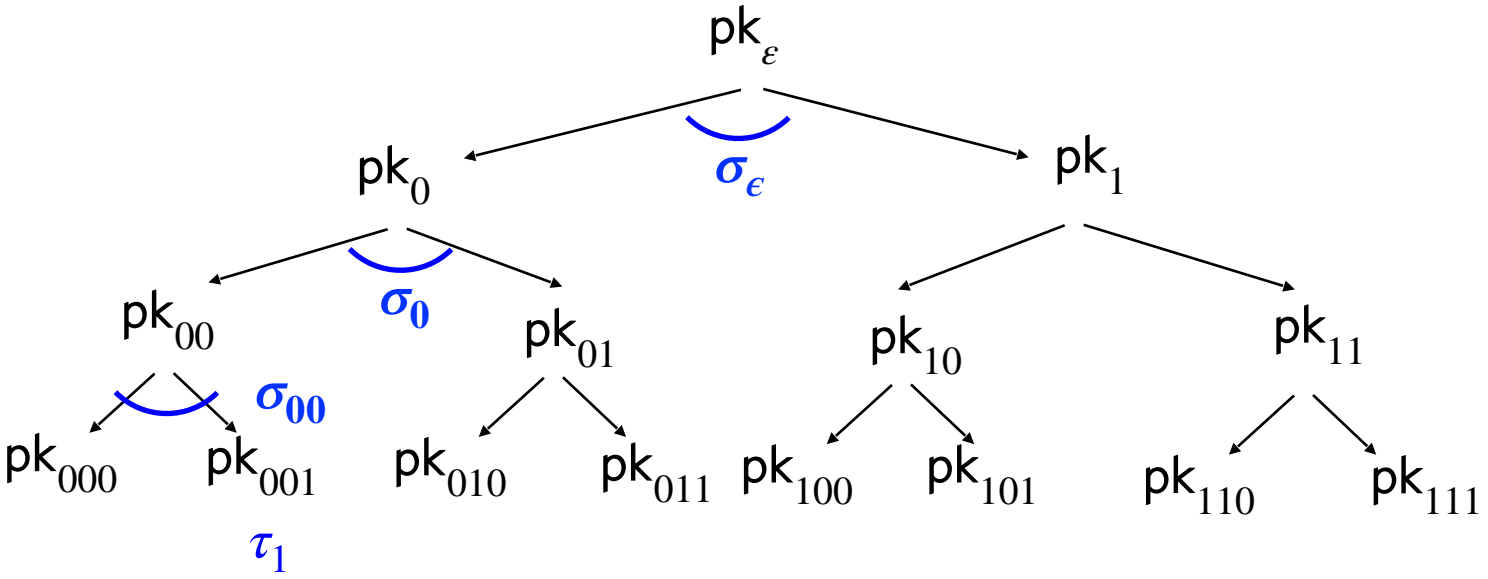
Step 2. How to Shrink the signatures.



Signature of the zeroth message m_0:

(Authentication path for pk_{000}, $\tau_0 \leftarrow \mathrm{Sign}(sk_{000}, m_0)$)

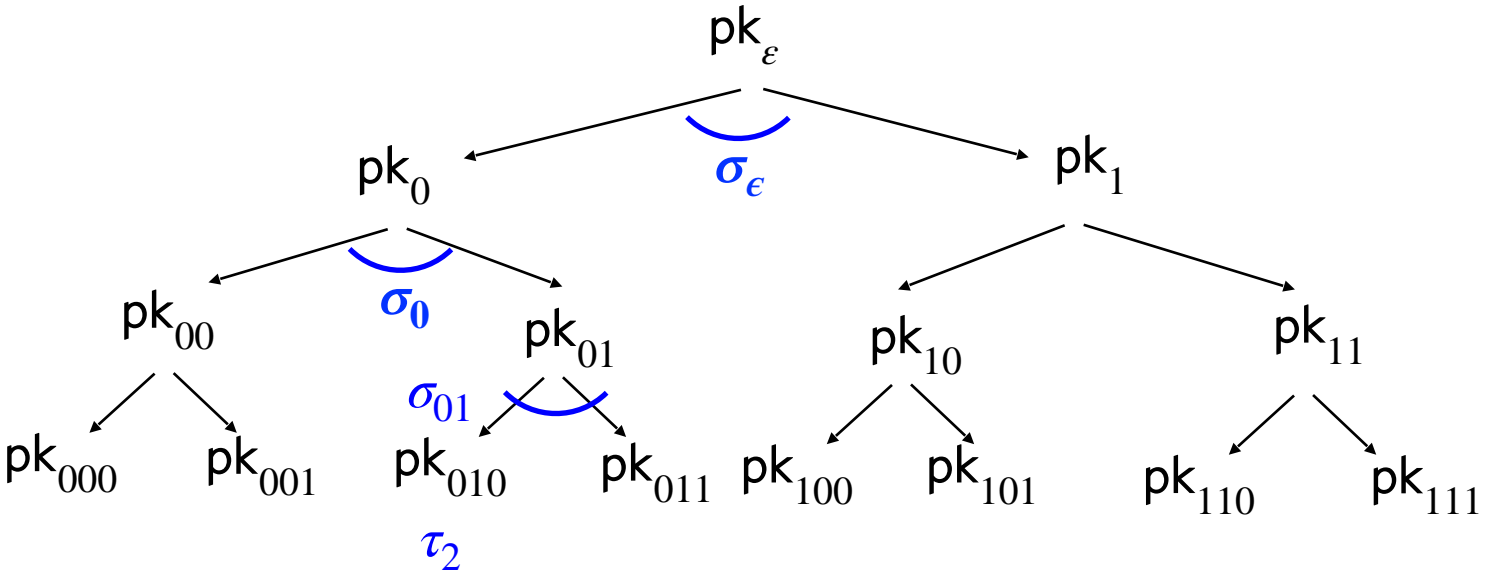
Step 2. How to Shrink the signatures.



Signature of message m_1

(Authentication path for pk_{001}, $\tau_1 \leftarrow \mathrm{Sign}(sk_{001}, m_1)$)

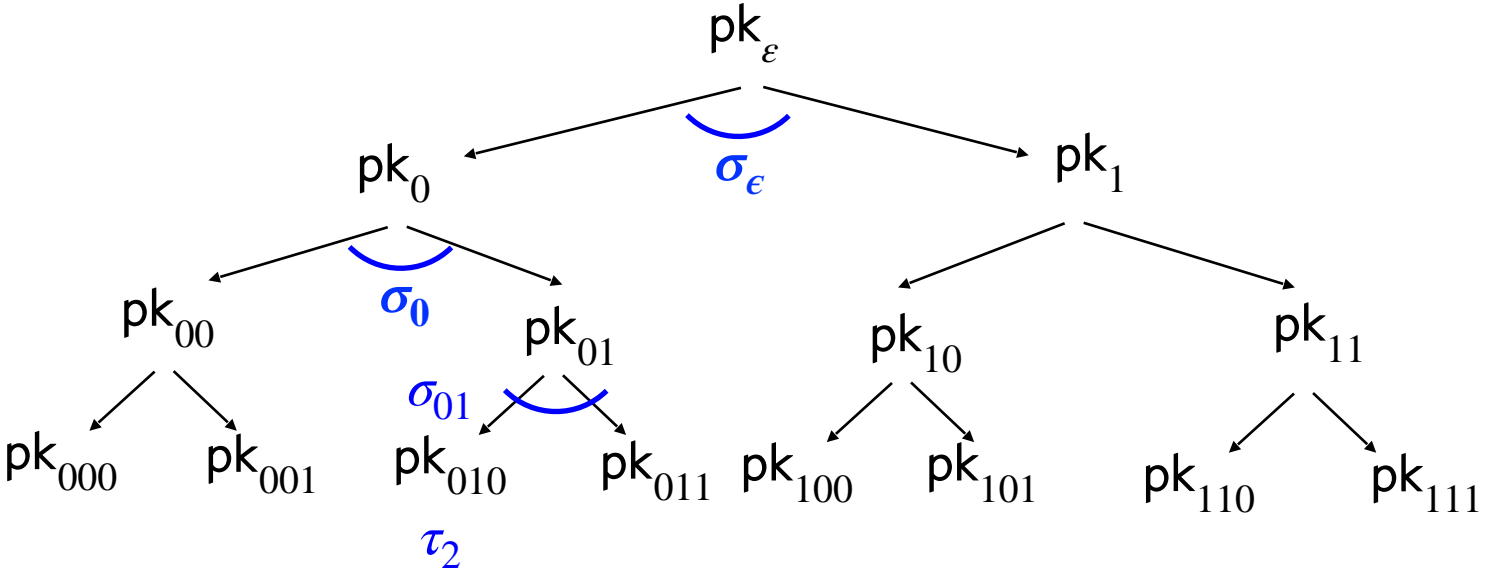
Step 2. How to Shrink the signatures.



Signature of message m_2

(Authentication path for pk_{010}, $\tau_2 \leftarrow \mathrm{Sign}(sk_{010}, m_2)$)

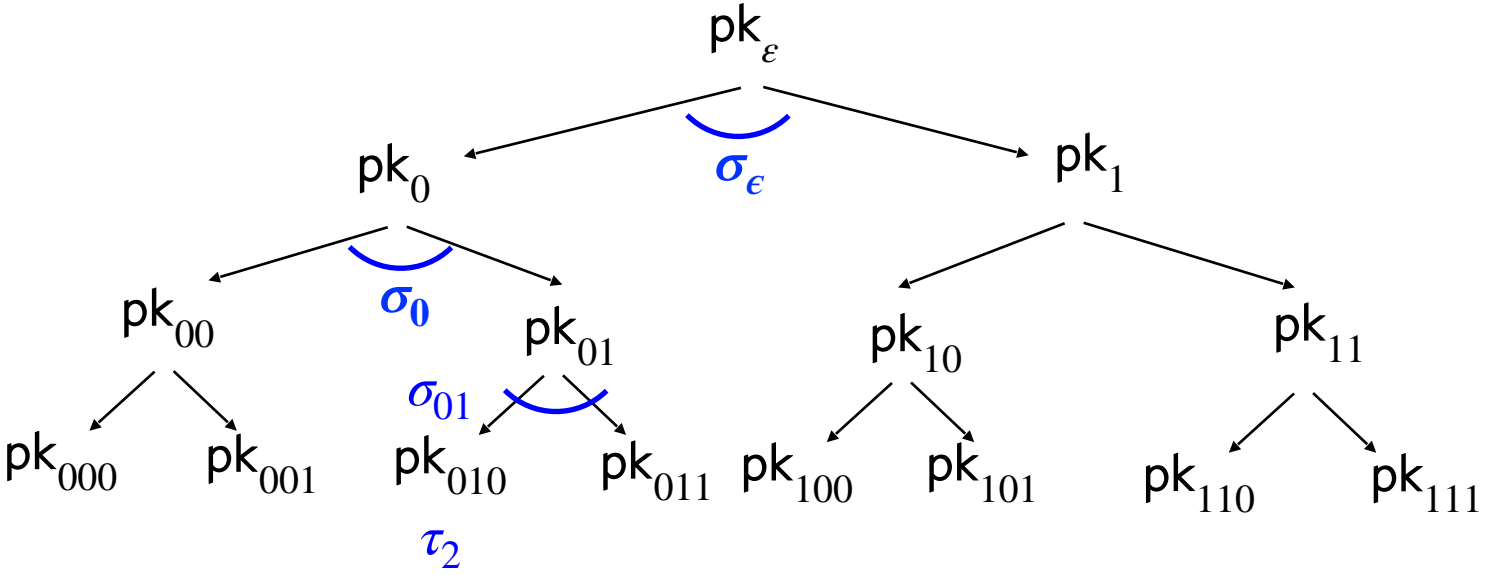
Step 2. How to Shrink the signatures.



GOOD NEWS:

Each verification key (incl. at the leaves) is used only once, so one-time security suffices!

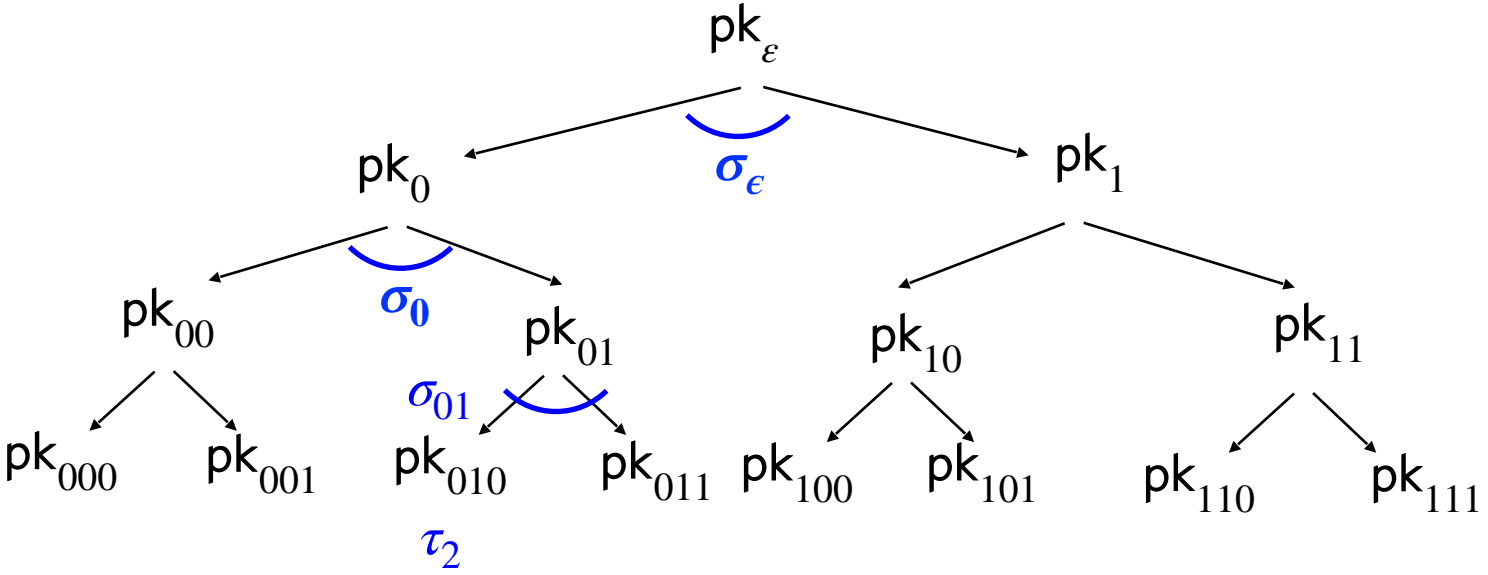
Step 2. How to Shrink the signatures.



GOOD NEWS:

Signatures consist of λ one-time signatures and do now grow with time!

Step 2. How to Shrink the signatures.



BAD NEWS:

Signer generates and keeps the entire ($\approx 2^{\lambda}$-size) signature tree in memory!

(Many-time) Signature Scheme

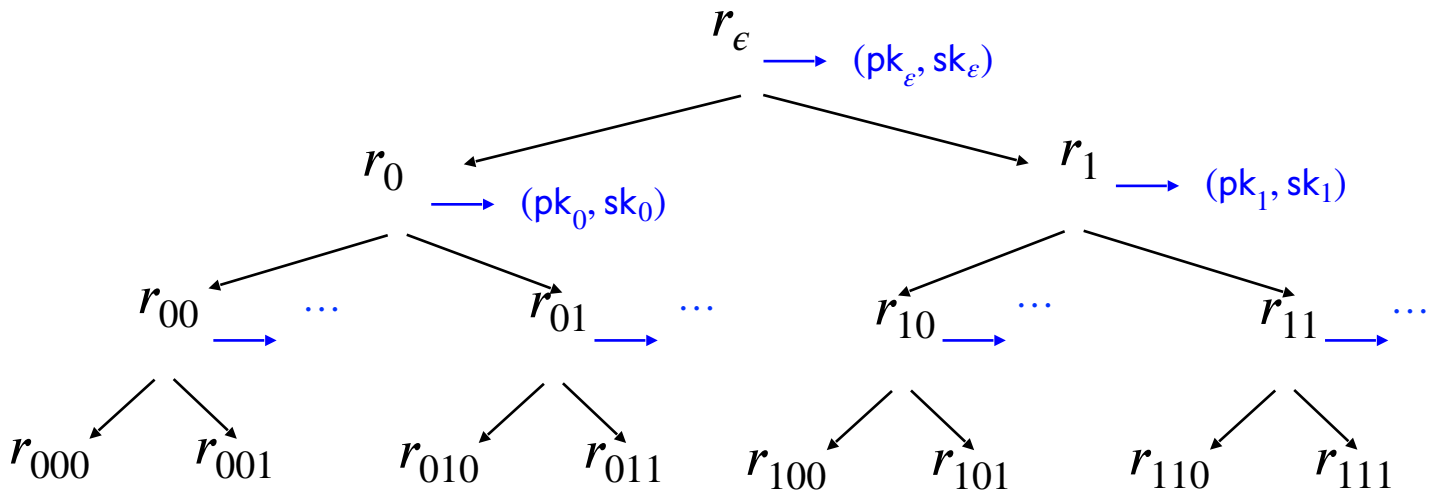
In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

Step 3. How to Shrink Alice's storage.
Idea: ***Pseudorandom Trees***

Step 3. Pseudorandom Signature Trees.



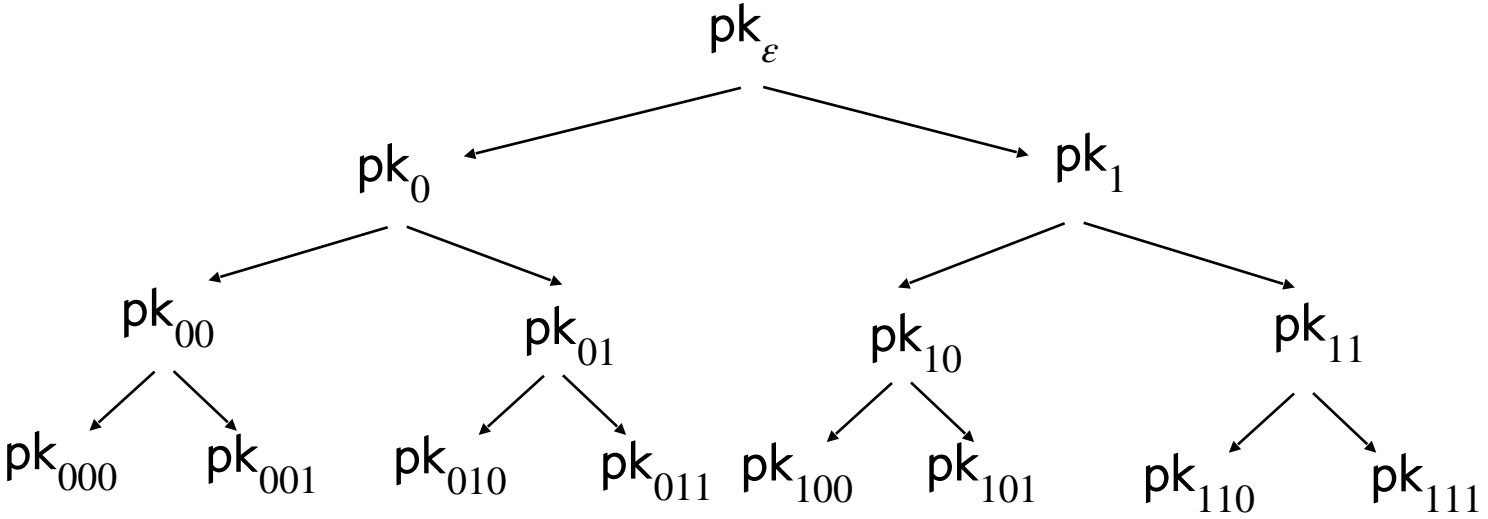
Tree of pseudorandom values:

The signing key is a PRF key k.

"Lazily" populate the nodes with $r_x := \mathsf{PRF}(k, x)$.

Use r_x to derive the keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{Gen}(1^\lambda; r_x)$.

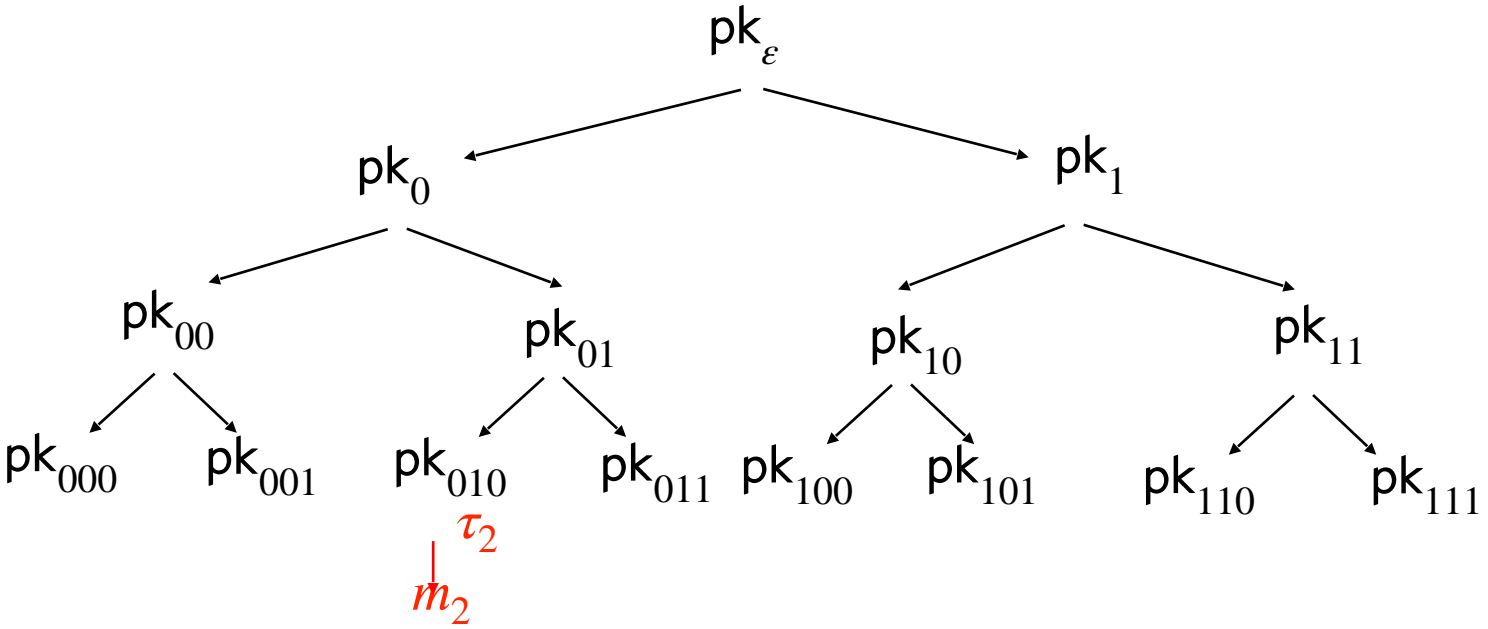
Step 3. Pseudorandom Signature Trees.



GOOD NEWS:

Short signatures and small storage for the signer

Step 3. Pseudorandom Signature Trees.



BAD NEWS:

Signer needs to keep a counter indicating which ***leaf*** (which tells her which secret key) to use next.

(Many-time) Signature Scheme

In four+ steps

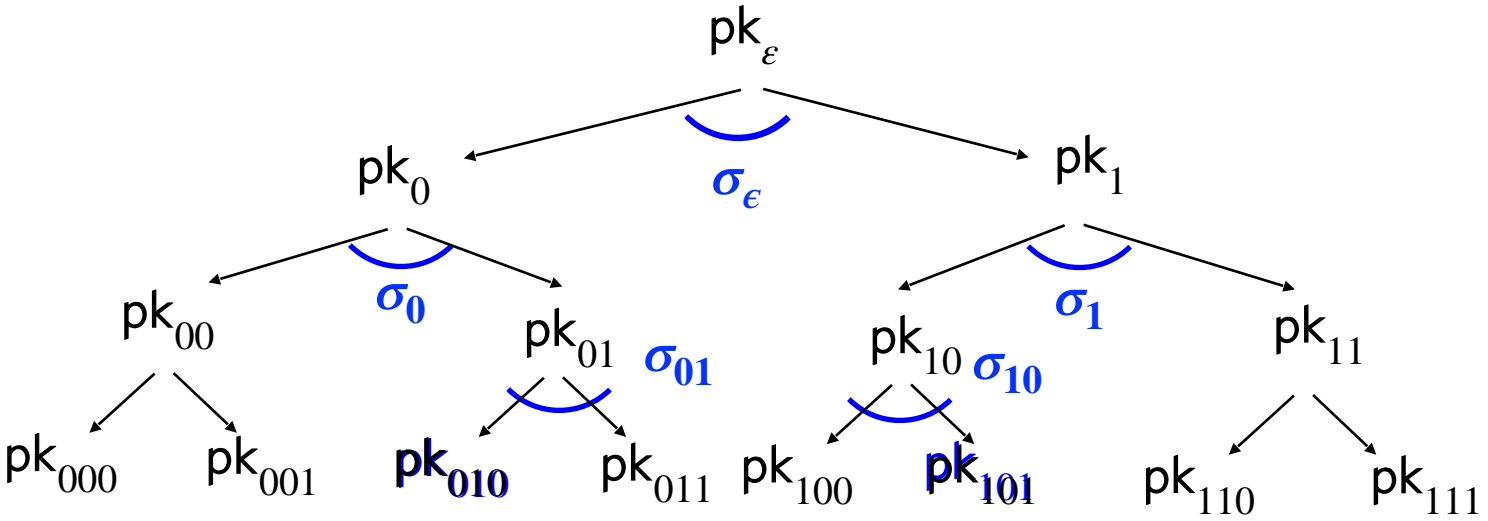
Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

Step 3. How to Shrink Alice's storage.
Idea: ***Pseudorandom Trees***

Step 4. How to make Alice stateless.
Idea: ***Randomization***

Step 4. Statelessness via Randomization



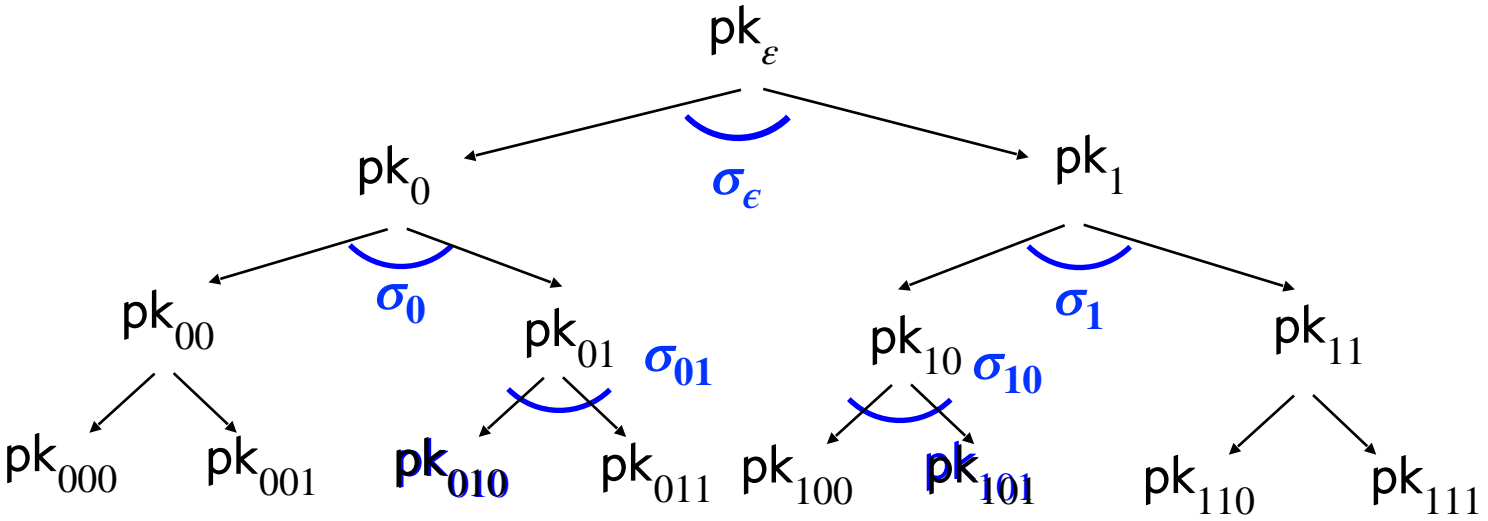
Signature of a message m:

Pick a **random** leaf r. Use pk_r to sign m.

$\sigma_r \leftarrow \mathsf{Sign}(\mathsf{sk}_r, m)$

Output $(r, \sigma_r,$ authentication path for $\mathsf{pk}_r)$

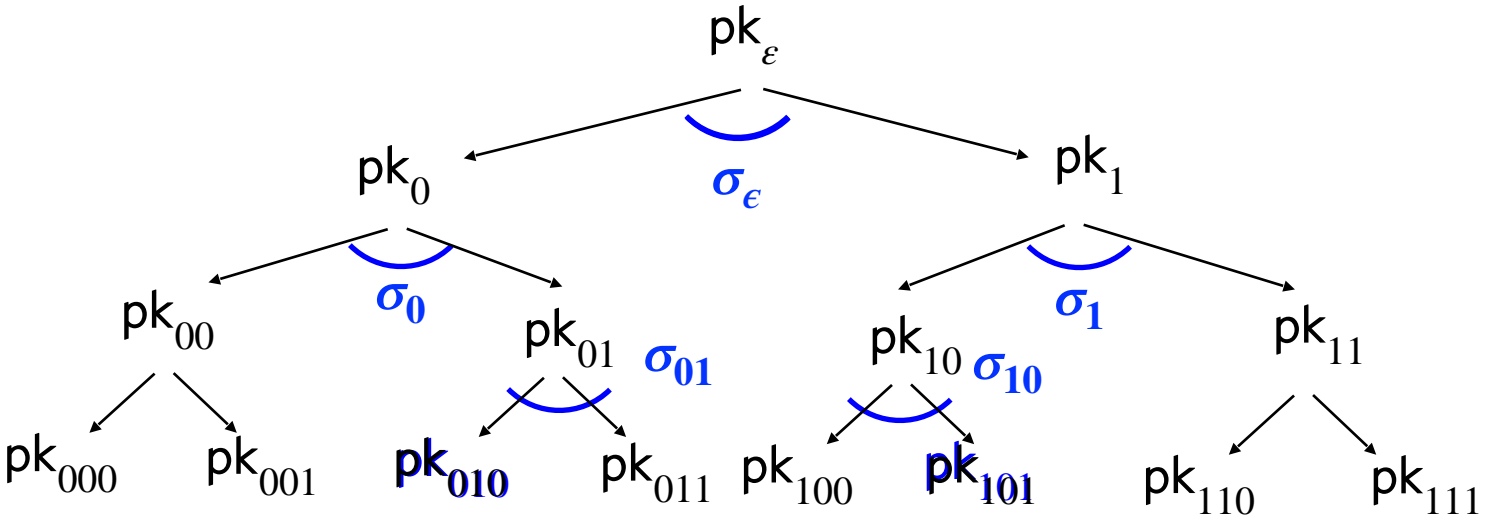
Step 4. Statelessness via Randomization



GOOD NEWS:

No need to keep state.

Step 4. Statelessness via Randomization



Key Idea:

If the signer produces q signatures, the probability she picks the same leaf twice is $\leq q^2/2^\lambda$.

(Many-time) Signature Scheme

In four+ steps

Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

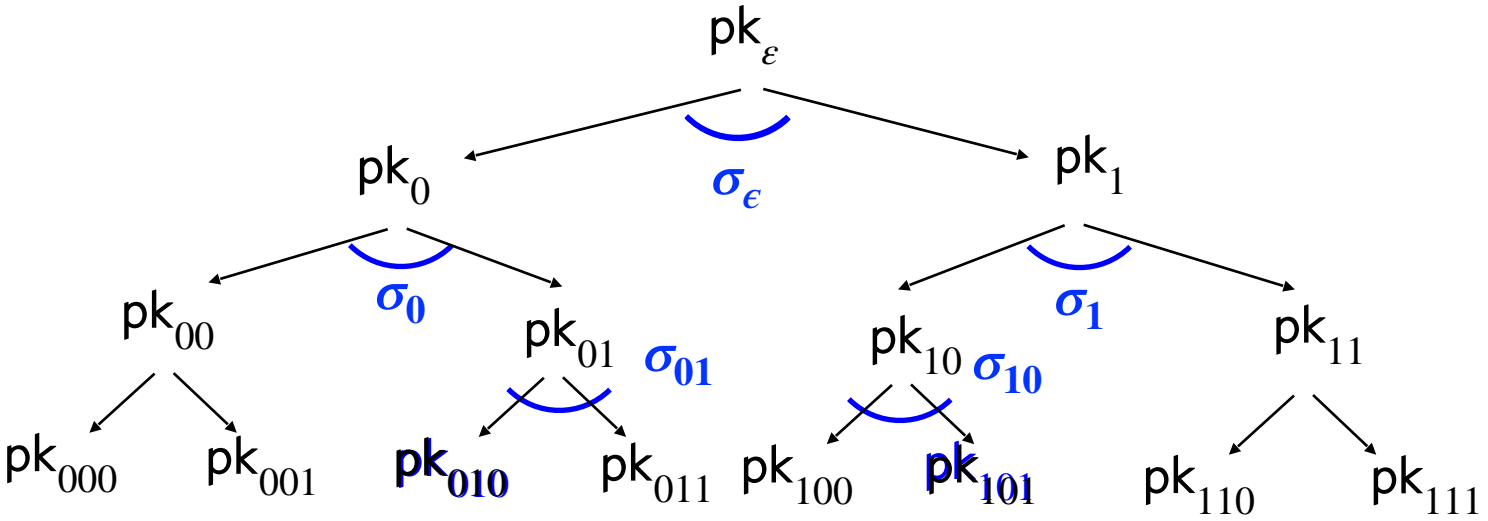
Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

Step 3. How to Shrink Alice's storage.
Idea: ***Pseudorandom Trees***

Step 4. How to make Alice stateless.
Idea: ***Randomization***

Step 5 (*optional*). How to make Alice stateless and deterministic. Idea: ***PRFs.***

Step 5. Making the Signer Deterministic.



Key Idea:

Generate r pseudo-randomly.

Have another PRF key k' and let $r = \text{PRF}(k', m)$

That’s it for the construction.

Digital Signature Construction

- Historically regarded as inefficient; therefore, never used in practice.

- However, this signature scheme (or variants thereof) are now called “hash-based signatures” and seeing a re-emergence as a candidate post-quantum secure signature scheme. E.g. https://sphincs.org/

“Vanilla” RSA Signatures

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e)$$

Sign(sk, m): Output signature $\sigma = m^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = m \pmod{N}$.

Problem: Existentially forgeable!

“Vanilla” RSA Signatures

Sign(sk, m): Output signature $\sigma = m^d \pmod N$.

Verify(vk, m, σ): Check if $\sigma^e = m \pmod N$.

Problem: Existentially forgeable!

Attack: Pick a random σ and output ($m = \sigma^e$, σ) as the forgery.

Problem: Malleable!

Attack: Given a signature of m, you can produce a signature of $2^e * m, 3^e * m, \ldots, m^2, m^3, \ldots$

“Vanilla” RSA Signatures

Sign(sk, m): Output signature $\sigma = m^d \pmod N$.

Verify(vk, m, σ): Check if $\sigma^e = m \pmod N$.

Fundamental Issues:

1. Can ”reverse-engineer” the message starting from the signature (Attack 1)

2. Algebraic structure allows malleability (Attack 2)

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e, \boldsymbol{H})$$

Sign(sk, m): Output signature $\sigma = \boldsymbol{H}(\boldsymbol{m})^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = \boldsymbol{H}(\boldsymbol{m}) \pmod{N}$.

So, what is H? Some very complicated "hash" function.

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e, \boldsymbol{H})$$

Sign(sk, m): Output signature $\sigma = \boldsymbol{H}(\boldsymbol{m})^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = \boldsymbol{H}(\boldsymbol{m}) \pmod{N}$.

H should be at least one-way to prevent Attack #1.

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e, \boldsymbol{H})$$

Sign(sk, m): Output signature $\sigma = \boldsymbol{H(m)}^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = \boldsymbol{H(m)} \pmod{N}$.

Hard to "algebraically manipulate" H(m) into H(related m').
(to prevent Attack #2.)

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e, \boldsymbol{H})$$

Sign(sk, m): Output signature $\sigma = \boldsymbol{H}(\boldsymbol{m})^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = \boldsymbol{H}(\boldsymbol{m}) \pmod{N}$.

Collision-resistance does not seem to be enough. (Given a CRHF h(m), you may be able to produce h(m') for related m'.)

How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\varphi(N)$ and let $d = e^{-1} \pmod{\varphi(N)}$.

$$\text{sk} = (N, d) \quad \text{and} \quad \text{pk} = (N, e, \boldsymbol{H})$$

Sign(sk, m): Output signature $\sigma = \boldsymbol{H}(\boldsymbol{m})^d \pmod{N}$.

Verify(vk, m, σ): Check if $\sigma^e = \boldsymbol{H}(\boldsymbol{m}) \pmod{N}$.

Collision-resistance does not seem to be enough. (Given a CRHF h(m), you may be able to produce h(m') for related m'.)

The Random Oracle Heuristic

Want: A public H that is "non-malleable".

Given H(m), it is hard to produce H(m') for any *non-trivially related* m'.

For every PPT adv A and "every non-trivial relation" R,

$$\Pr\left[A(H(m)) = H(m') : R(m, m') = 1\right] = \mathrm{negl}(\lambda)$$

How about the relation R where

$R(x, y) = 1$ if and only if

$y = \mathrm{H}(\mathrm{x})$?

The Random Oracle Heuristic

Proxy: A public H that "behaves like a random function"

(A PRF also behaves like a random function, but PRF_K is ***not*** publicly computable.)

Reality:

$\mathcal{A}^{H}$

Random Oracle Heuristic:

$\mathcal{A}^{H}(1^{\lambda})$

H is virtually a black box.
The only way to compute H is by calling the oracle.