CIS 5560

Cryptography
Lecture 18

Course website:

pratyushmishra.com/classes/cis-5560-s24/

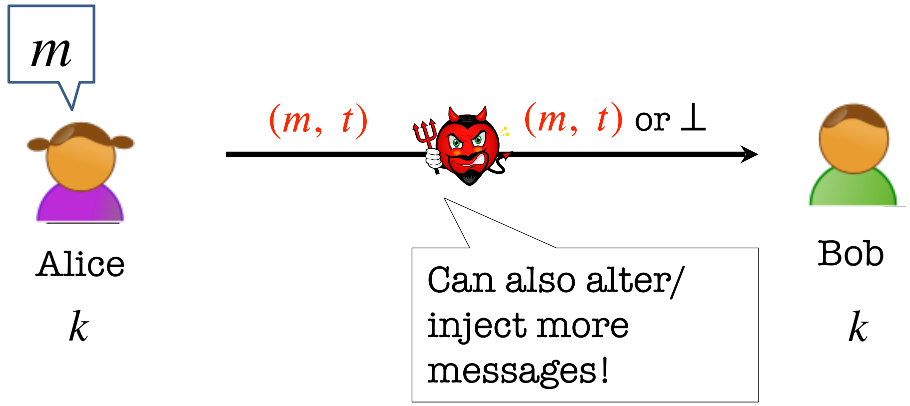
Slides adapted from Dan Boneh and Vinod Vaikuntanathan

Announcements

- **HW 8 out Wednesday evening**
  - Due **Wednesday Apr 10** at 11:59PM on Gradescope
  - Covers
    - RSA
    - little bit of IND-CCA PKE

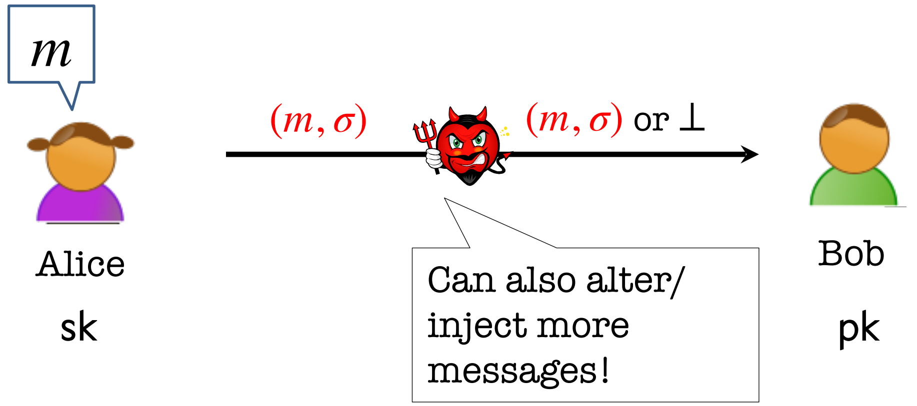
Recap of last lecture

Symmetric-key Message Authentication



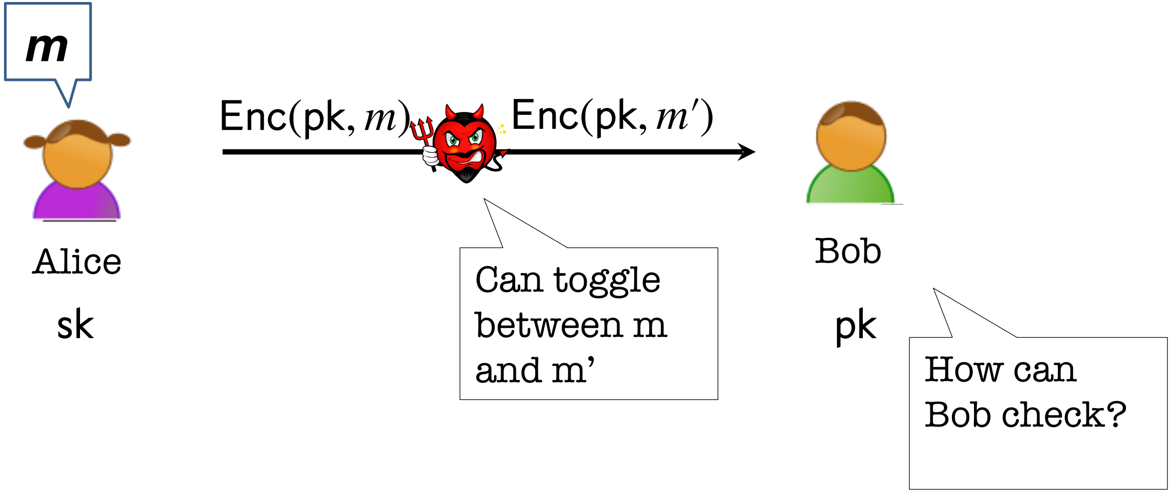
We want Alice to generate a tag for the message m which is **hard to generate** without the secret key k.

Public-key Message Authentication?



We want Alice to generate a signature for the message m which is **hard to forge** without the secret/signing key sk.

Does PKE not solve this?



Anybody can encrypt, and no way for recipient to check.

New primitive: Digital Signatures

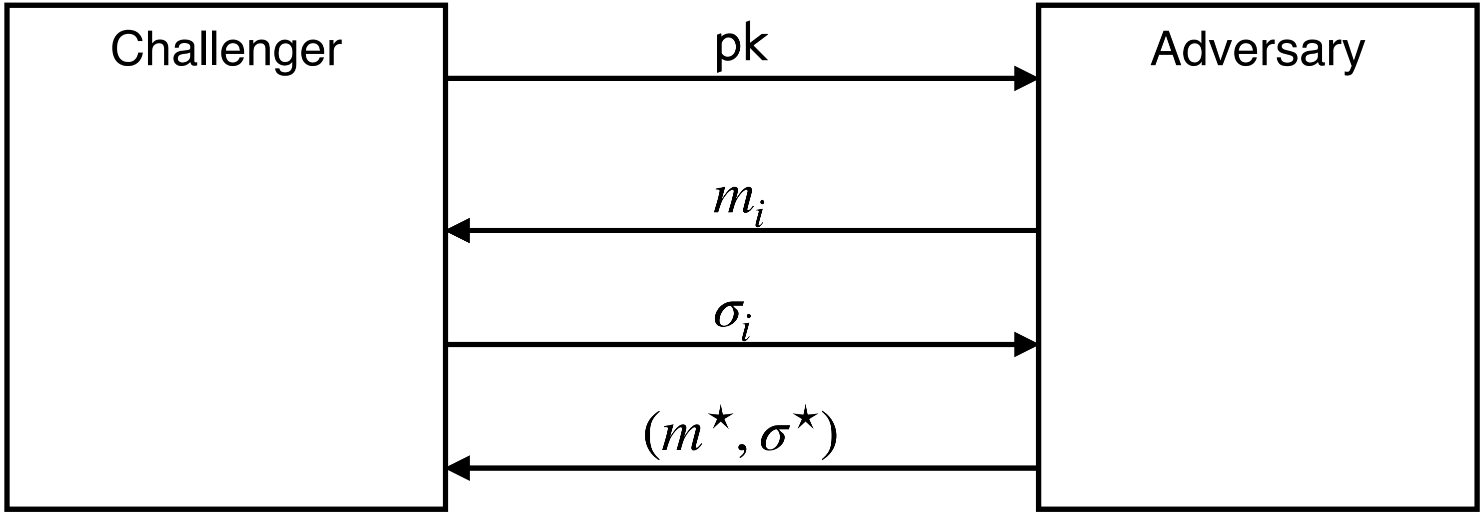
Digital Signatures: Definition

A triple of PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ such that

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

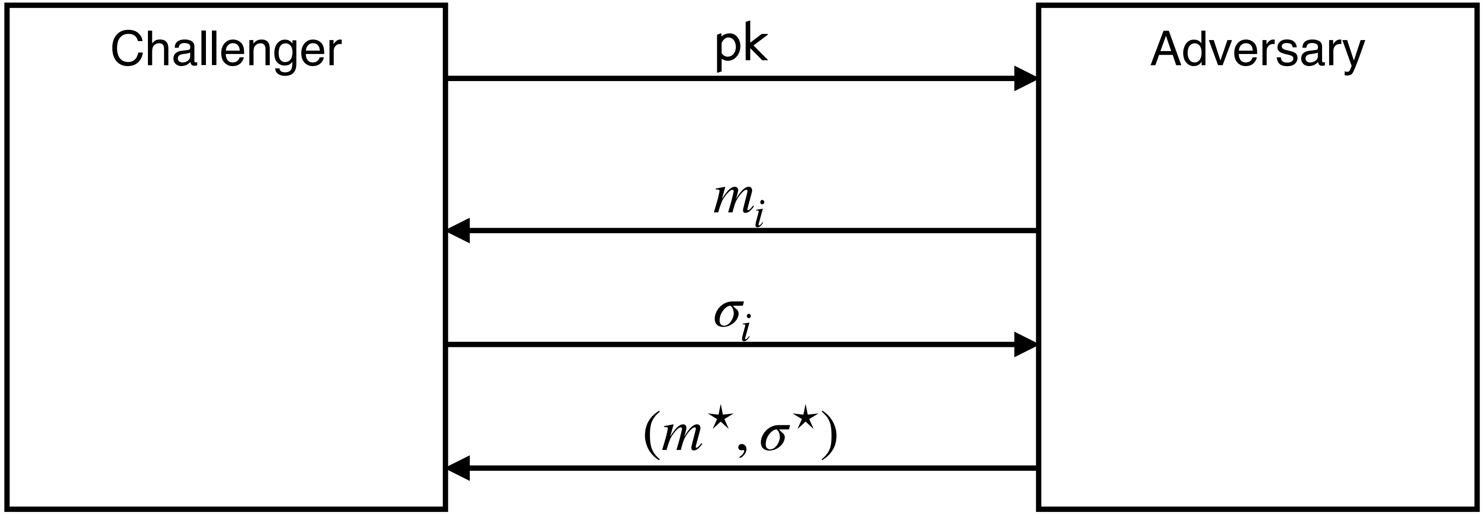
Correctness: For all vk, sk, m: $\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$

EUF-CMA for Signatures



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

Strong EUF-CMA for Signatures



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

Digital Signatures vs. MACs

Signatures	**MACs**
n users require n key-pairs	n users require n^2 keys
Publicly Verifiable	Privately Verifiable
Transferable	**Not** Transferable
Provides Non-Repudiation (is this a good thing or a bad thing?)	Does not provide Non-Rep.

Let (Gen, Sign,V) be a signature scheme.

Suppose an attacker is able to find $m_0 \neq m_1$ such that

$$\mathbf{V(pk, m_0, \sigma) = V(pk, m_1, \sigma)} \quad \text{for all } \sigma \text{ and keys } (pk, sk) \leftarrow \text{Gen}$$

Can this signature be secure?

- ○ Yes, the attacker cannot forge a signature for either m_0 or m_1
- ○ No, signatures can be forged using a chosen msg attack
- ○ It depends on the details of the scheme

Alice generates a (pk,sk) and gives pk to her bank.

Later Bob shows the bank a message m="**pay Bob 100$**" properly signed by Alice, i.e. Verify(pk,m,sig) = 1

Alice says she never signed m. Is Alice lying?

- ○ Alice is lying: existential unforgeability means Alice signed m and therefore the Bank should give Bob 100$ from Alice's account
- ○ Bob could have stolen Alice's signing key and therefore the bank should not honor the statement
- ○ What a mess: the bank will need to refer the issue to the courts

Applications

Applications

Code signing:

- Software vendor signs code
- Clients have vendor's pk.   Install software if signature verifies.

software vendor

sk

initial software install (pk)

[software udate #1 , sig]

[software udate #2 , sig]

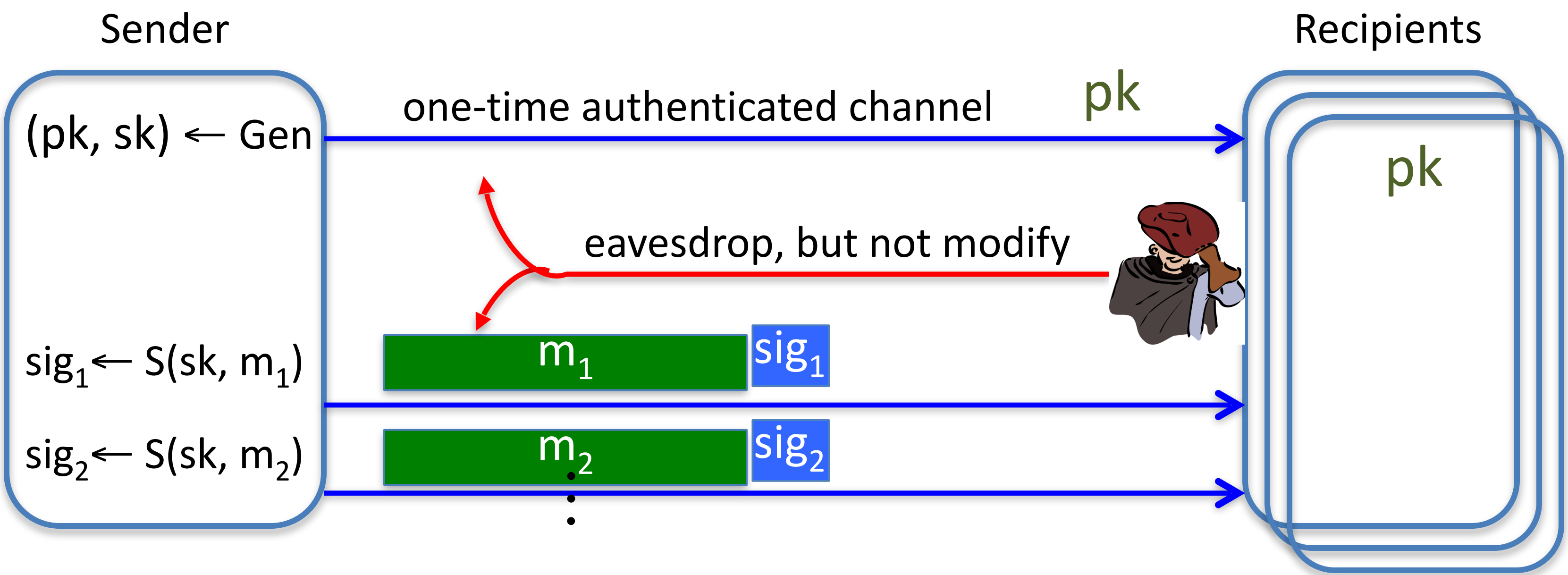
many clients

More generally:

One-time authenticated channel (non-private, one-directional)

$\Longrightarrow$ many-time authenticated channel

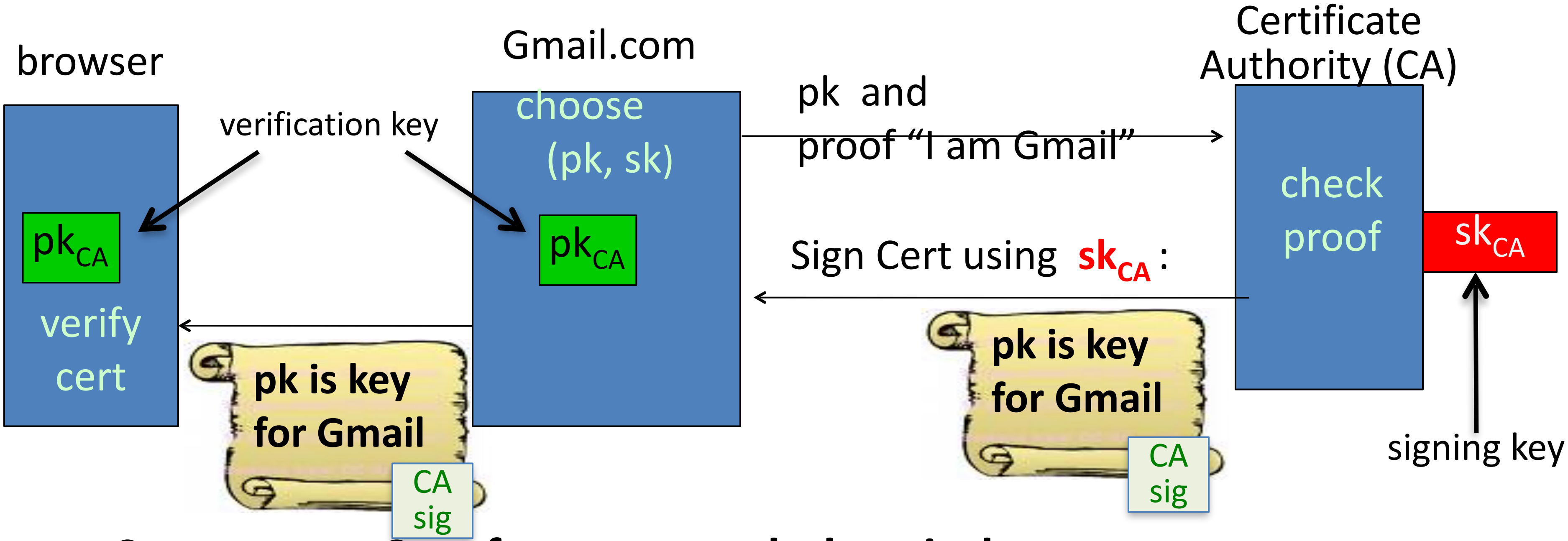
Initial software install is authenticated, but not private



Important application: Certificates

Problem: browser needs server's public-key to setup a session key

Solution: server asks trusted 3rd party (CA) to sign its public-key pk



Server uses Cert for an extended period (e.g. one year)

Certificates: example

Important fields:

Serial Number	5814744488373890497
Version	3
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters	none
Not Valid Before	Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time
Not Valid After	Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 71 6C DD E0 0A C9 76 ...
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : 8A 38 FE D6 F5 E7 F6 59 ...

Equifax Secure Certificate Authority
↳ GeoTrust Global CA
↳ Google Internet Authority G2
↳ mail.google.com

mail.google.com
Issued by: Google Internet Authority G2
Expires: Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
This certificate is valid

Details

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google Inc
Common Name	mail.google.com
Issuer Name	
Country	US
Organization	Google Inc
Common Name	Google Internet Authority G2

What entity generates the CA's secret key sk_{CA} ?

- the browser
- Gmail
- the CA
- the NSA

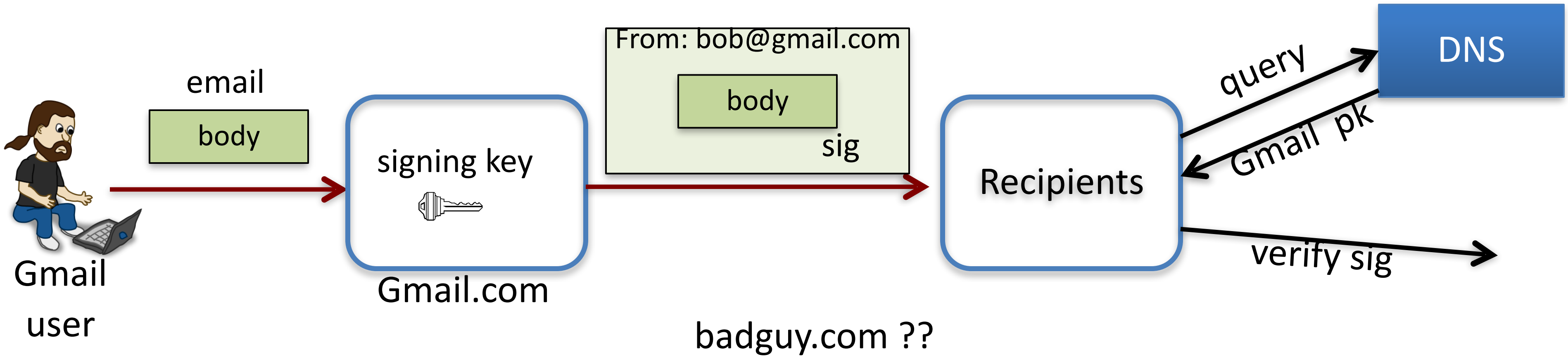
Signing email: DKIM (domain key identified mail)

Problem: bad email claiming to be from **someuser@gmail.com**

but in reality, mail is coming from domain **badguy.com**

⇒ Incorrectly makes gmail.com look like a bad source of email

Solution: **gmail.com** (and other sites) sign every outgoing mail



When to use signatures

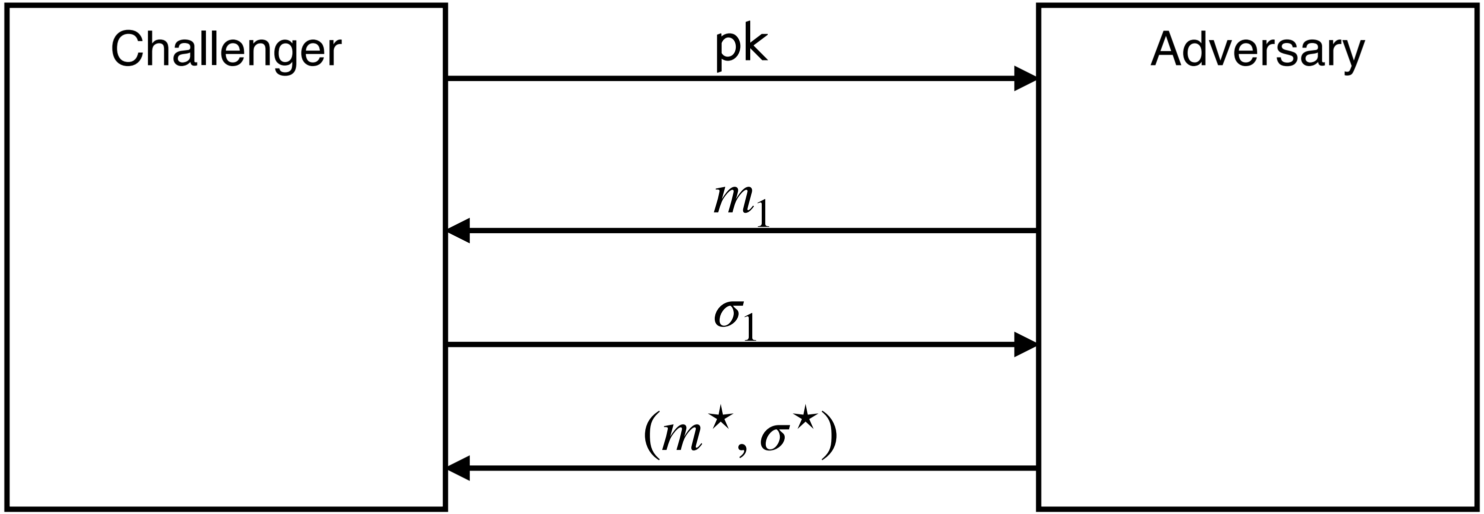
Generally speaking:

- If one party signs and **one** party verifies: **use a MAC**
  - Often requires interaction to generate a shared key
  - Recipient can modify the data and re-sign it before passing the data to a 3rd party

- If one party signs and **many** parties verify: **use a signature**
  - Recipients **cannot** modify received data before passing data to a 3rd party (non-repudiation)

Constructions

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



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

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 $f(\sigma) = y_b$

<u>**Claim**</u>: 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. $\sigma = (z_{1,m_1}, z_{2,m_2}, \ldots, z_{n,m_n})$

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.

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

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

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

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

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	pk_0

Alice starts with a secret signing Key sk_0

When signing the next message m_2

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

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

Output ???

$$\mathrm{pk}_0 \xrightarrow{\sigma_1} \overset{m_1}{\mathrm{pk}_1}$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	pk_0

Alice starts with a secret signing Key sk_0

When signing the next message m_2

Generate a new pair (sk_2, pk_2)

Produce signature $\sigma_2 \leftarrow \text{Sign}(sk_1, m_2 || pk_2)$

Output $pk_2 || \sigma_2$??

$$pk_0 \xrightarrow{\sigma_1} \overset{m_1}{pk_1}$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	pk_0

Alice starts with a secret signing Key sk_0

When signing the next message m_2

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

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

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

$$
\begin{array}{ccc}
 & & m_1 \\
\mathrm{pk}_0 & \xrightarrow{\sigma_1} & \mathrm{pk}_1
\end{array}
$$

Step 1: Stateful Many-time Signatures

Idea: Signature Chains.

Alice	pk_0

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.

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

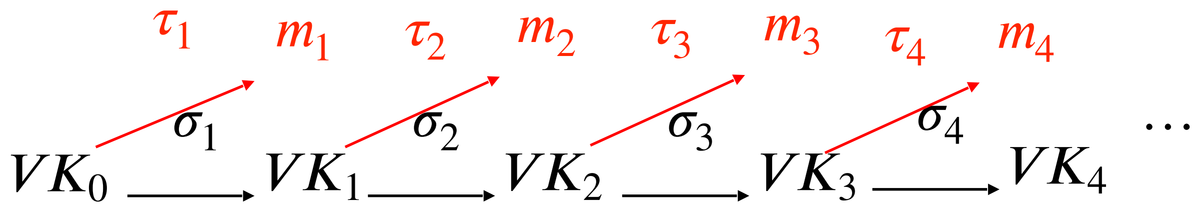
Step 1: Stateful Many-time Signatures

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***

Alice	VK_ϵ

Step 2. How to Shrink the signatures.

$$VK_\epsilon$$

Alice	VK_ϵ

Step 2. How to Shrink the signatures.

VK_ϵ

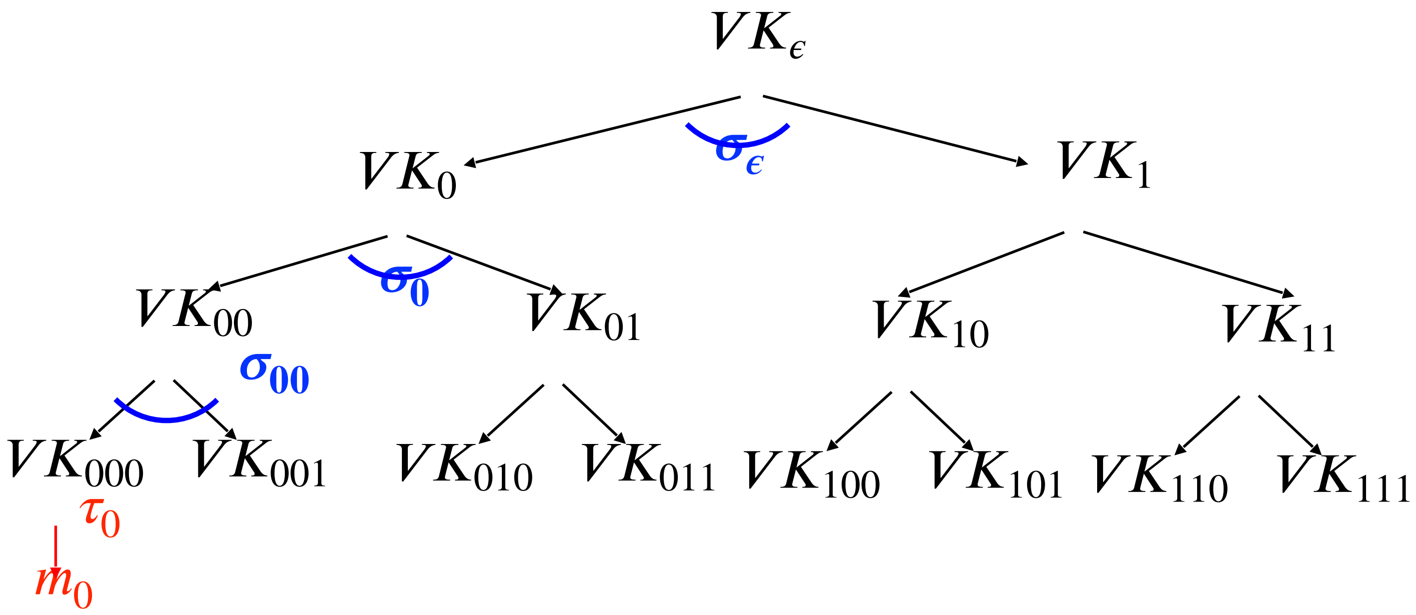
VK_0 VK_1

VK_{00} VK_{01} VK_{10} VK_{11}

VK_{000} VK_{001} VK_{010} VK_{011} VK_{100} VK_{101} VK_{110} VK_{111}

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.

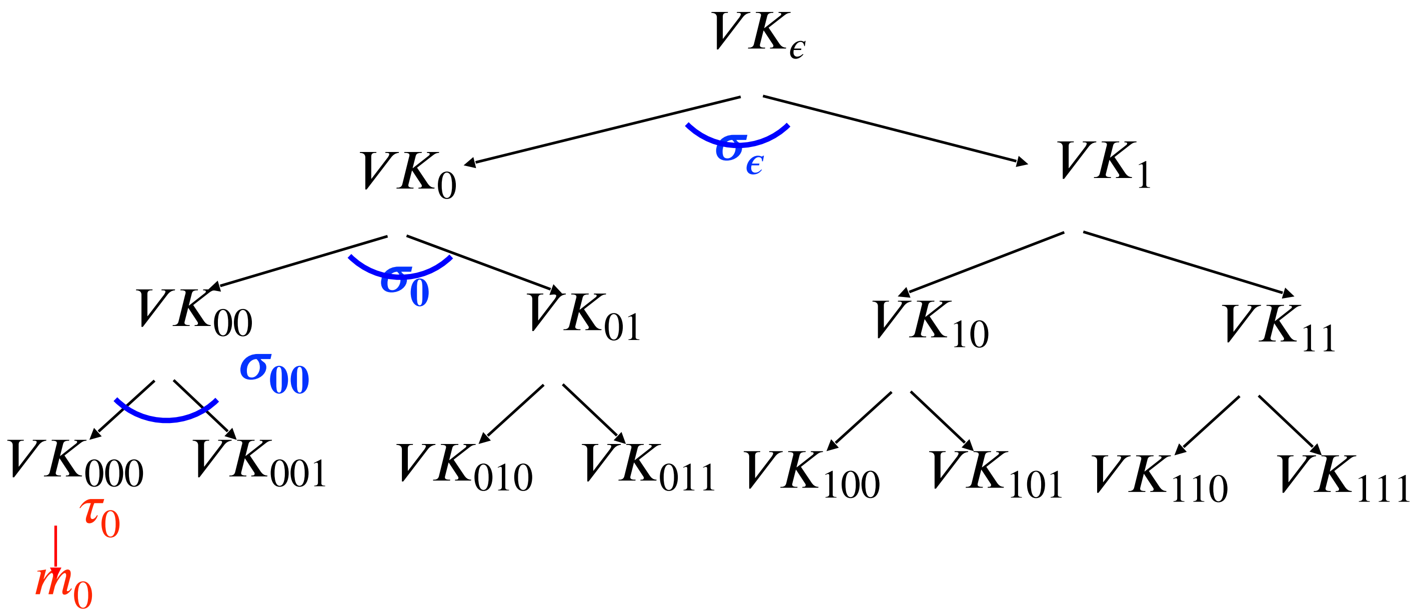


Signature of the first message m_0:

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 m_0:

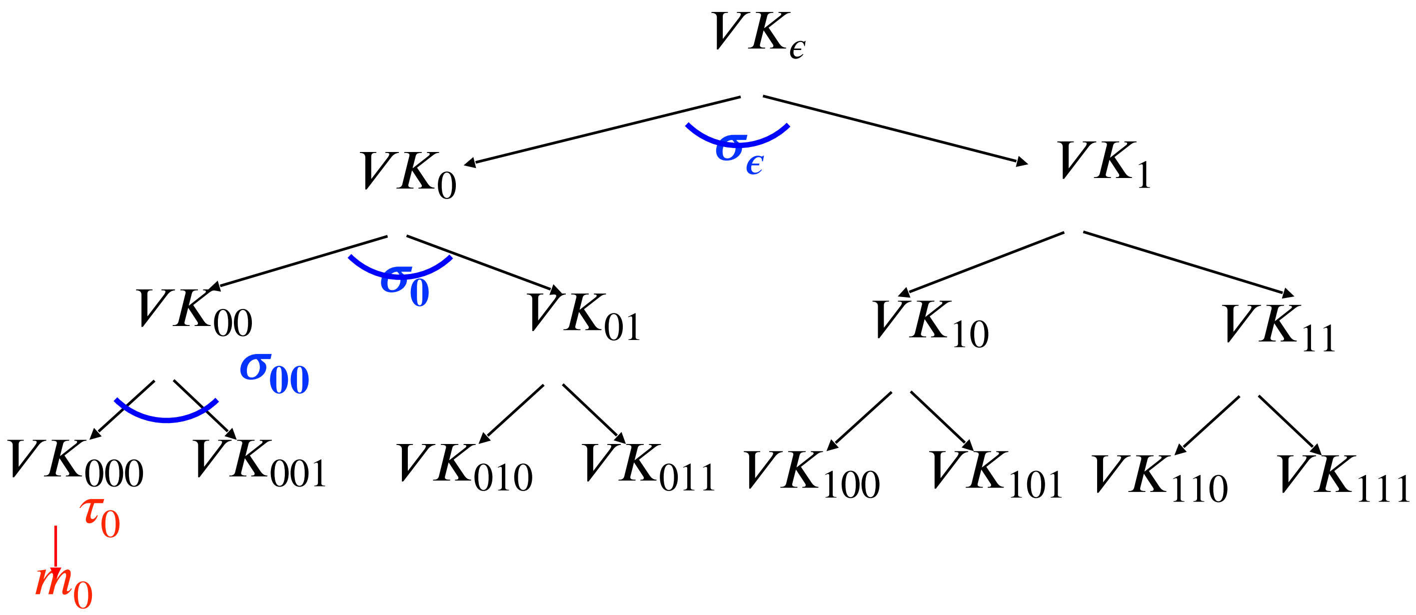
$(\sigma_\epsilon \leftarrow \mathrm{Sign}(SK_\epsilon, VK_0||VK_1),$

$\sigma_0 \leftarrow \mathrm{Sign}(SK_0, VK_{00}||VK_{01}),$

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

$\tau_0 \leftarrow \mathrm{Sign}(SK_{000}, m_0))$

Step 2. How to Shrink the signatures.



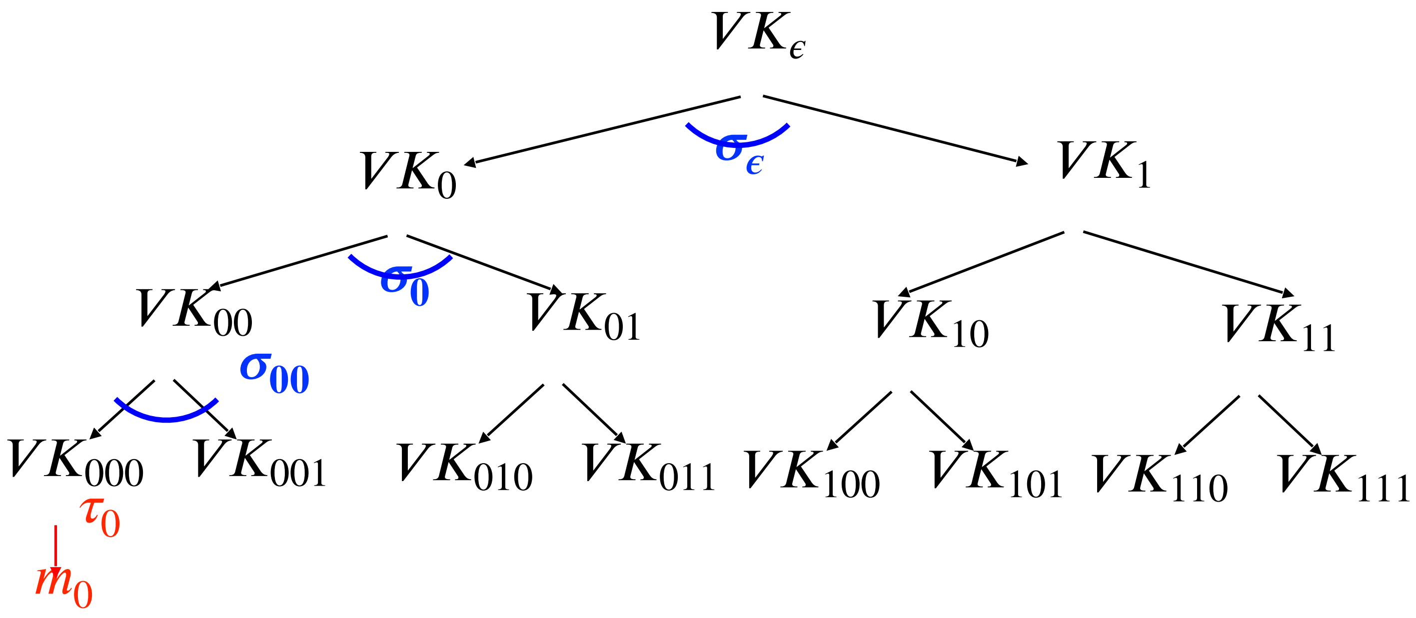
Authentication Path for VK_{000}:

$(\sigma_\epsilon \leftarrow \mathrm{Sign}(SK_\epsilon, VK_0 || VK_1),$

$\sigma_0 \leftarrow \mathrm{Sign}(SK_0, VK_{00} || VK_{01})$

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

Step 2. How to Shrink the signatures.

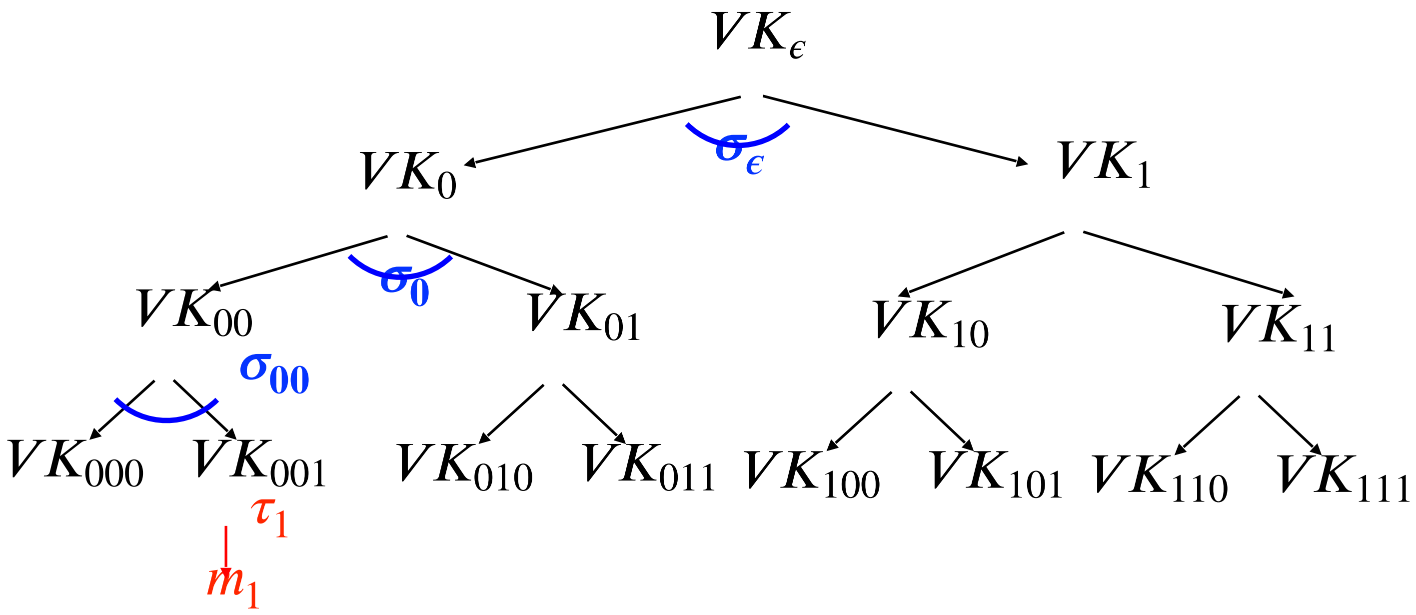


Signature of the first message m_0:

(Authentication path for VK_{000},

$\tau_0 \leftarrow \mathrm{Sign}(SK_{000}, m_0)$)

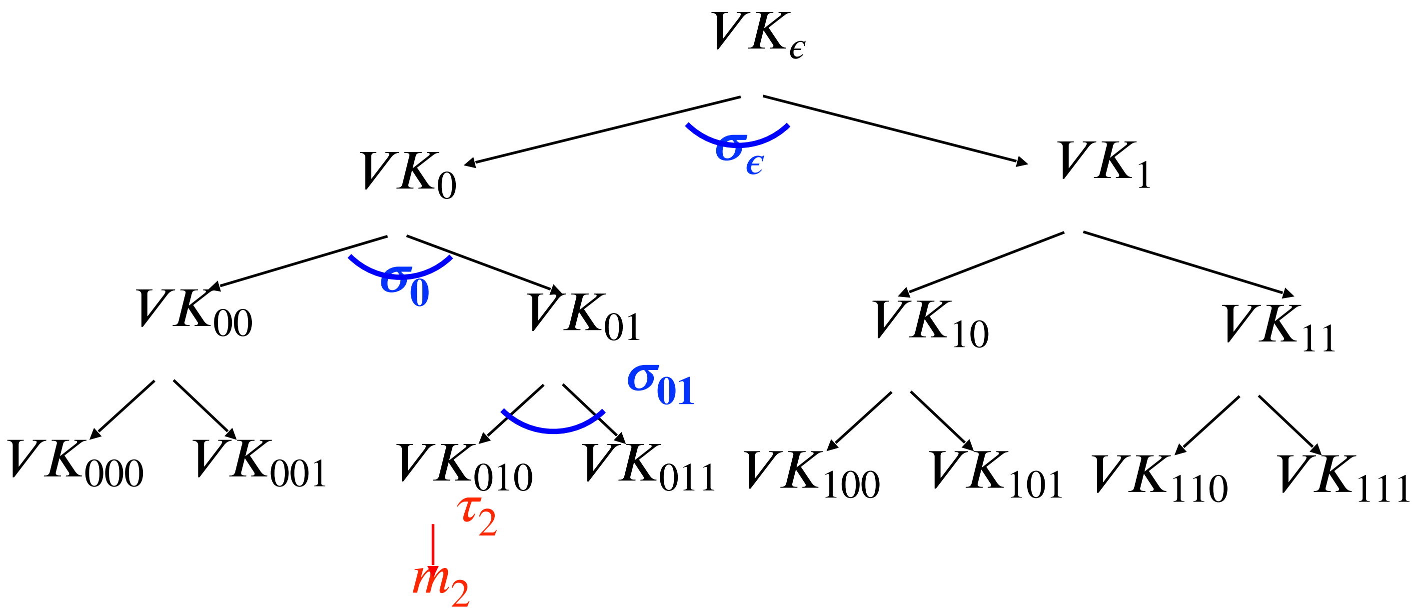
Step 2. How to Shrink the signatures.



Signature of the second message m_1:
(Authentication path for VK_{001},

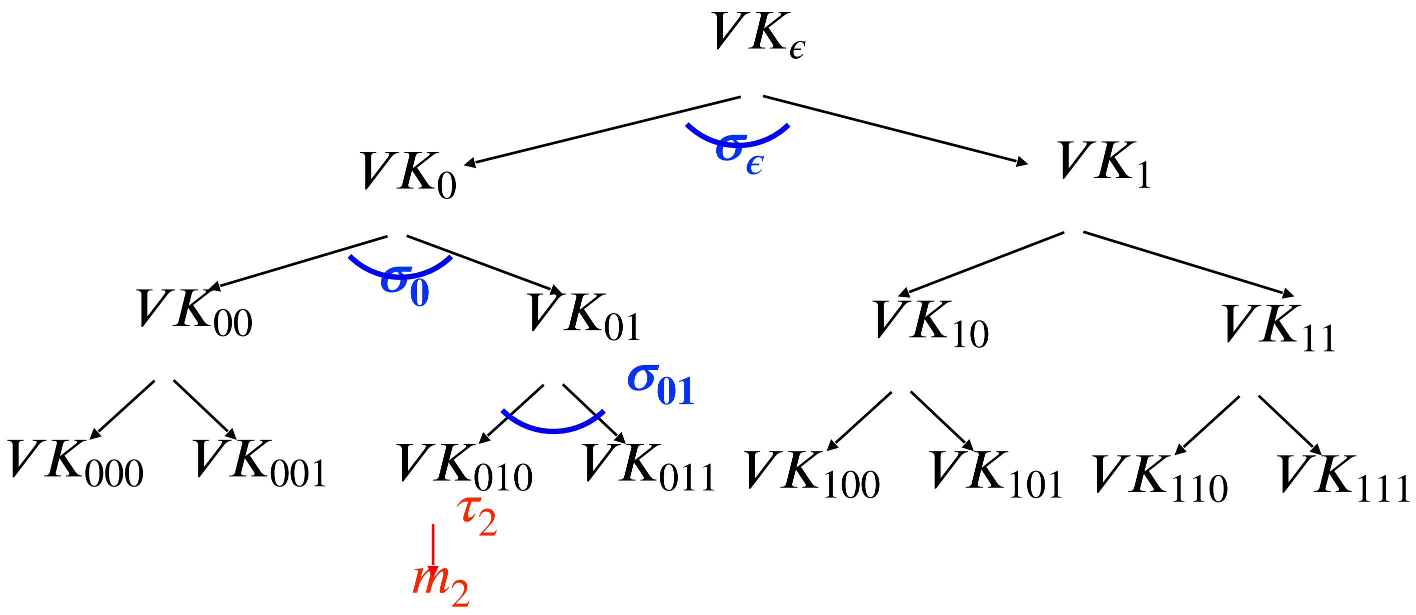
$\tau_0 \leftarrow \mathrm{Sign}(SK_{001}, m_1)$)

Step 2. How to Shrink the signatures.



Signature of the third message m_2:
(Authentication path for VK_{010},
$\tau_2 \leftarrow \mathrm{Sign}(SK_{010}, m_2)$)

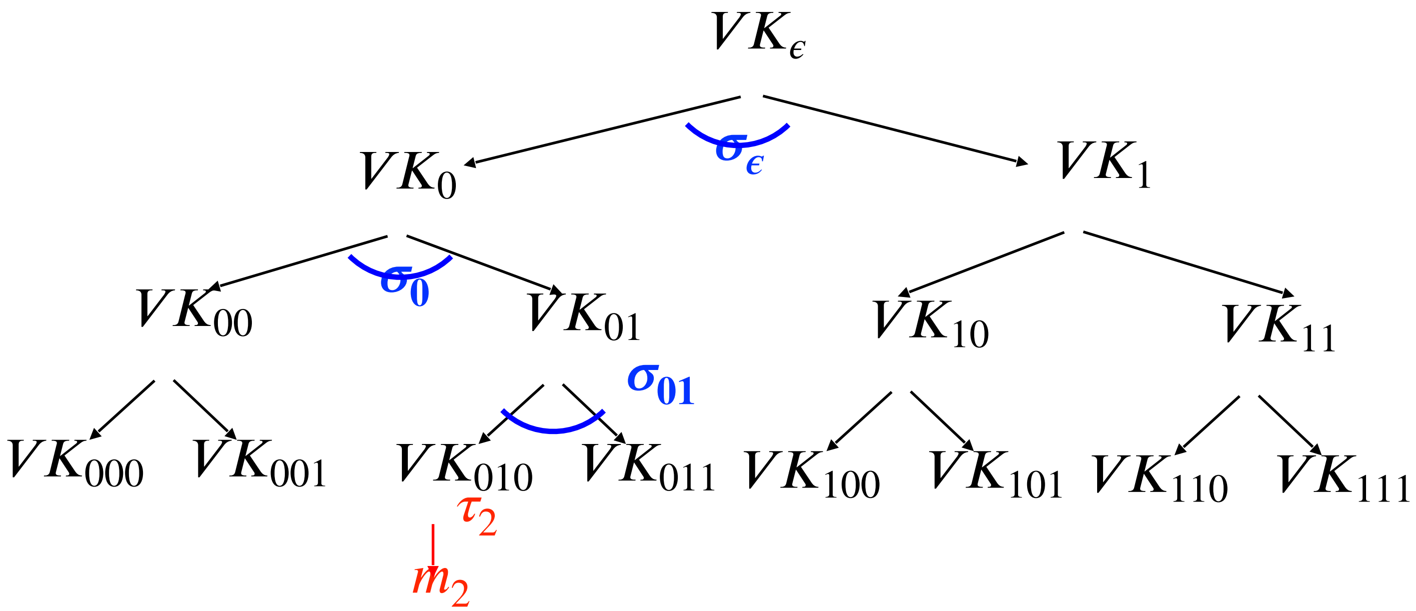
Step 2. How to Shrink the signatures.



GOOD NEWS:

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

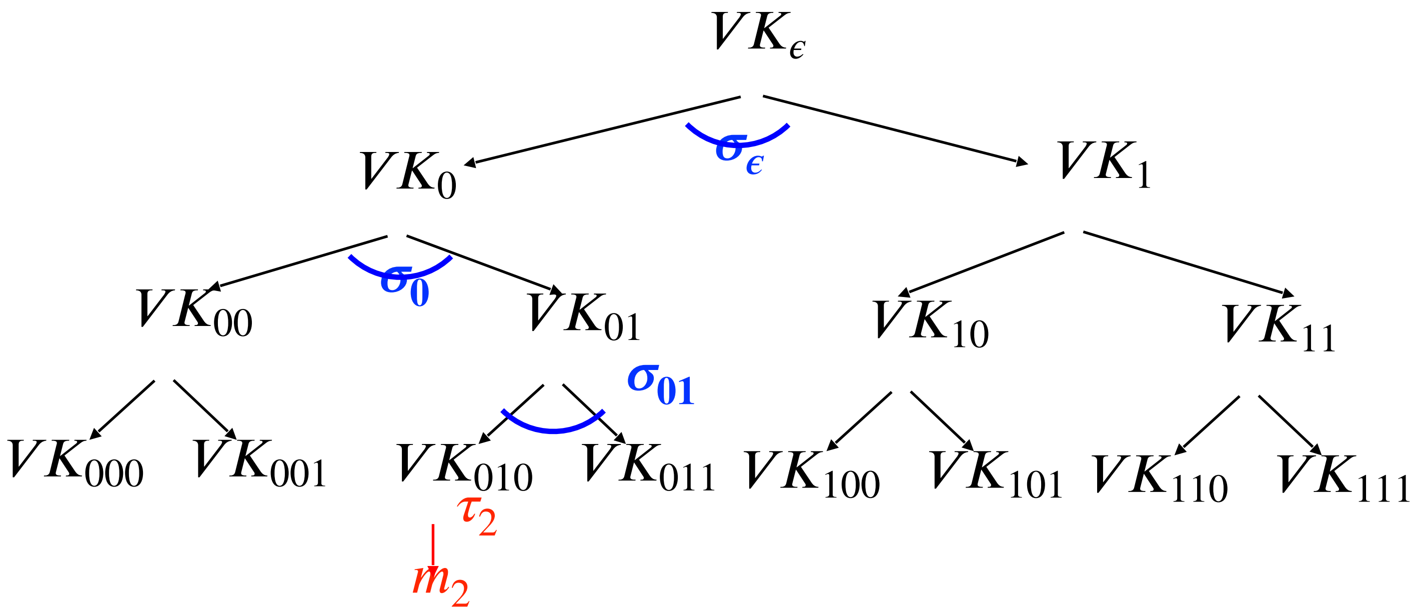
Step 2. How to Shrink the signatures.



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

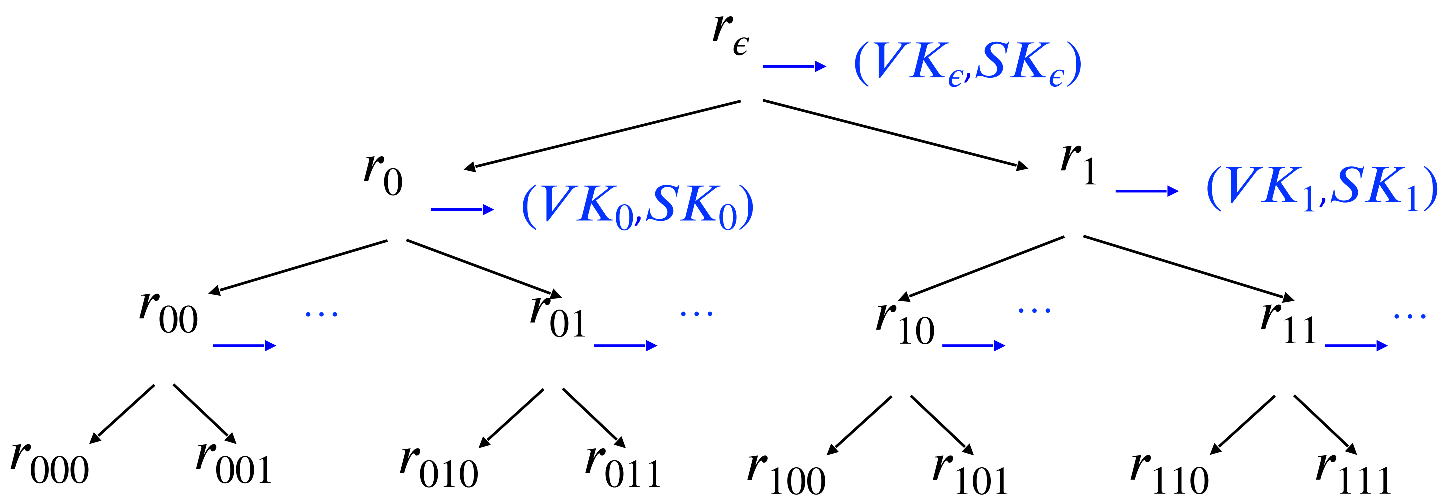
In four+ steps

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.



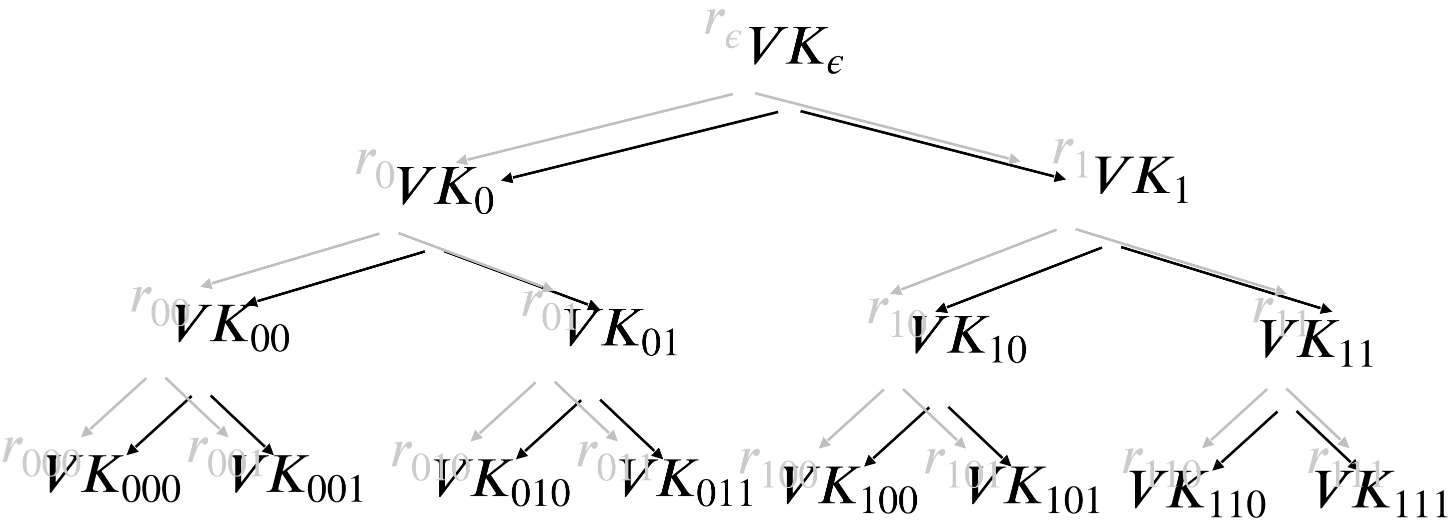
Tree of pseudorandom values:

The signing key is a PRF key K.

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.

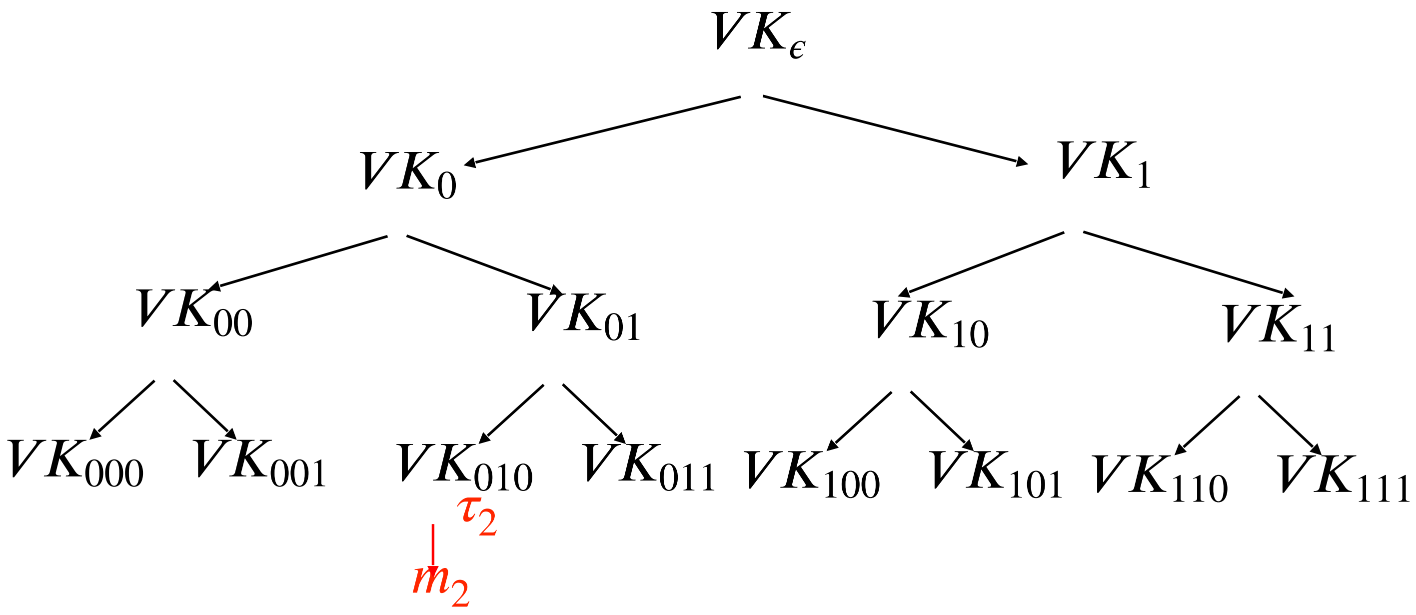
$$\begin{tikzcd}[column sep=tiny]
 & & & VK_{\epsilon} \arrow[dll] \arrow[drr] & & & & \\
 & VK_0 \arrow[dl] \arrow[dr] & & & & VK_1 \arrow[dl] \arrow[dr] & & \\
VK_{00} \arrow[d] \arrow[d] & & VK_{01} & & VK_{10} & & VK_{11} &
\end{tikzcd}$$

VK_{000} VK_{001} VK_{010} VK_{011} VK_{100} VK_{101} VK_{110} VK_{111}

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

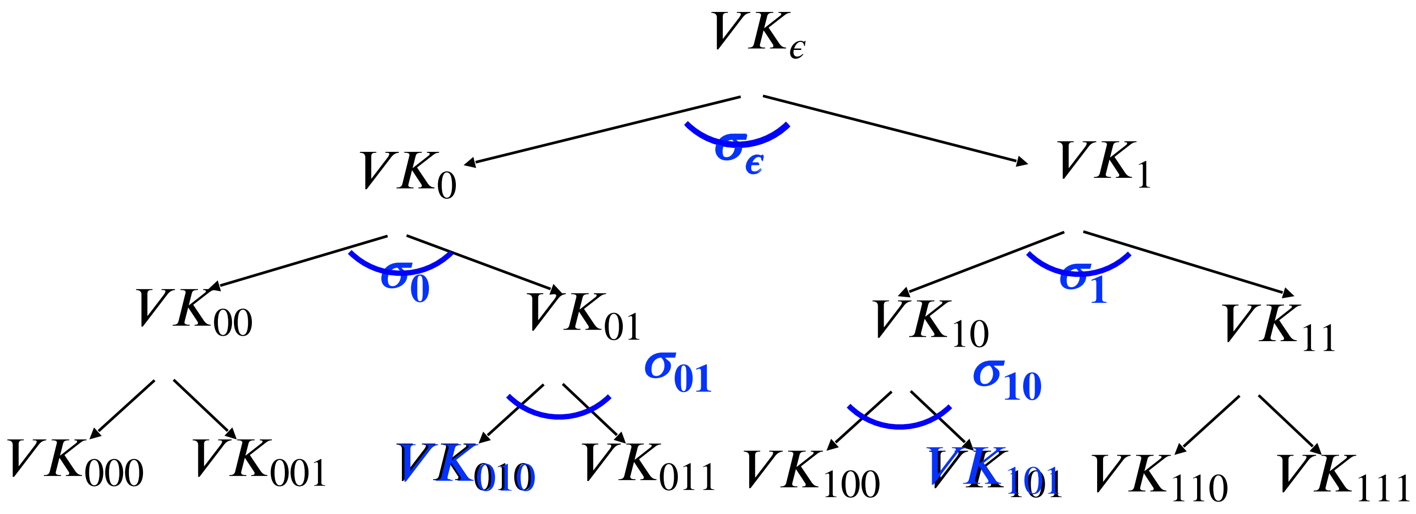
Step 1. Stateful, Growing Signatures. Idea: Signature ***Chains***

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



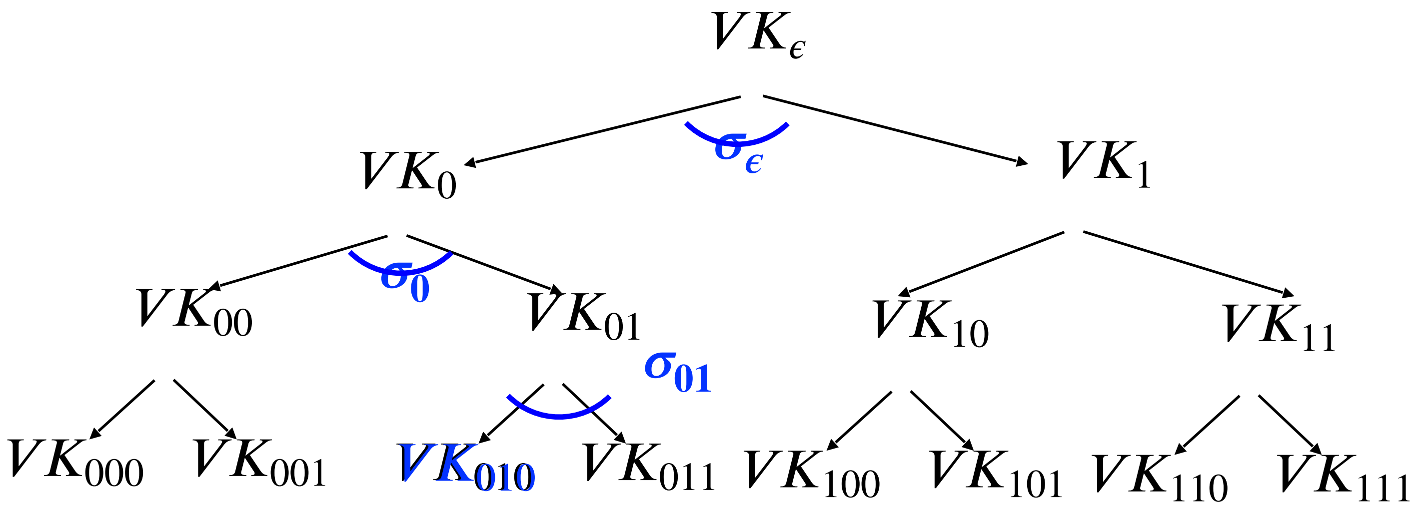
Signature of a message m:

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

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

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

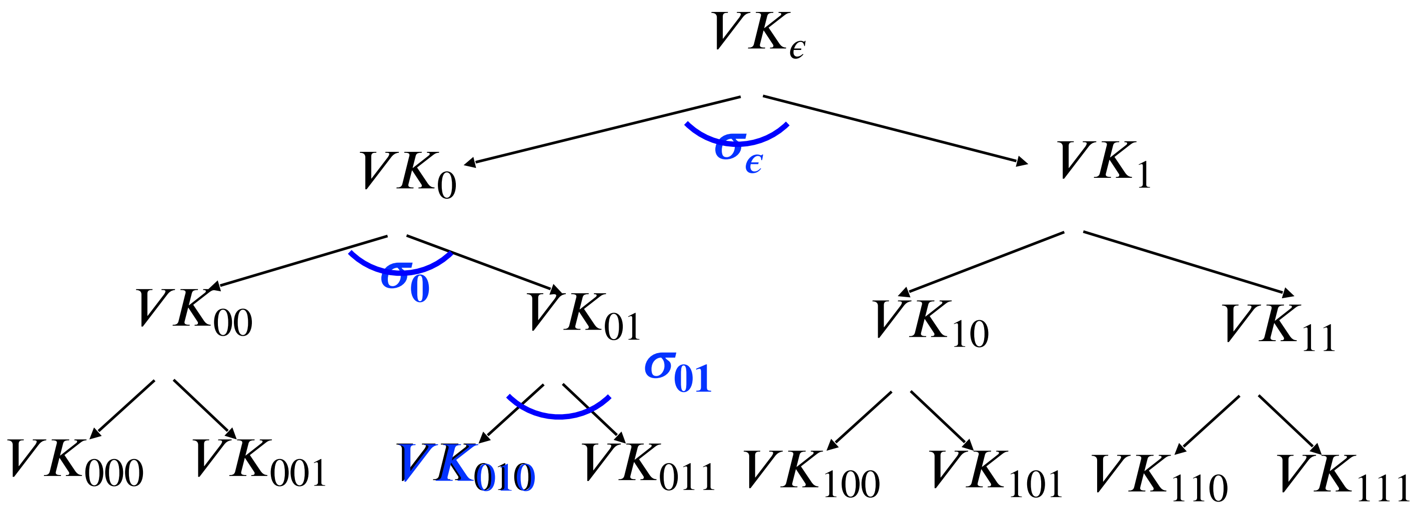
Step 4. Statelessness via Randomization



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***

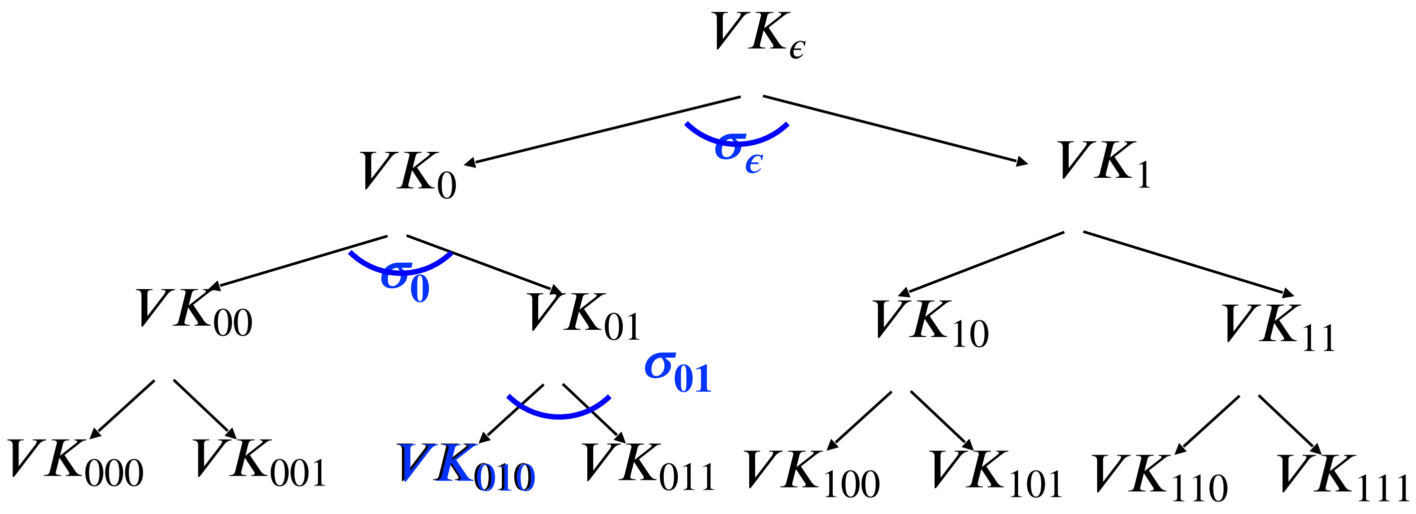
Step 2. How to Shrink the signatures. Idea: Signature ***Trees***

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 $r = PRF(K', m)$

That’s it for the construction.