MIT 6.875

Foundations of Cryptography
Lecture 12

RECAP from L11

Digital Signatures: Definition

A triple of PPT algorithms $(Gen, Sign, Verify)$ s.t.

- $(vk, sk) \leftarrow Gen(1^n)$.
- $\sigma \leftarrow Sign(sk, m)$.
- $\mathrm{A}cc(1)/Rej(0) \leftarrow Verify(vk, m, \sigma)$.

Correctness: For all vk, sk, m:

$$Verify\big(vk, m, Sign(sk, m)\big) = \text{accept}.$$

EUF-CMA Security

(Existentially Unforgeable against a Chosen Message Attack)

Challenger

Eve

$(vk, sk) \leftarrow Gen(1^n)$ $\xrightarrow{\quad vk \quad}$

$\xleftarrow{\quad m_i \quad}$ poly many times

$\sigma_i \leftarrow Sign(sk, m_i)$ $\xrightarrow{\quad \sigma_i \quad}$

$\xleftarrow{\quad m^*, \sigma^* \quad}$

Eve wins if Verify(vk, m^*, σ^*)= 1 and $m^* \notin \{m_1, m_2, \ldots\}$.
The signature scheme is EUF-CMA-secure if no PPT Eve can win with probability better than negl(n).

Lamport (One-time) Signatures

How to sign n bits

Verification Key VK: $\begin{bmatrix} y_{1,0} & y_{2,0} & & y_{n,0} \\ y_{1,1} & y_{2,1} & & y_{n,1} \end{bmatrix}$

where $y_{i,c} = f(x_{i,c})$.

Signing an n-bit message $(m_1, \ldots, m_n)$:

The signature is $\left(x_{1,m_1}, \ldots, x_{n,m_n}\right)$.

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.

TODAY: Digital Signatures, Continued

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.

Step 0: How to Sign Polynomially Many Bits

(with a fixed verification key)

Detour: Collision-Resistant Hash Functions

A compressing **family of functions** $\mathcal{H} = \{\mathrm{h}: \{0,1\}^m \to \{0,1\}^n\}$ (where $m > n$) for which it is computationally hard to find collisions.

Def: $\mathcal{H}$ is collision-resistant if for every PPT algorithm A, there is a negligible function μ s.t.

$$\Pr_{h \leftarrow \mathcal{H}}[A(1^n, h) = (x, y): x \neq y, h(x) = h(y)] = \mu(n)$$

Do CRHFs exist?

- **Theoretical Constructions**: assuming discrete logarithms (as well as under several other number-theoretic assumptions)

- **Practical Constructions**: SHA3.

- **Domain Extension Theorem**: If there exist hash functions compressing $n+1$ bits to n bits, then there are hash functions that compress any $\mathrm{poly}(n)$ bits into n bits.

How to Sign Polynomially Many Bits

(with a fixed verification key)

Idea: Hash the message into n bits and sign the hash.

Signing Key SK: $\begin{bmatrix} x_{1,0} & x_{2,0} & & x_{n,0} \\ x_{1,1} & x_{2,1} & & x_{n,1} \end{bmatrix}$

Verification Key VK: $\begin{bmatrix} y_{1,0} & y_{2,0} & & y_{n,0} \\ y_{1,1} & y_{2,1} & & y_{n,1} \end{bmatrix}$ **and $\boldsymbol{h} \leftarrow \boldsymbol{\mathcal{H}}$.**

Signing an n-bit message m: Compute the hash $z = h(m)$.

The signature is $\left(x_{1,z_1}, \ldots, x_{n,z_n}\right)$.

Verifying (m, σ): Recompute the hash $z = h(m)$.

Check if $\forall i$: $f(\sigma_i) = y_{i,z_i}$

How to Sign Polynomially Many Bits

(with a fixed verification key)

Claim: Assuming f is a OWF and $\mathcal{H}$ is a collision-resistant family, no PPT adv can produce a signature of m given a signature of a single $m' \neq m$.

Proof Idea:

Either the adversary picked m' s.t. $h(m') = h(m)$, in which case she violated collision-resistance of $\mathcal{H}$.

(or)

She produced a Lamport signature on a “message” $z' \neq z$, in which case she violated one-time security of Lamport, and therefore the one-wayness of f.

Let's go back to CRHFs…

$p = 2q + 1$ is a "safe" prime.

$$\mathcal{H} = \{\mathrm{h}\colon (\mathbb{Z}_q)^2 \rightarrow QR_p\}$$

Each function $h_{g_1, g_2} \in \mathcal{H}$ is parameterized by two generators g_1 and g_2 of QR_p (a group of order q).

$h_{g_1, g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

This compresses 2 log q bits into log p ≈ log q + 1 bits.

Let's go back to CRHFs…

$p = 2q + 1$ is a "safe" prime.

$$\mathcal{H} = \{\mathrm{h}: (\mathbb{Z}_q)^2 \rightarrow QR_p\}$$

Each function $h_{g_1, g_2} \in \mathcal{H}$ is parameterized by two generators g_1 and g_2 of QR_p (a group of order q).

$h_{g_1, g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

Why is this collision-resistant? Suppose there is an adversary that finds a collision (x_1, x_2) and (y_1, y_2)…

Let's go back to CRHFs…

$h_{g_1,g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2}$ mod p.

Why is this collision-resistant? Suppose there is an adversary that finds a collision (x_1, x_2) and (y_1, y_2)…

$g_1^{x_1} g_2^{x_2}$ = $g_1^{y_1} g_2^{y_2}$ mod p.

$g_1^{x_1-y_1}$ = $g_2^{y_2-x_2}$ mod p.

(assume wlog $x_1 - y_1 \neq 0 \; mod \; q$)

g_1 = $g_2^{(y_2-x_2)(x_1-y_1)^{-1}}$ mod p. $\Rightarrow$ $DLOG_{g_2}(g_1)$!

Other Constructions of CRHFs

From the hardness of factoring, lattice problems etc.

Not known to follow from the existence of one-way functions or even one-way permutations…

"Black-box separations": Certain ways of constructing CRHF from OWF/OWP cannot work.

"Finding collisions on a one-way street", Daniel Simon, Eurocrypt 1998.

Nevertheless, big open problem: OWF/OWP $\Longrightarrow^{?}$ CRHF?

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	VK_0

Alice starts with a secret signing Key SK_0.

When signing a message m_1:

Generate a new pair (VK_1, SK_1).

Produce signature $\sigma_1 \leftarrow \text{Sign}(SK_0, m_1||VK_1)$

Output $VK_1||\sigma_1$.

Remember $VK_1||m_1||\sigma_1$ as well as SK_1.

To verify a signature $VK_1||\sigma_1$ for message m_1:

Run Verify($VK_0, m_1||VK_1, \sigma_1$)

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	VK_0

Alice starts with a secret signing Key SK_0.

When signing a message m_1:

Generate a new pair (VK_1, SK_1).

Produce signature $\sigma_1 \leftarrow \text{Sign}(SK_0, m_1||VK_1)$

Output $VK_1||\sigma_1$.

Remember $VK_1||m_1||\sigma_1$ as well as SK_1.

$$VK_0 \xrightarrow{\sigma_1} \overset{m_1}{VK_1}$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	VK_0

Alice starts with a secret signing Key SK_0.

When signing the next message m_2:

Generate a new pair (VK_2, SK_2).

Produce signature $\sigma_2 \leftarrow \mathrm{Sign}(SK_1, m_2||VK_2)$

Output $VK_2||\sigma_2$??

$$VK_0 \xrightarrow{\sigma_1} \overset{m_1}{VK_1}$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	VK_0

Alice starts with a secret signing Key SK_0.

When signing the next message m_2:

Generate a new pair (VK_2, SK_2).

Produce signature $\sigma_2 \leftarrow \text{Sign}(SK_1, m_2 || VK_2)$

Output $VK_1 || VK_2 || \sigma_2 ??$

$$VK_0 \xrightarrow{\sigma_1} \begin{matrix} m_1 \\ VK_1 \end{matrix}$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	VK_0

Alice starts with a secret signing Key SK_0.

When signing the next message m_2:

Generate a new pair (VK_2, SK_2).

Produce signature $\sigma_2 \leftarrow \mathrm{Sign}(SK_1, m_2 || VK_2)$

Output $VK_1 || m_1 || \sigma_1 || VK_2 || \sigma_2$.

$$VK_0 \xrightarrow{\sigma_1} \overset{m_1}{VK_1} \xrightarrow{\sigma_2} \overset{m_2}{VK_2} \xrightarrow{\sigma_3} \overset{m_3}{VK_3} \xrightarrow{\sigma_4} \overset{m_4}{VK_4} \quad \cdots$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	VK_0

Alice starts with a secret signing Key SK_0.

When signing the next message m_2:

Generate a new pair (VK_2, SK_2).

Produce signature $\sigma_2 \leftarrow \text{Sign}(SK_1, m_2 || VK_2)$

Output $VK_1 || m_1 || \sigma_1 || VK_2 || \sigma_2$.

(additionally) remember $VK_2 || m_2 || \sigma_2$ as well as SK_2

$$VK_0 \xrightarrow{\sigma_1} \overset{m_1}{VK_1} \xrightarrow{\sigma_2} \overset{m_2}{VK_2} \xrightarrow{\sigma_3} \overset{m_3}{VK_3} \xrightarrow{\sigma_4} \overset{m_4}{VK_4} \quad \cdots$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

An optimization: Need to remember only the past verification keys, not the past messages.

Use (part of) VK_i to sign m_{i+1} and the rest to sign VK_{i+1}.

$$\begin{tikzcd}
 & m_1 & m_2 & m_3 & m_4 & \iddots \\
VK_0 \arrow[ur, "\tau_1"] \arrow[r, "\sigma_1"'] & VK_1 \arrow[ur, "\tau_2"] \arrow[r, "\sigma_2"'] & VK_2 \arrow[ur, "\tau_3"] \arrow[r, "\sigma_3"'] & VK_3 \arrow[ur, "\tau_4"] \arrow[r, "\sigma_4"'] & VK_4 & \cdots
\end{tikzcd}$$

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 $O(T)$.

$$
\begin{tikzcd}
 & m_1 & m_2 & m_3 & m_4 & \\
VK_0 \arrow[r, "\sigma_1"] \arrow[ur, "\tau_1"] & VK_1 \arrow[r, "\sigma_2"] \arrow[ur, "\tau_2"] & VK_2 \arrow[r, "\sigma_3"] \arrow[ur, "\tau_3"] & VK_3 \arrow[r, "\sigma_4"] \arrow[ur, "\tau_4"] & VK_4 & \cdots
\end{tikzcd}
$$

(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***

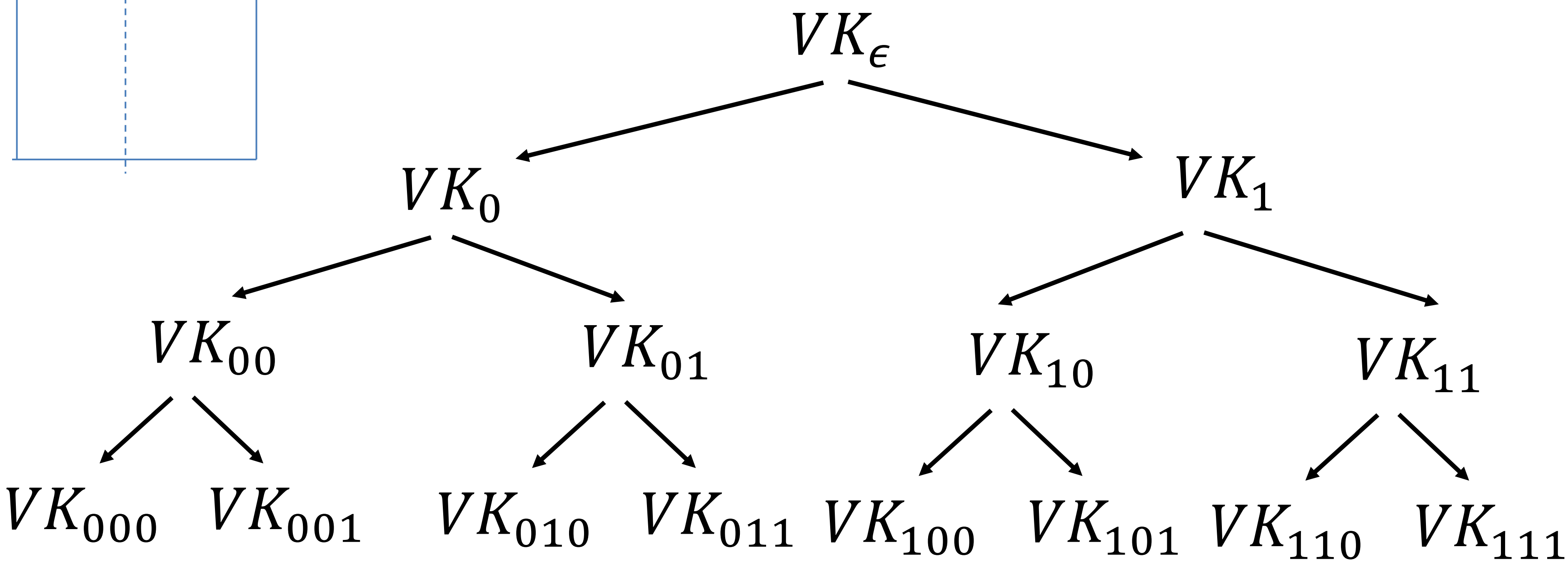
Alice	VK_ϵ

Step 2. How to Shrink the signatures.

$$VK_\epsilon$$

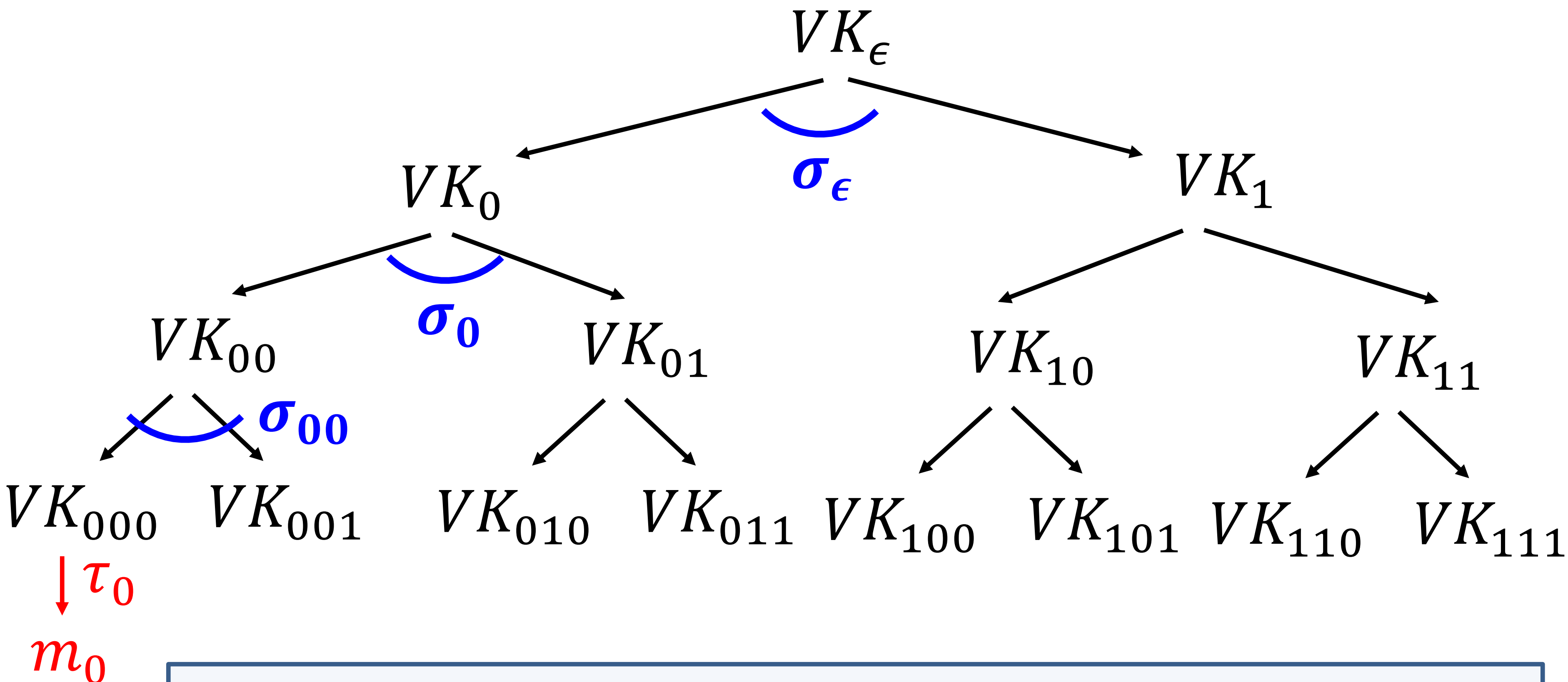
Alice	VK_ϵ

Step 2. How to Shrink the signatures.



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

Step 2. How to Shrink the signatures.

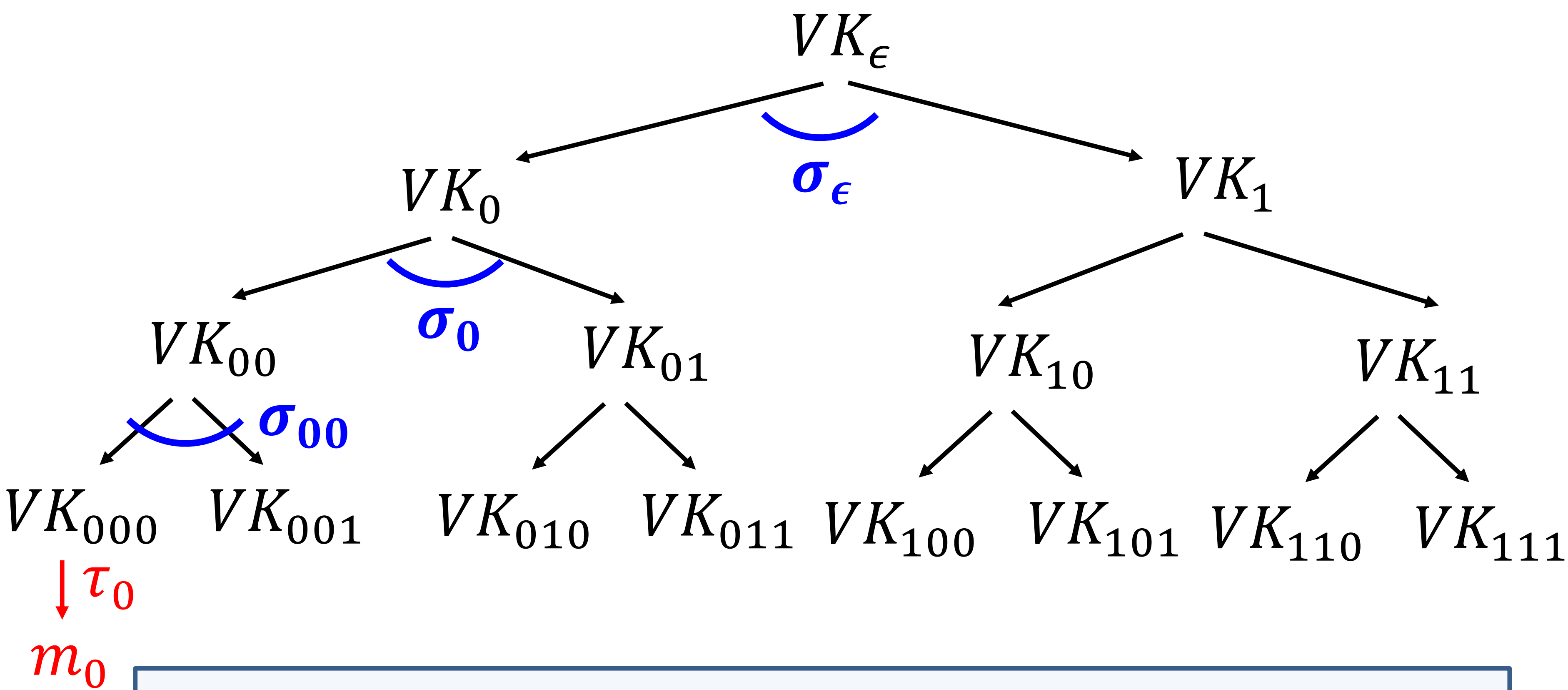


Signature of the first message $\boldsymbol{m_0}$:

Use VK_{000} to sign m_0.

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

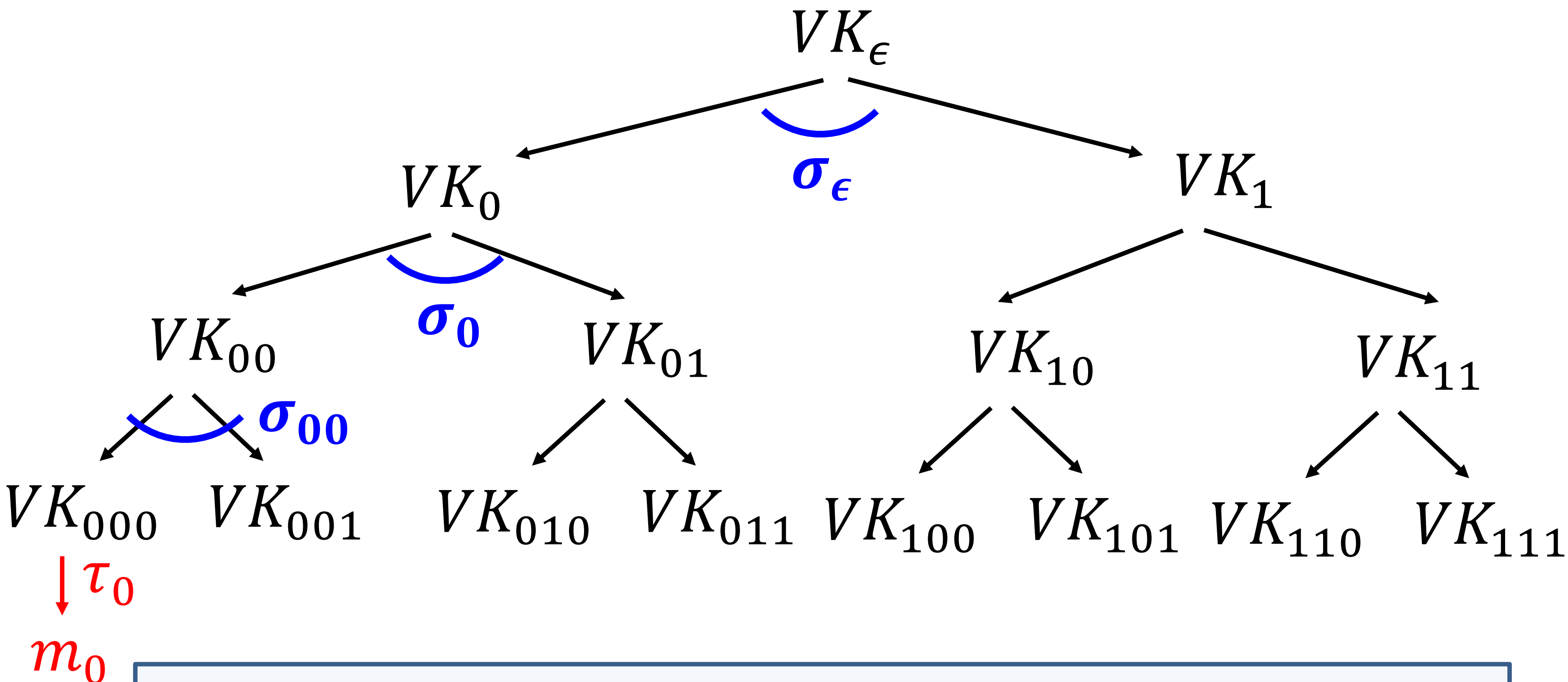
Step 2. How to Shrink the signatures.



Signature of the first message $\boldsymbol{m_0}$:

$$\Big(\boldsymbol{\sigma_\epsilon} \leftarrow \mathrm{Sign}(SK_\epsilon, VK_0||VK_1),\ \boldsymbol{\sigma_0} \leftarrow \mathrm{Sign}(SK_0, VK_{00}||VK_{01}),$$
$$\boldsymbol{\sigma_{00}} \leftarrow \mathrm{Sign}(SK_{00}, VK_{000}||VK_{001}),\ \boldsymbol{\tau_0} \leftarrow \mathrm{Sign}(SK_{000}, m_0)\Big)$$

Step 2. How to Shrink the signatures.

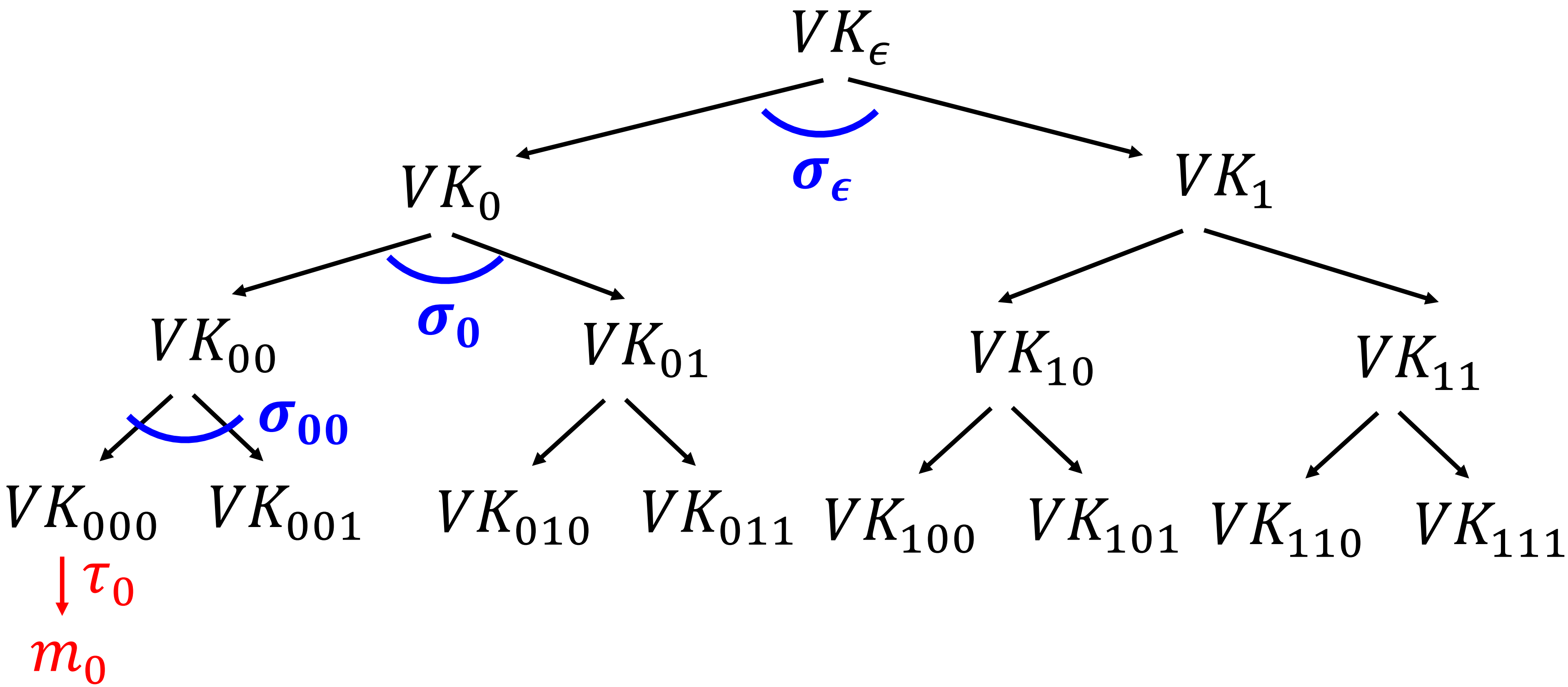


Authentication Path for VK_{000}:

$\Big(\boldsymbol{\sigma_\epsilon} \leftarrow \mathrm{Sign}(SK_\epsilon, VK_0 || VK_1), \boldsymbol{\sigma_0} \leftarrow \mathrm{Sign}(SK_0, VK_{00} || VK_{01}),$

$\boldsymbol{\sigma_{00}} \leftarrow \mathrm{Sign}(SK_{00}, VK_{000} || VK_{001})\Big)$

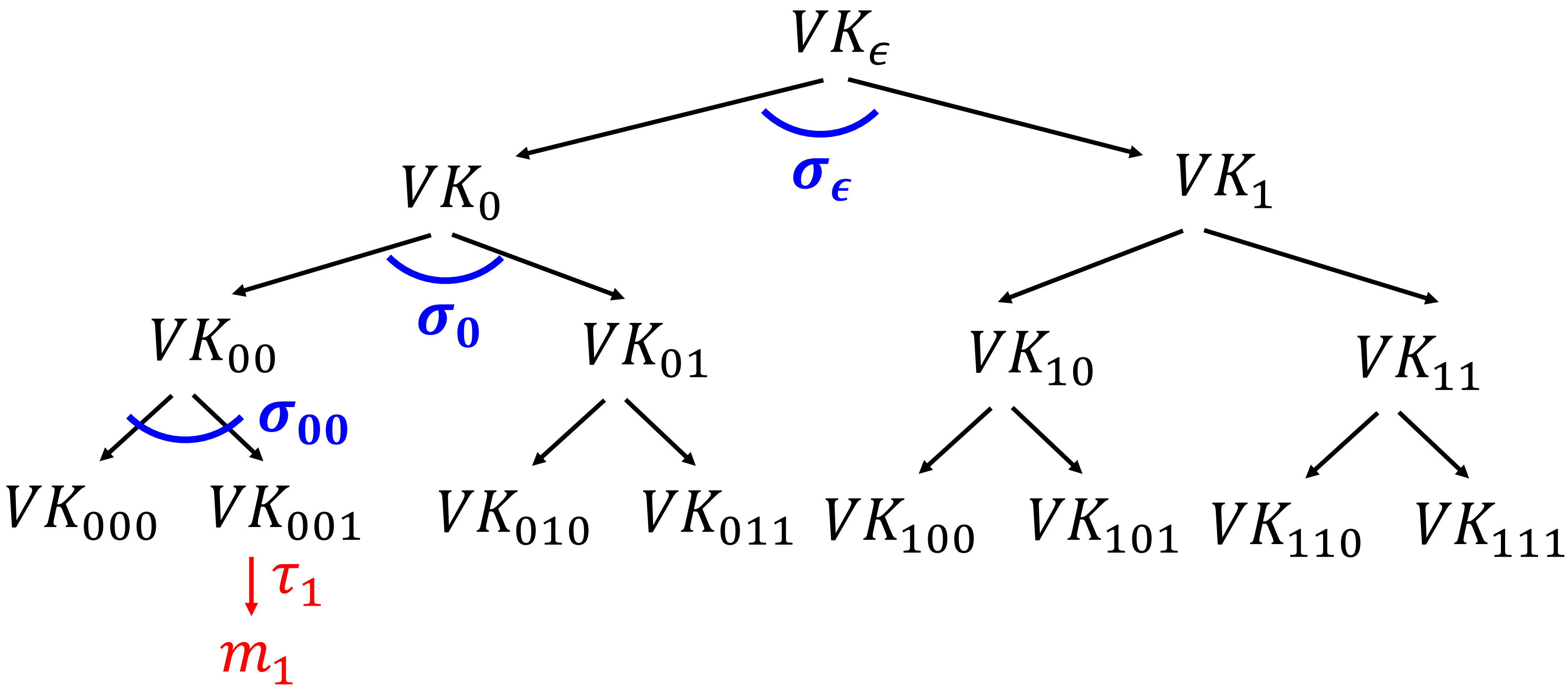
Step 2. How to Shrink the signatures.



Signature of the first message $\boldsymbol{m_0}$:

$\big($Authentication path for VK_{000}, $\boldsymbol{\tau_0} \leftarrow \mathrm{Sign}(SK_{000}, m_0)\big)$

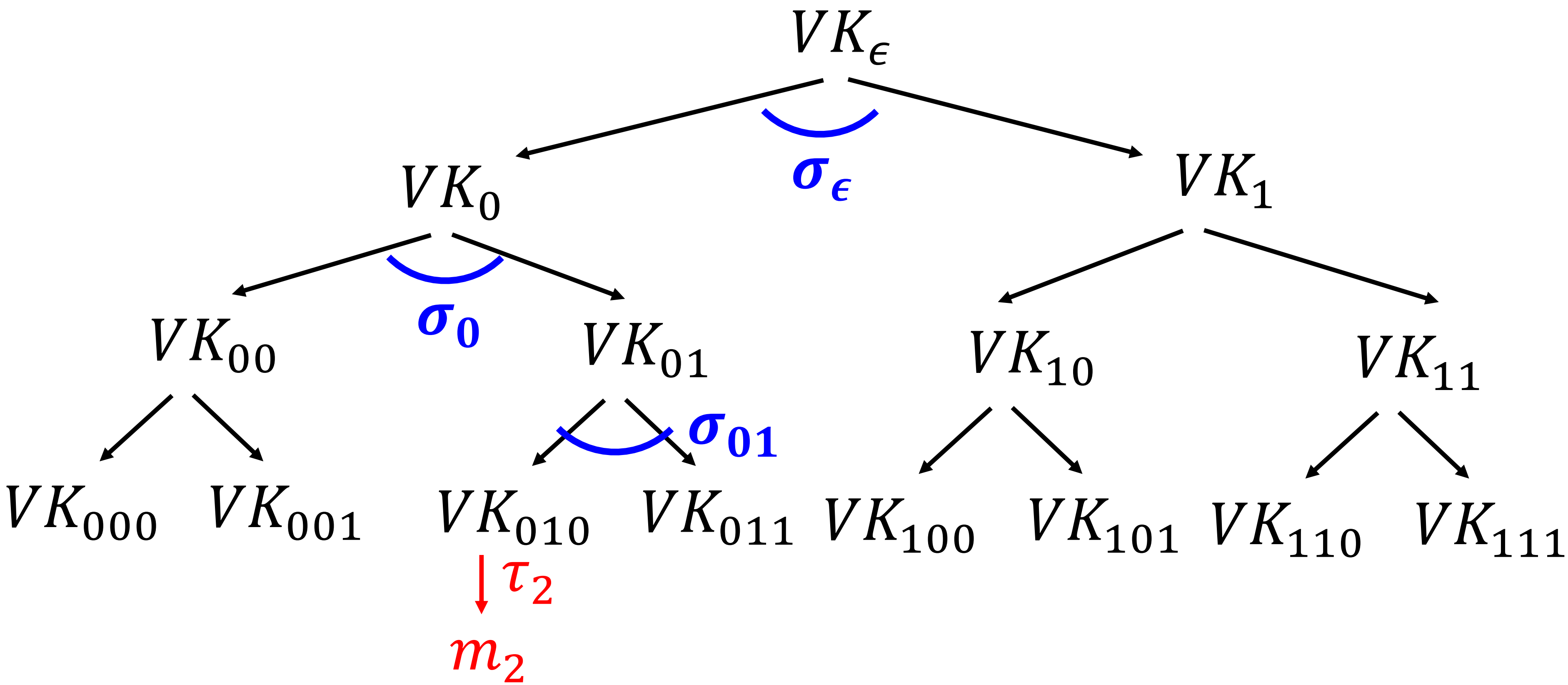
Step 2. How to Shrink the signatures.



Signature of the second message m_1:

(Authentication path for VK_{001}, $\tau_0 \leftarrow \mathrm{Sign}(SK_{001}, m_1)$)

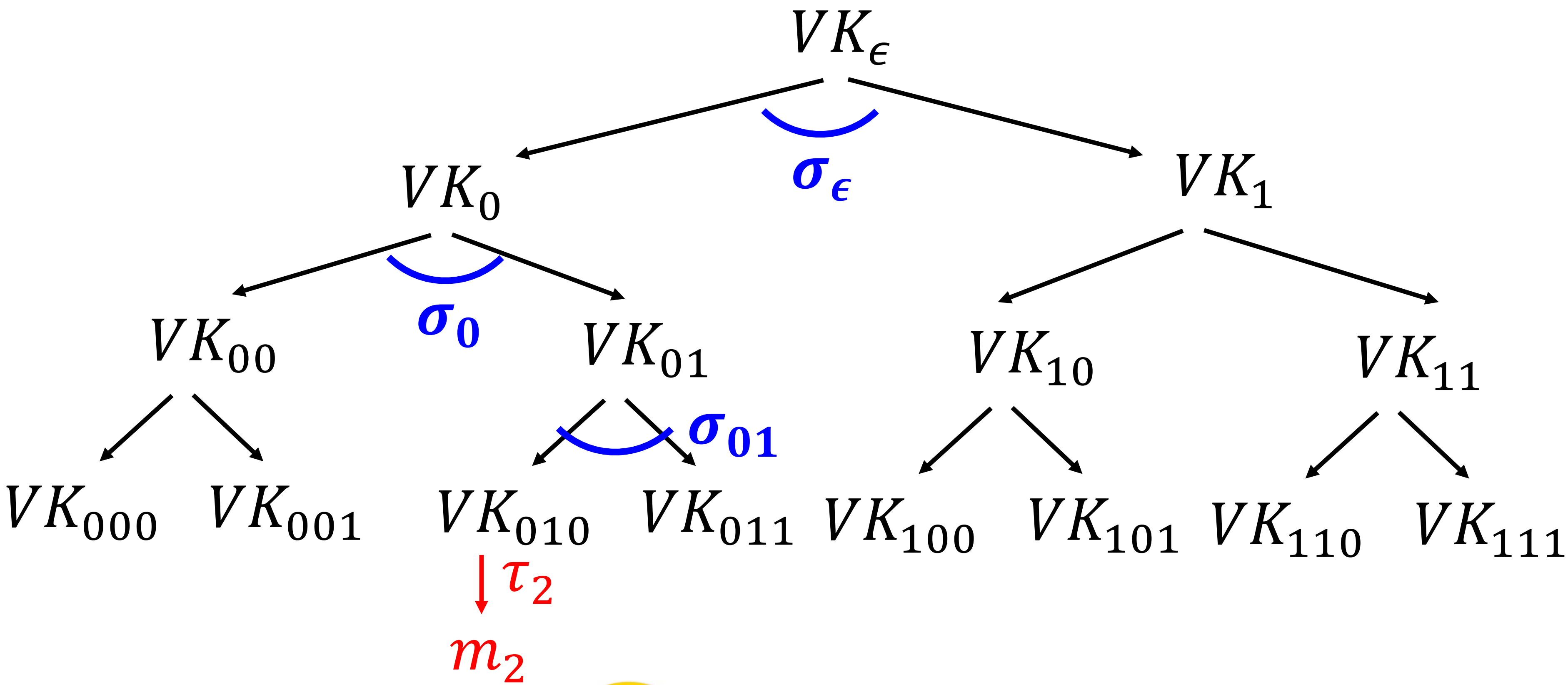
Step 2. How to Shrink the signatures.



Signature of the third message m_2:

$\big($Authentication path for VK_{010}, $\boldsymbol{\tau_2} \leftarrow \mathrm{Sign}(SK_{010}, m_2)\big)$

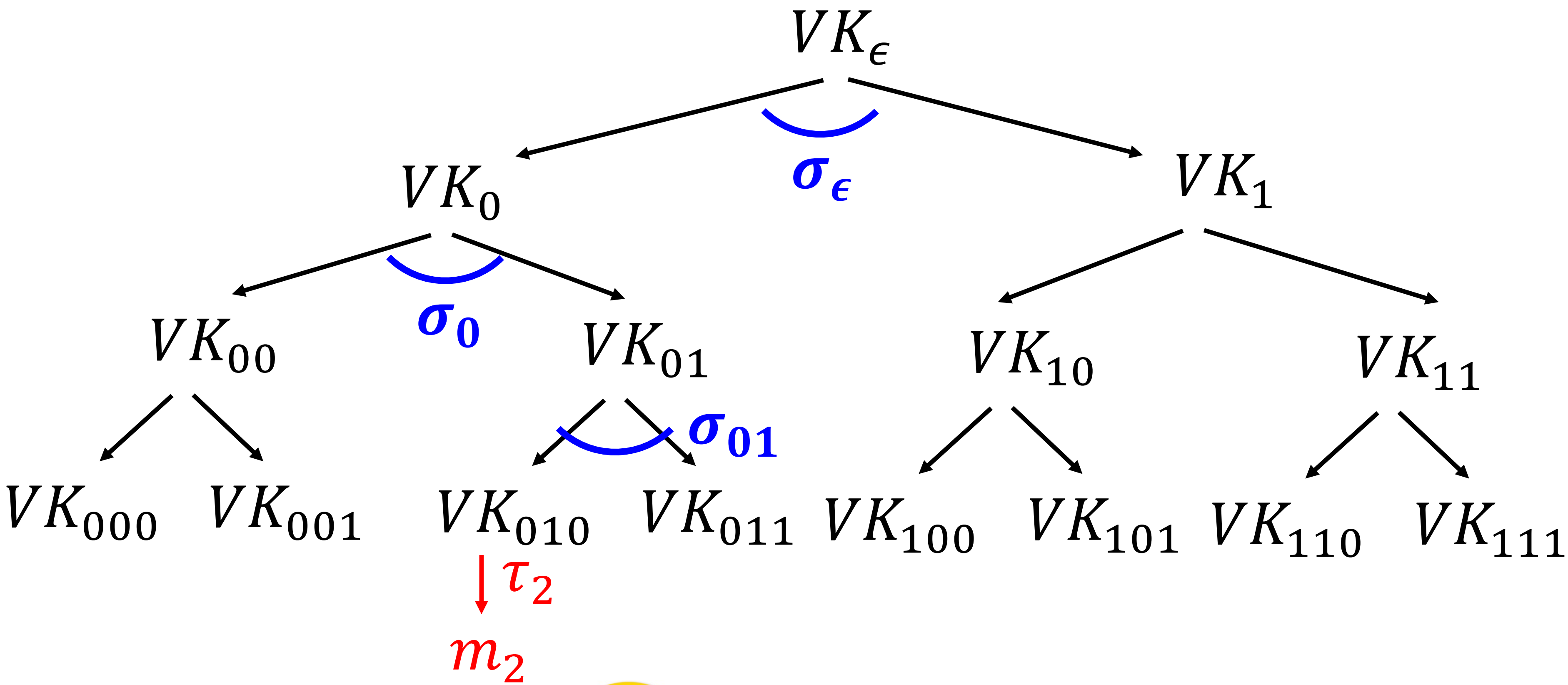
Step 2. How to Shrink the signatures.



GOOD NEWS: 

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

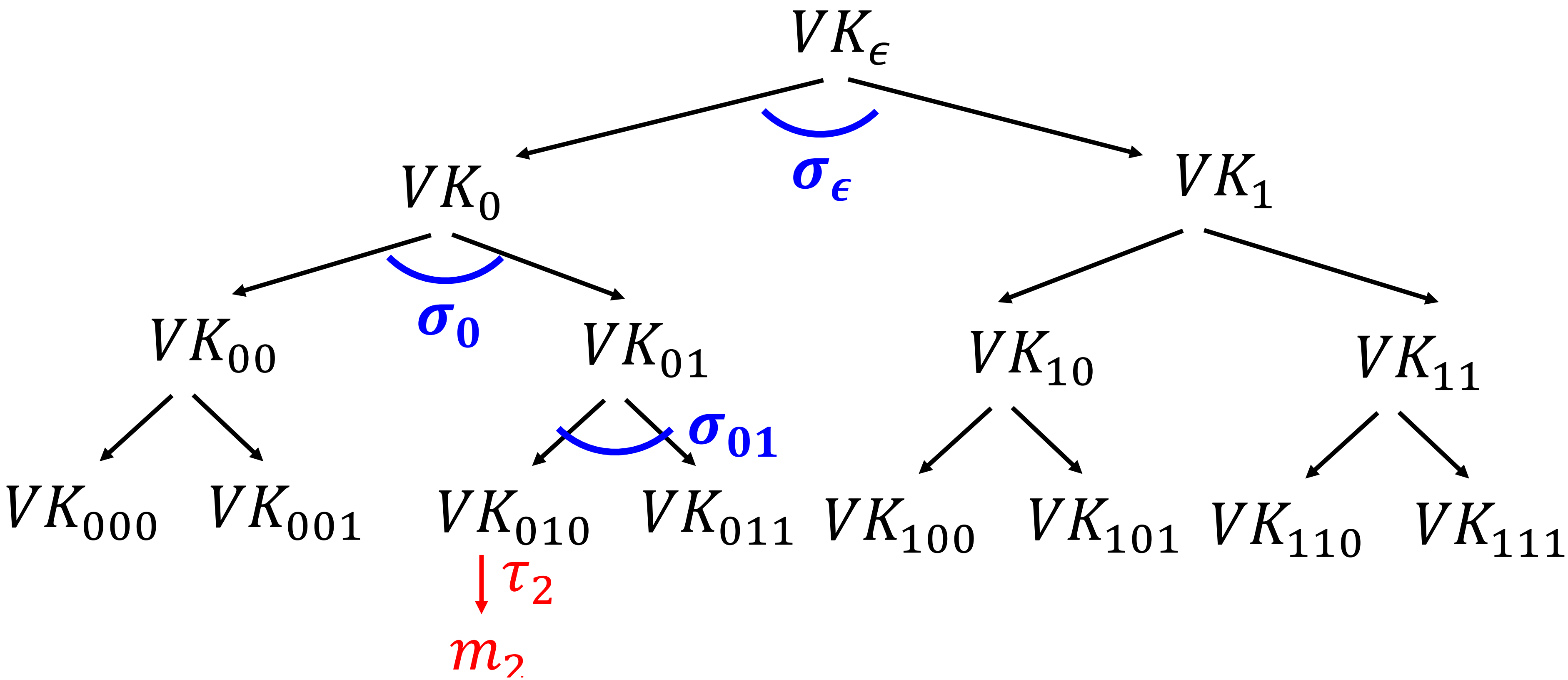
Step 2. How to Shrink the signatures.



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

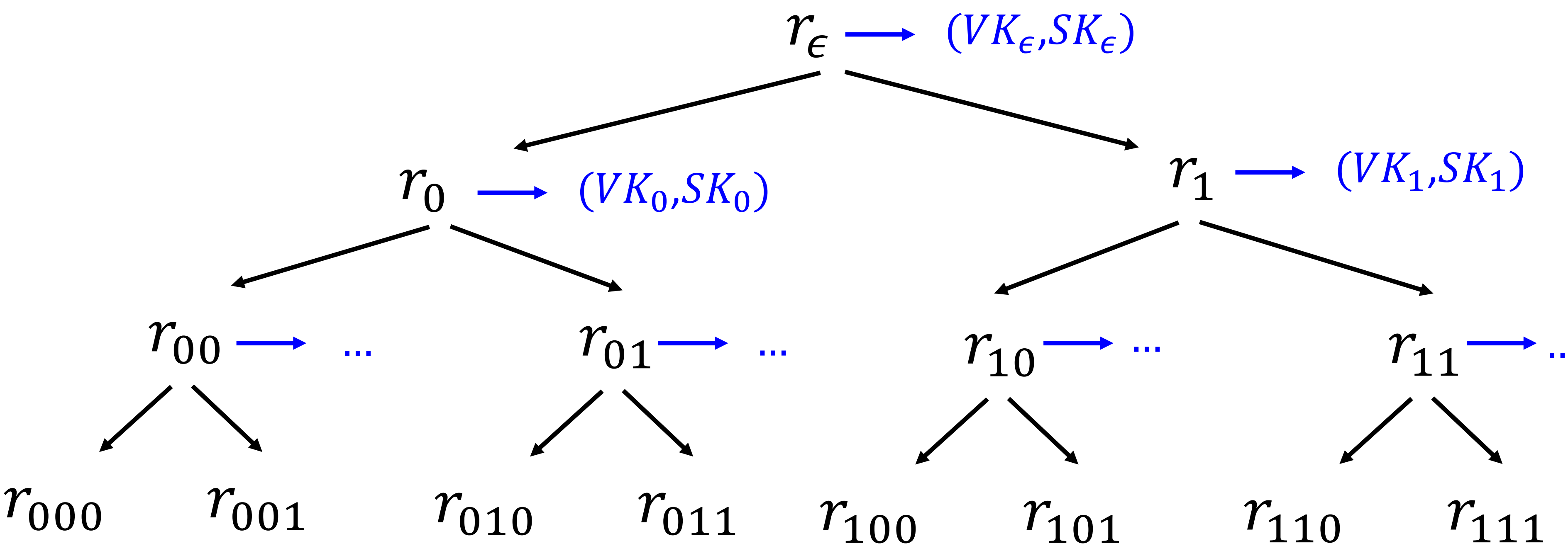
In four+ steps

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.



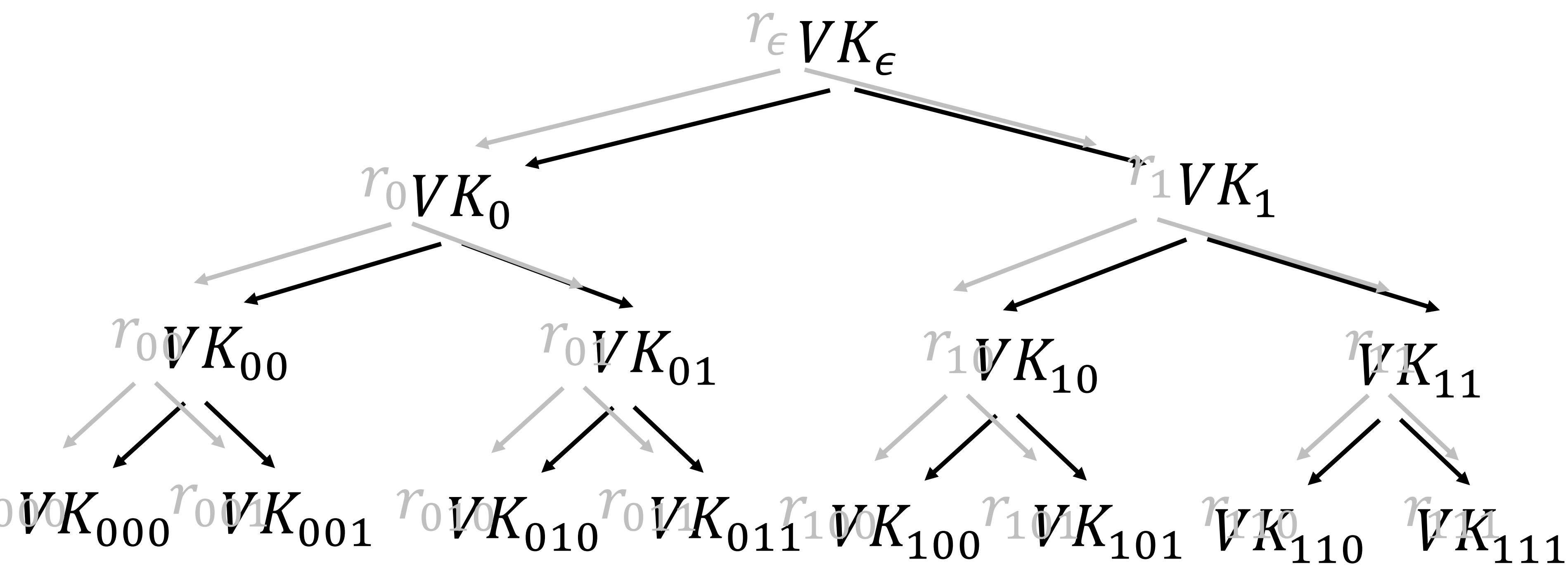
Tree of pseudorandom values:

The signing key is a PRF key K.

Populate the nodes with $r_x = PRF(K, x)$.

Use r_x to derive the keys $(VK_x, SK_x) \leftarrow Gen(1^\lambda; r_x)$.

Step 3. Pseudorandom Signature Trees.



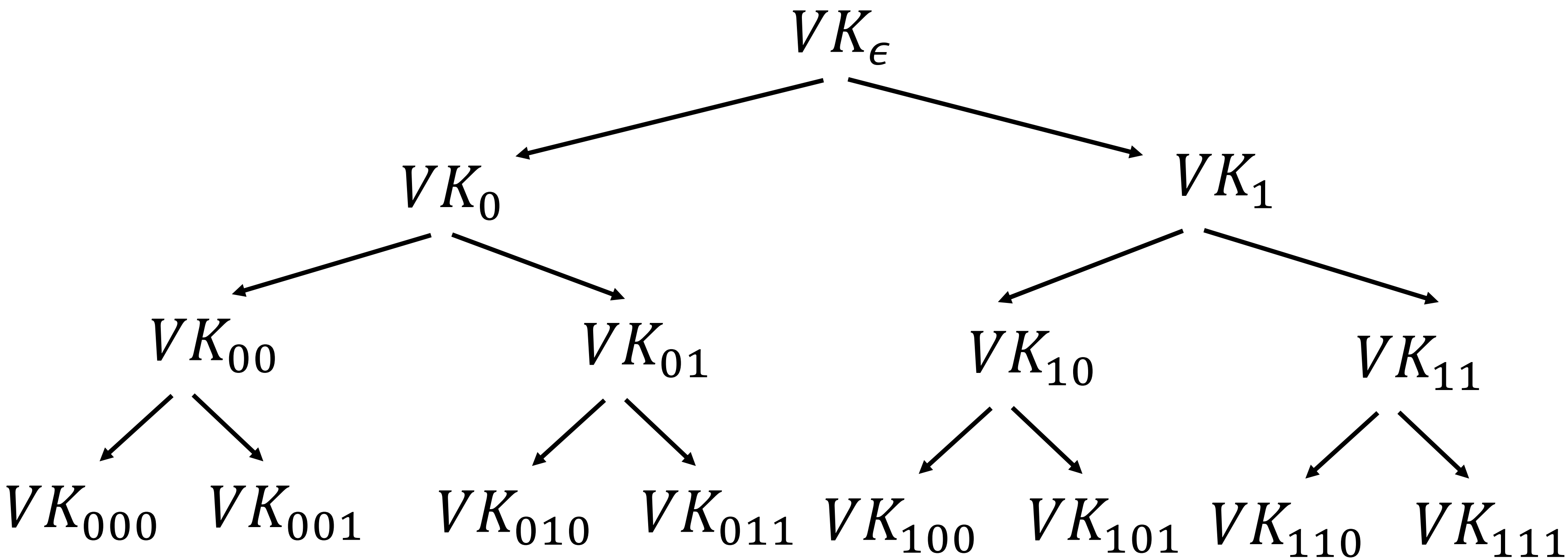
Tree of pseudorandom values:

The signing key is a PRF key K.

Populate the nodes with $r_x = PRF(K, x)$.

Use r_x to derive the keys $(VK_x, SK_x) \leftarrow Gen(1^\lambda; r_x)$.

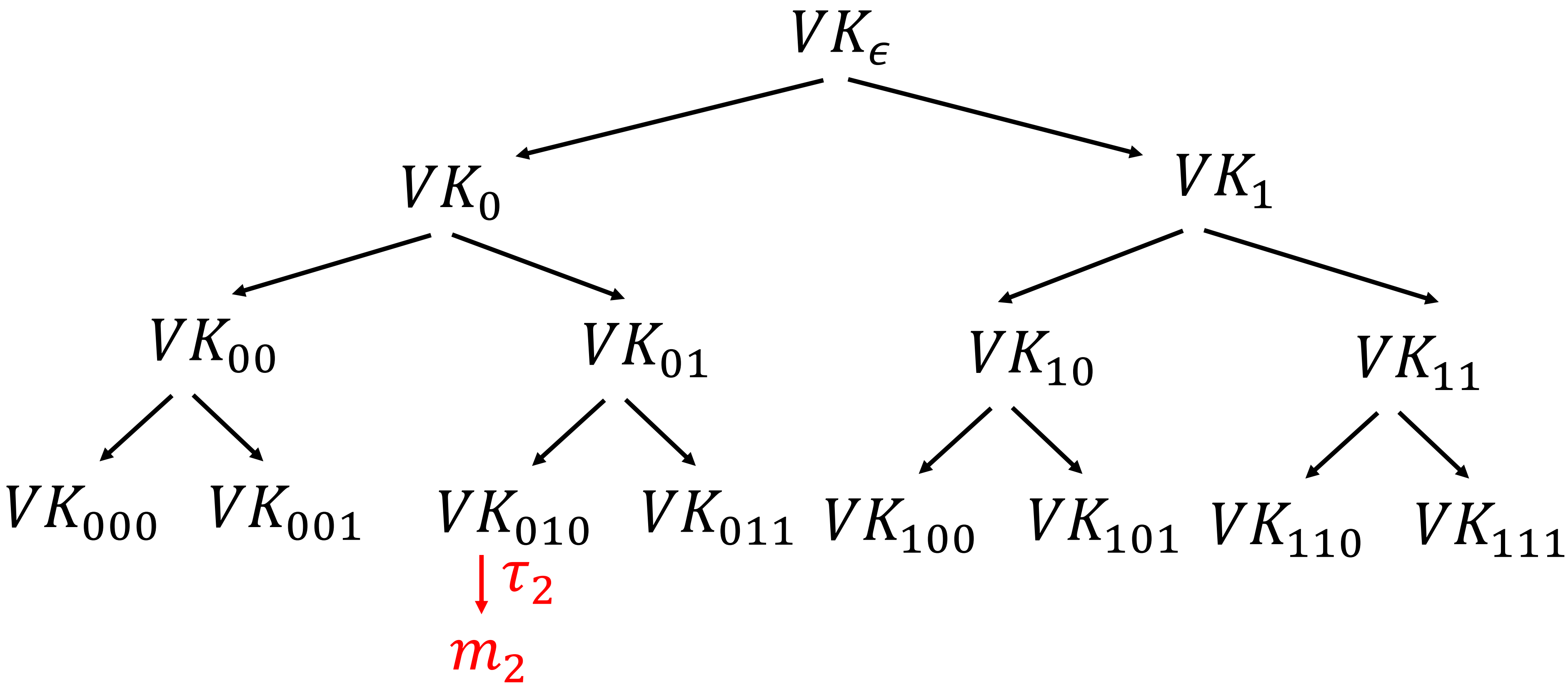
Step 3. Pseudorandom Signature Trees.



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

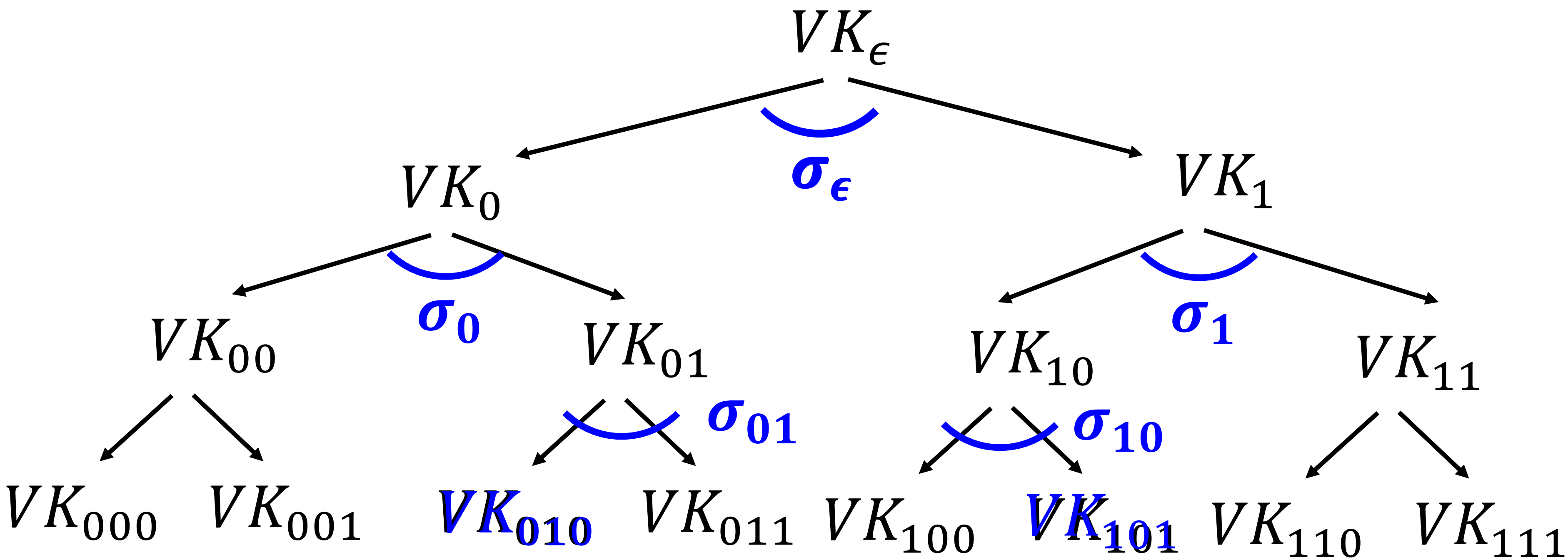
Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

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



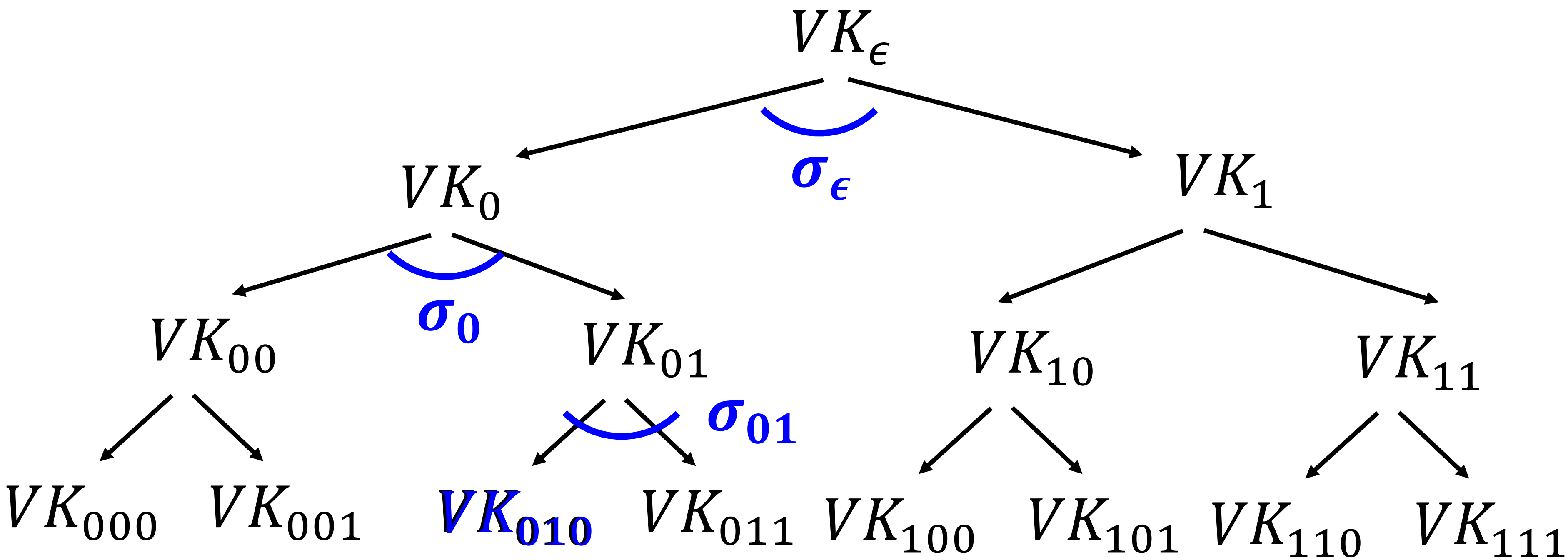
Signature of a message $\boldsymbol{m}$:

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

$$\sigma_r \leftarrow \mathrm{Sign}(SK_r, m)$$

Output $(\mathrm{r}, \sigma_r,$ authentication path for $VK_r)$

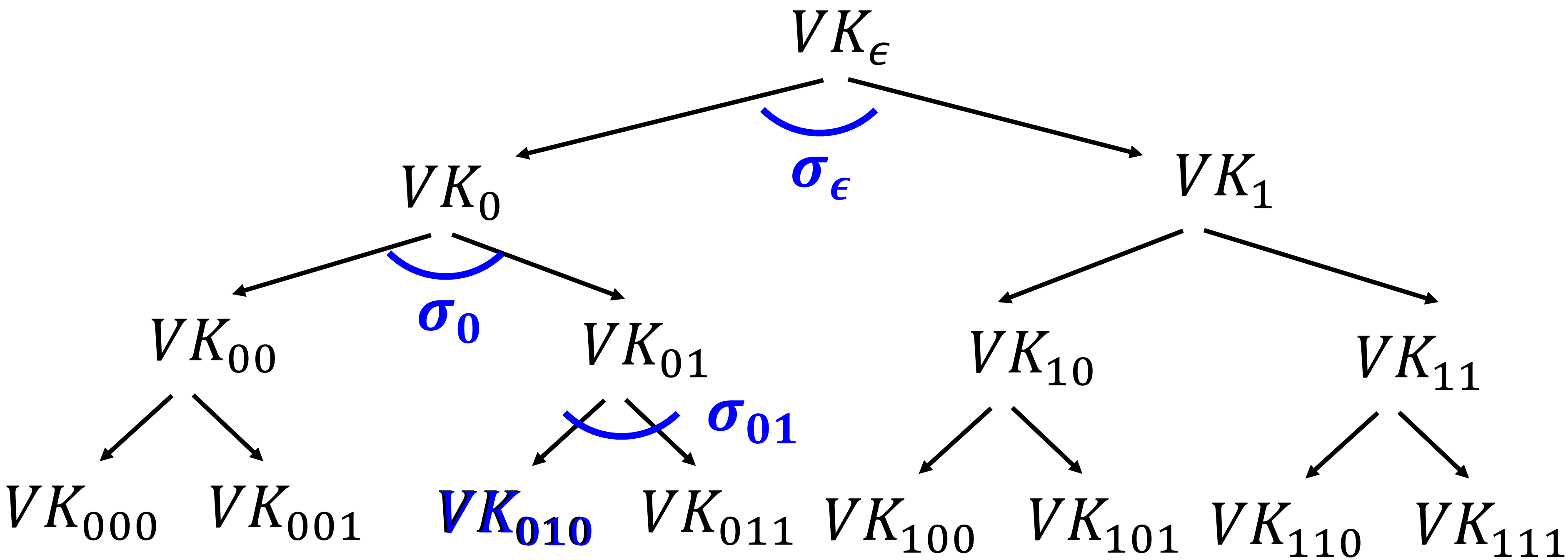
Step 4. Statelessness via Randomization



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***

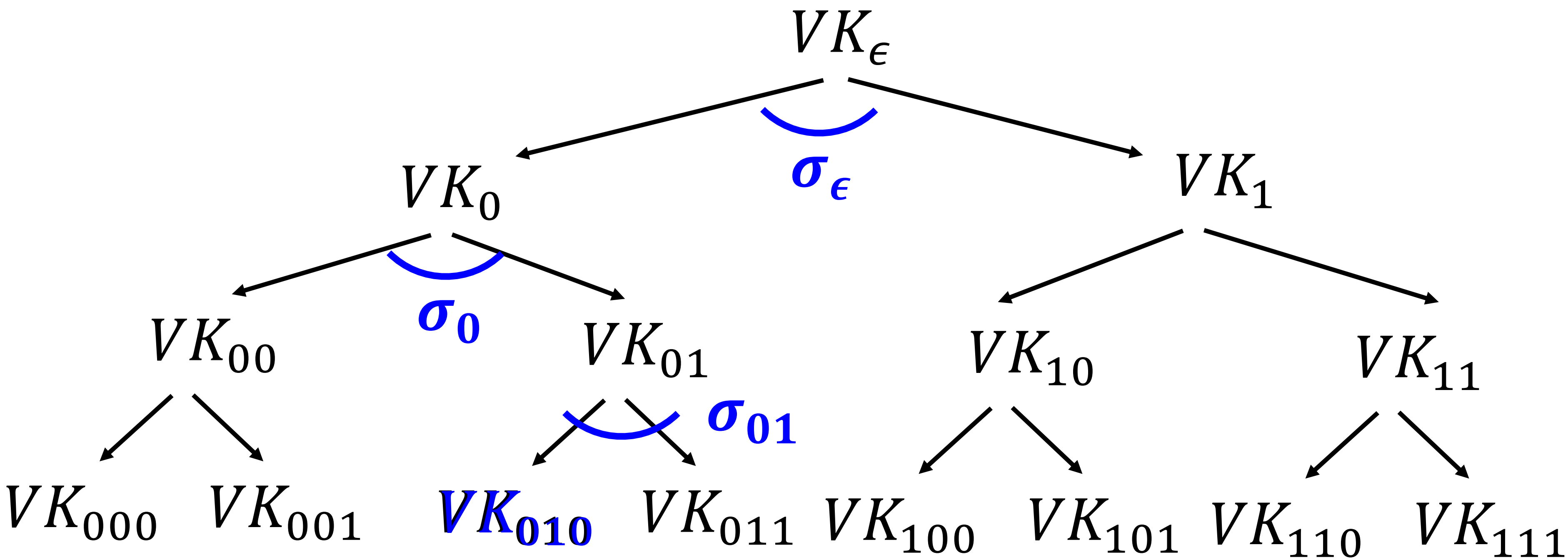
Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

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 $r = PRF(K', \quad)$

That’s it for the construction.