Lecture 10
Signature Schemes

Stefan Dziembowski
www.crypto.edu.pl/Dziembowski

University of Warsaw

5.12.18 **version 1.1**

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, “hash-and-sign”, “full-domain-hash”
3. Constructions based on discrete log
   a) identification schemes
   b) Schnorr signatures
   c) DSA signatures
4. Theoretical constructions

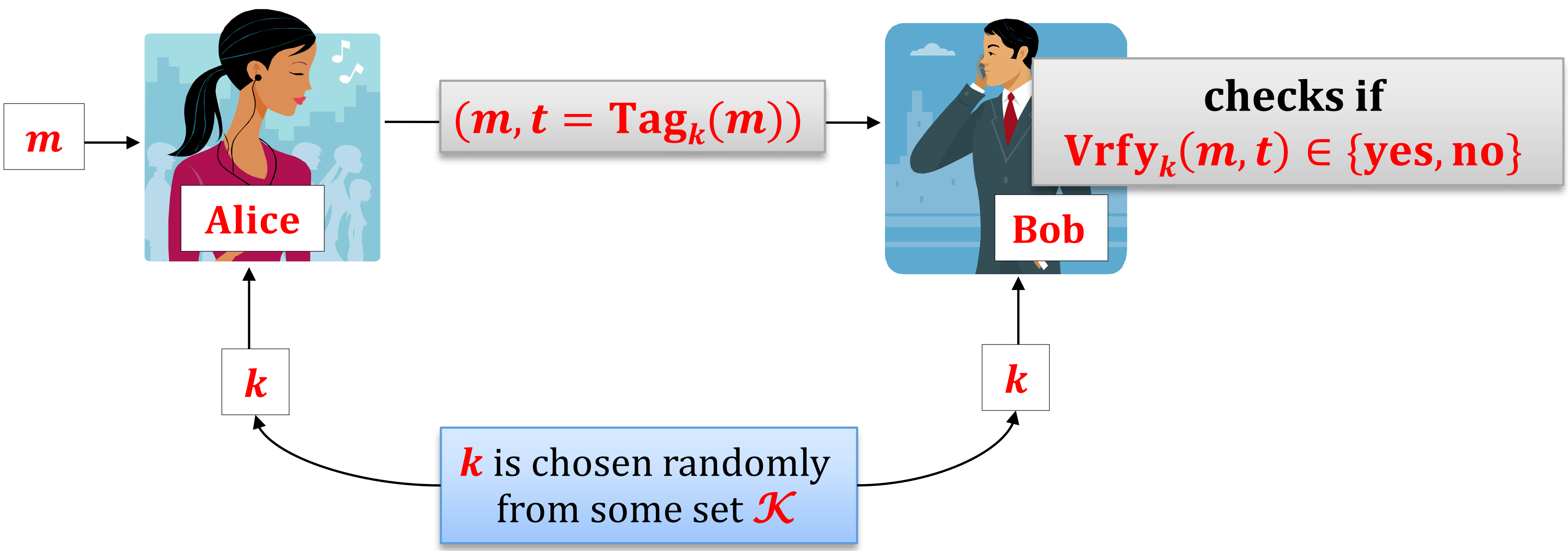
Signature schemes

digital signature schemes

$\approx$

MACs in the public-key setting

Message Authentication Codes

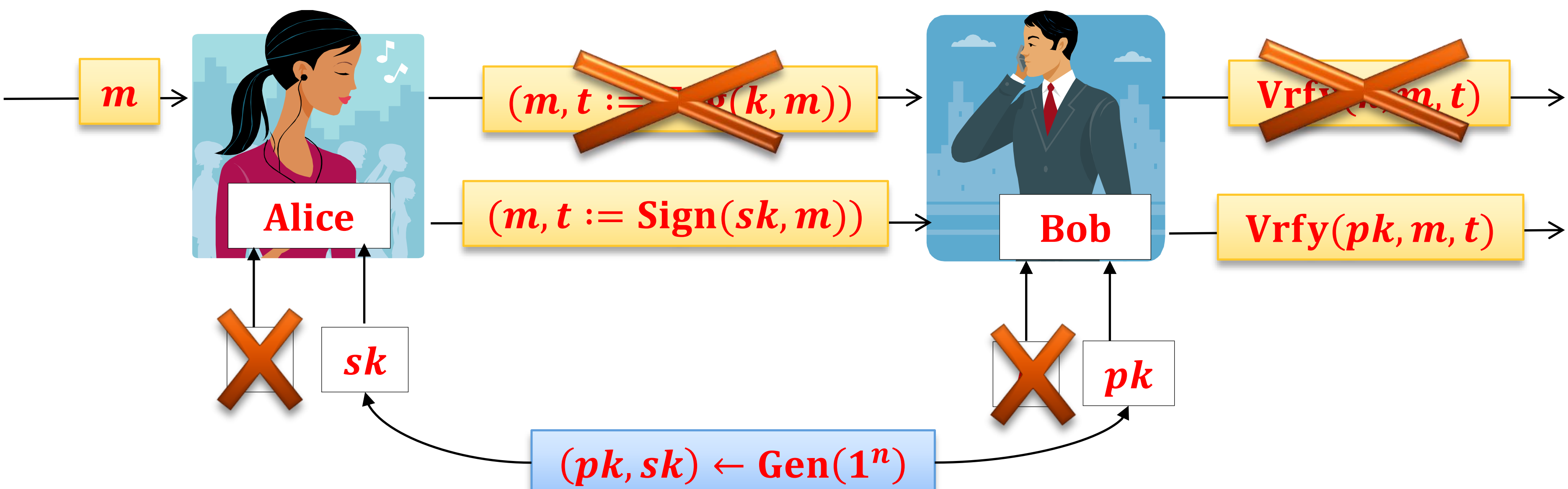


Signatures

- ***sk*** is used for **computing a tag**,
- ***pk*** is used for **verifying correctness of the tag**.

this will be called "**signatures**"

Sign – the signing algorithm



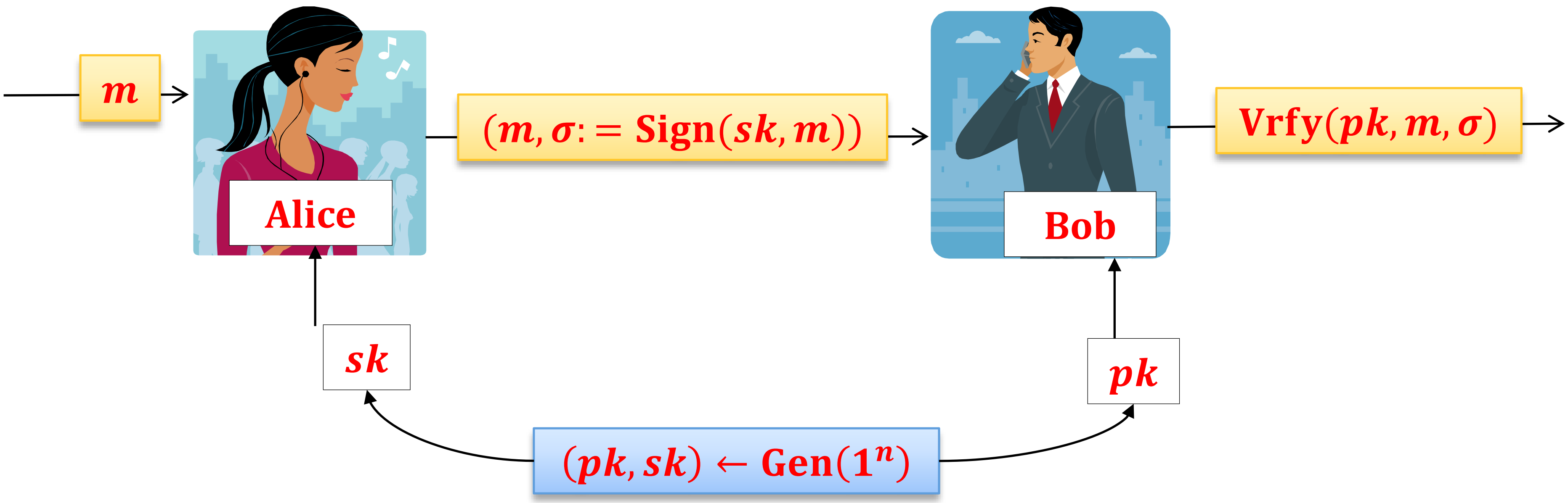
Advantages of the signature schemes

Digital signatures are:

1. **publicly verifiable**,
2. **transferable**, and
3. provide **non-repudiation**

(we explained it on **Lecture 7**, we now present the formal definition)

Signature Schemes



Digital Signature Schemes

A **digital signature scheme** is a tuple $(\mathbf{Gen}, \mathbf{Sign}, \mathbf{Vrfy})$ of poly-time algorithms, such that:

- the **key-generation** algorithm **Gen** takes as input a security parameter $\mathbf{1}^n$ and outputs a pair $(\boldsymbol{pk}, \boldsymbol{sk})$,
- the **signing** algorithm **Sign** takes as input a key $\boldsymbol{sk}$ and a message $\boldsymbol{m} \in \{\mathbf{0}, \mathbf{1}\}^*$ and outputs a signature $\boldsymbol{\sigma}$,
- the **verification** algorithm **Vrfy** takes as input a key $\boldsymbol{pk}$, a message $\boldsymbol{m}$ and a signature $\boldsymbol{\sigma}$, and outputs a bit $\boldsymbol{b} \in \{\mathbf{yes}, \mathbf{no}\}$.

If $\mathbf{Vrfy}_{pk}(\boldsymbol{m}, \boldsymbol{\sigma}) = \mathbf{yes}$ then we say that $\boldsymbol{\sigma}$ is a **valid signature on the message** $\boldsymbol{m}$.

Correctness

We require that it always holds that:

$$P\left(\mathbf{Vrfy}_{pk}(m, \mathbf{Sign}_{sk}(m)) \neq yes\right) \textbf{ is negligible in } n$$

What remains is to define **security**.

How to define security?

We have to assume that the adversary can see some pairs

$$(m_1, \sigma_1), \ldots, (m_t, \sigma_t)$$

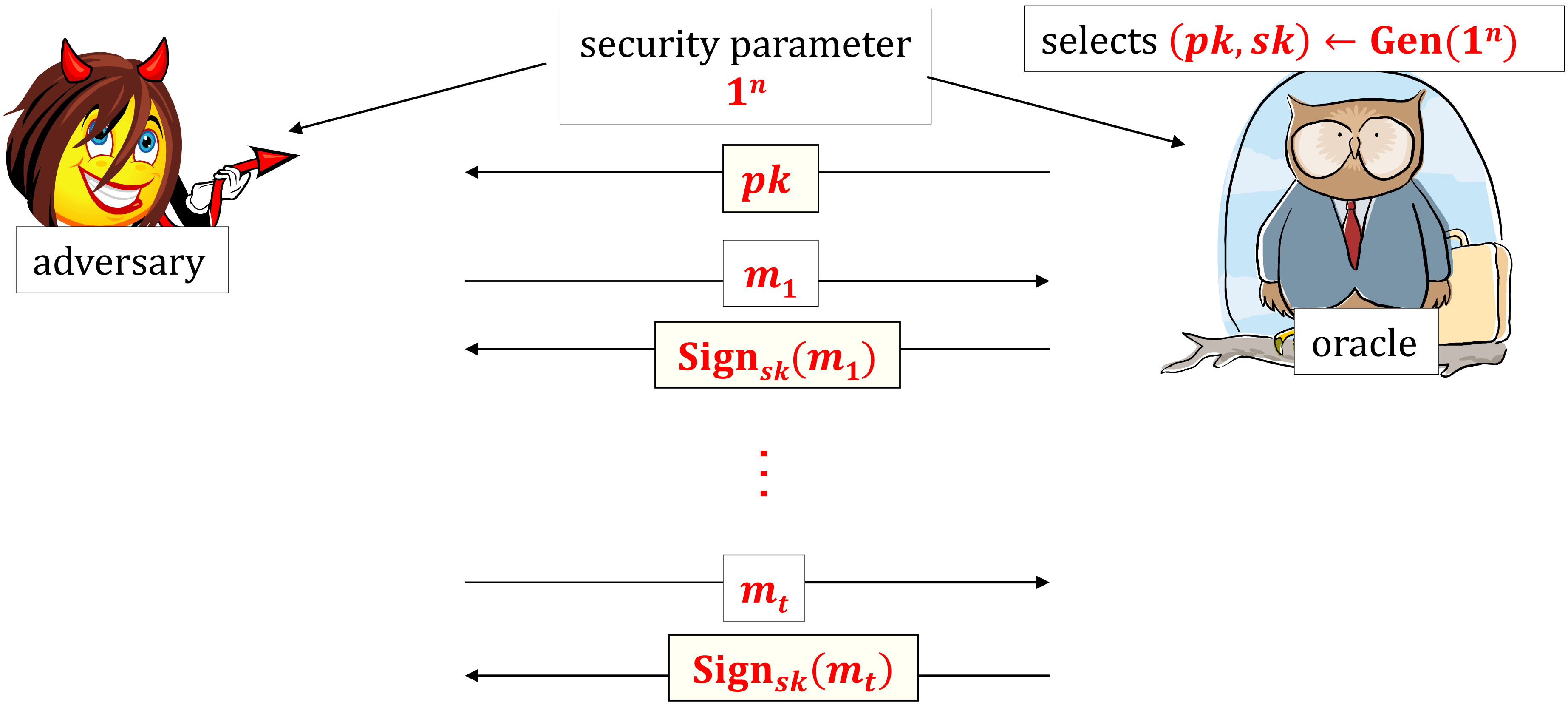
As in the case of MACs, we need to specify:

1. how the messages $m_1, \ldots, m_t$ are chosen,
2. what is the goal of the adversary.

Good tradition: be as pessimistic as possible!

Therefore we assume that:

1. The adversary is allowed to chose $m_1, \ldots, m_t$.
2. The **goal of the adversary** is to produce a valid signature on some m' such that $m' \notin \{m_1, \ldots, m_t\}$.



We say that the adversary **breaks the signature scheme** if at the end she outputs (m', σ') such that

1. $\mathbf{Vrfy}(m', \sigma') = \mathbf{yes}$
2. $m' \notin \{m_1, \ldots, m_t\}$.

The security definition

sometimes we just say: **unforgeable** (if the context is clear)

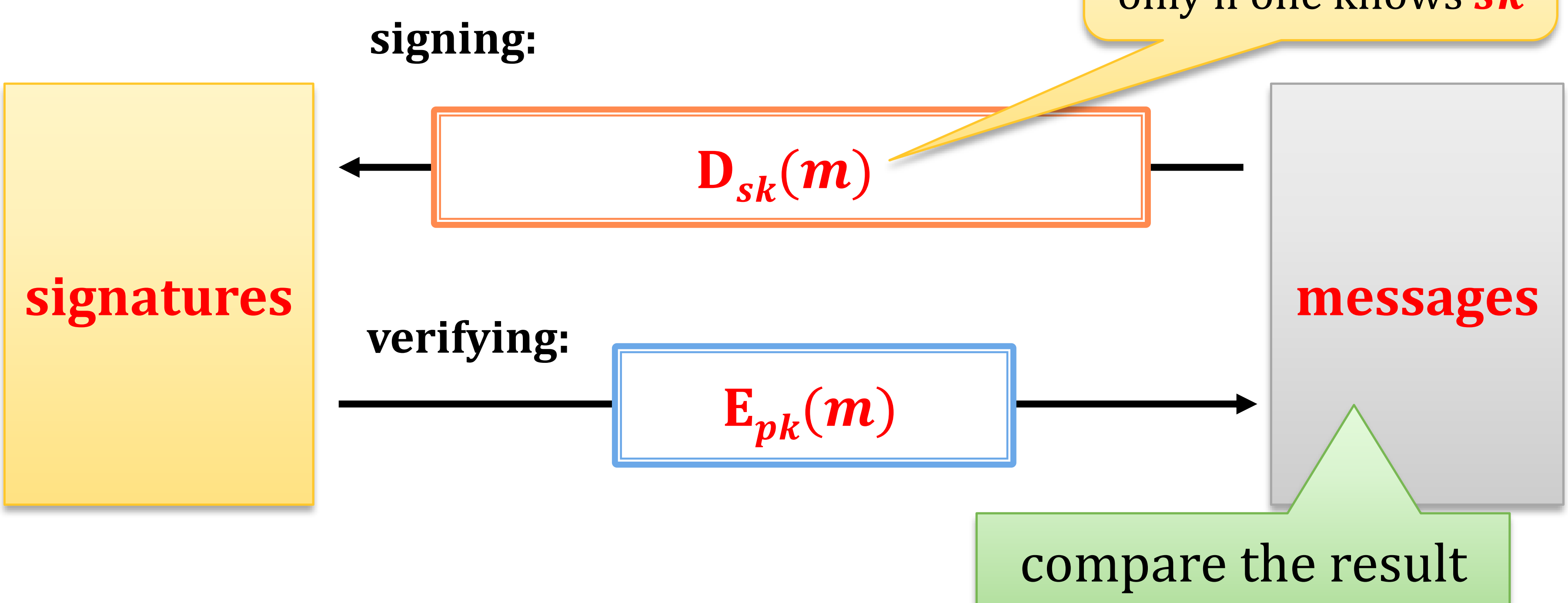
We say that **(Gen, Sign, Vrfy)** is **existentially unforgeable under an adaptive chosen-message attack** if

$\forall$ polynomial-time adversary A $P(A$ **breaks it**$)$ is negligible (in n)

How to construct signature schemes?

Remember this idea?

$\{\mathbf{E}_{pk} : X \to X\}_{pk \in \mathbf{keys}}$ - a family of trapdoor permutations indexed by $\boldsymbol{pk}$



We said: In general it's not that simple.

In general it's not that simple

Not every trapdoor permutation is OK.

example: the **RSA** function

There exist **other ways** to create signature schemes.

One can even construct a signature scheme **from any one-way function**.
(this is a theoretical construction)

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, “hash-and-sign”, “full-domain-hash”
3. Constructions based on discrete log
   a) identification schemes
   b) Schnorr signatures
   c) DSA signatures
4. Theoretical constructions

The “handbook RSA signatures”

$N = pq$, such that p and q are random primes, and $|p| = |q|$

e – random such that $e \perp (p-1)(q-1)$
d – such that $ed = 1 \ (\mathbf{mod}\ (p-1)(q-1))$

messages and **signatures**: Z_N

- $\sigma := \mathbf{Sign}_{N,d}(m) = m^d \ \mathbf{mod}\ N$
- $\mathbf{Vrfy}_{N,e}(m, \sigma) =$ **output yes iff** $\sigma^e \mathbf{mod}\ N = m$

Problems with the “handbook RSA” [1/2]

“no-message attack”:

The adversary can forge a signature on a “random” message m.

Given the public key (N, e):

he just selects a random $\sigma \leftarrow Z_N$ and computes

$$m := \sigma^e \bmod N.$$

Trivially, σ is a valid signature on m.

Problems with the "handbook RSA" [2/2]

How to forge a signature on an arbitrary message $\boldsymbol{m}$?

Use the homomorphic properties of RSA.

adversary

oracle

Oracle → adversary: (N, e)

Adversary → oracle: m_1

Oracle → adversary: $\mathbf{Sign}_{sk}(m_1) = m_1^d \bmod N$

Adversary → oracle: m_2

Oracle → adversary: $\mathbf{Sign}_{sk}(m_2) = m_2^d \bmod N$

chooses:

1. any $m_1 \neq 1$
2. $m_2 := m/m_1 \bmod N$

computes ($\bmod N$):

$$
\begin{aligned}
& m_1^d \cdot m_2^d \\
= \; & (m_1 \cdot m_2)^d \\
= \; & m^d
\end{aligned}
$$

this is a valid signature on $\boldsymbol{m}$

Is it a problem?

In many applications – probably not.

But we would like to have schemes that are **not** **application-dependent**...

Solution

Before computing the **RSA function** – apply some function $\boldsymbol{H}$.

$\boldsymbol{N = pq}$, such that $\boldsymbol{p}$ and $\boldsymbol{q}$ are random primes, and $|\boldsymbol{p}| = |\boldsymbol{q}|$

$\boldsymbol{e}$ – random such that $\boldsymbol{e \perp (p-1)(q-1)}$
$\boldsymbol{d}$ – such that $\boldsymbol{ed = 1 \ (\mathrm{mod}\ (p-1)(q-1))}$

messages and **signatures**: $\boldsymbol{Z_N}$

- $\boldsymbol{\sigma := \mathrm{Sign}_{N,d}(m) = \left(H(m)\right)^d \bmod N}$
- $\boldsymbol{\mathrm{Vrfy}_{N,e}(m, \sigma)}$ = **output yes iff** $\boldsymbol{\sigma^e \bmod N = H(m)}$

How to choose such $\boldsymbol{H}$?

A minimal requirement:

it should be collision-resistant.

(because if the adversary can find two messages $\boldsymbol{m}, \boldsymbol{m}'$
such that
$$\boldsymbol{H}(\boldsymbol{m}) = \boldsymbol{H}(\boldsymbol{m}')$$
then he can forge a signature on $\boldsymbol{m}'$ by asking the oracle
for a signature on $\boldsymbol{m}$)

A typical choice of H

Usually H is one of the popular **hash functions**.

Additional advantage:

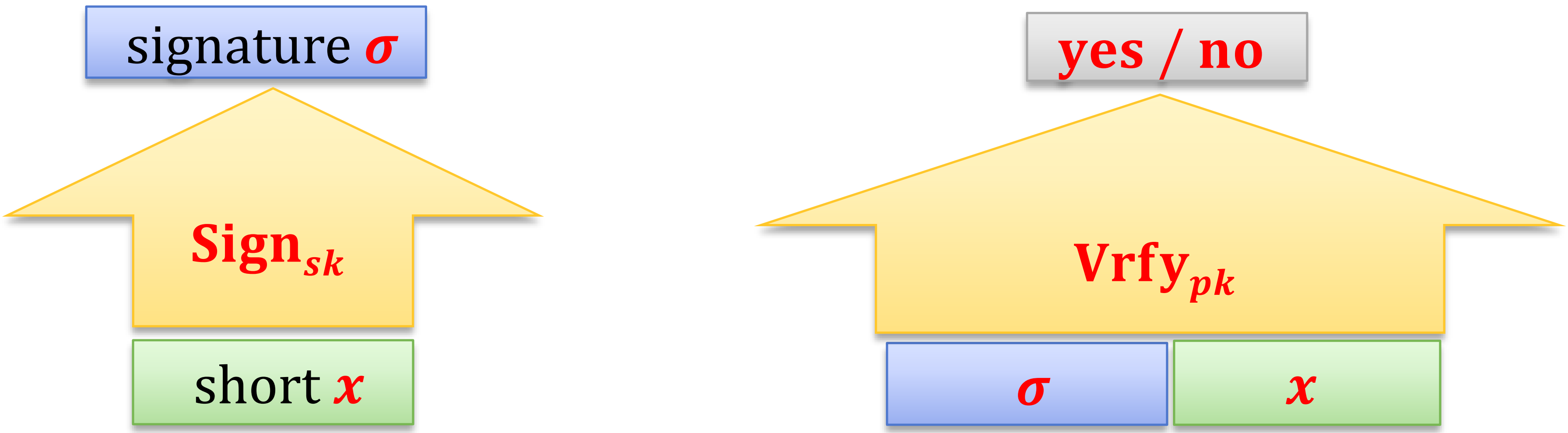
We can sign **very long messages** keeping the modulus N small (it's much more efficient!) – we will come back to it later.

It is called a

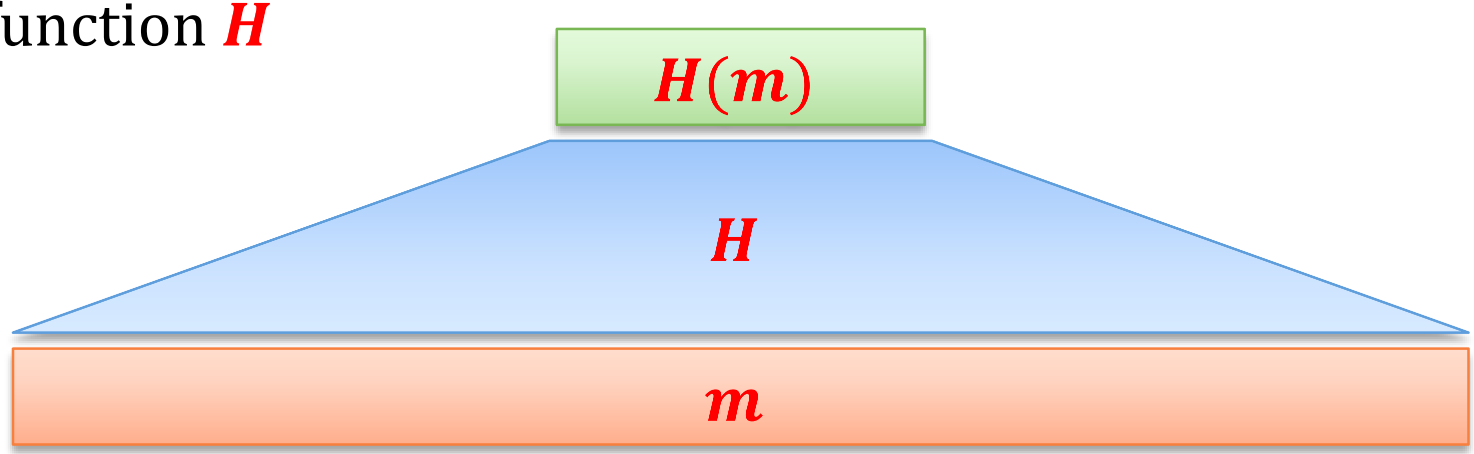
hash-and-sign paradigm.

Hash-and-Sign [1/4]

1. **(Gen, Sign, Vrfy)** – a signature scheme "for short messages"

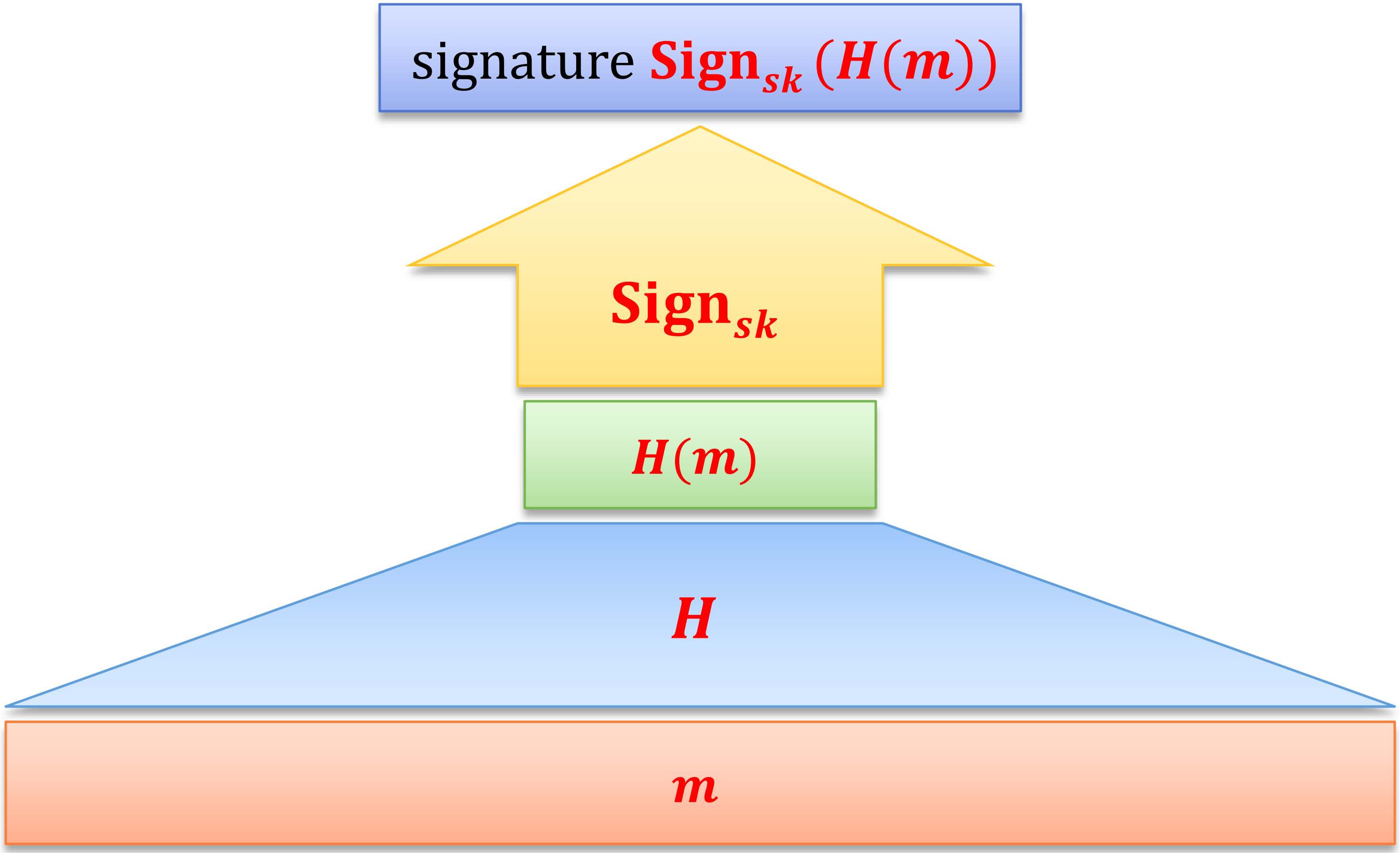


2. a hash function H



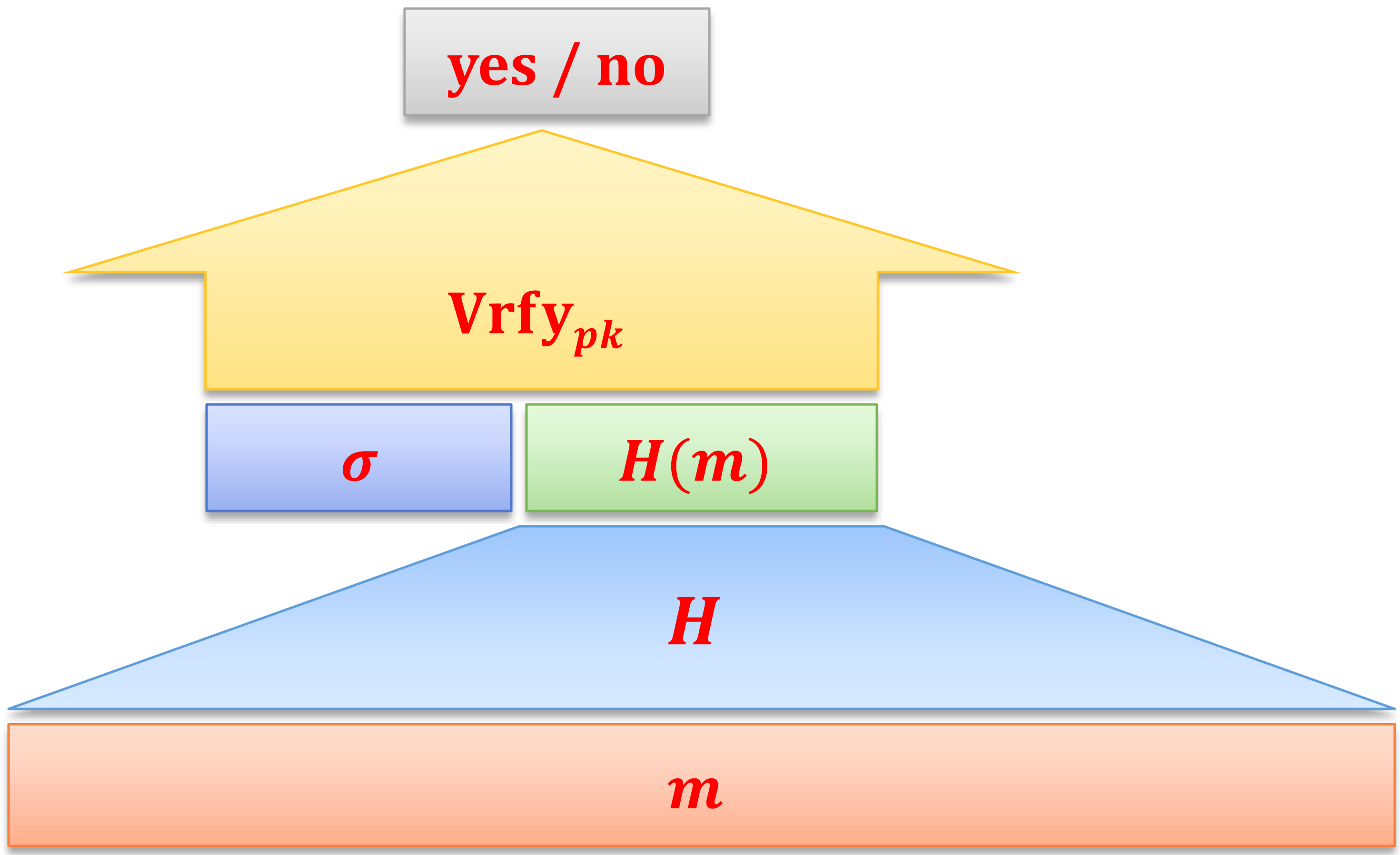
Hash-and-Sign [2/4]

How to sign a message $\boldsymbol{m}$?



Hash-and-Sign [3/4]

How to verify?



Hash-and-Sign [4/4]

It can be proven that this construction is secure.

For this we need to assume that $\boldsymbol{H}$ is taken from a family of collision-resilient hash functions.

$$\{H^s\}_{s \in \mathbf{keys}}$$

Then $\boldsymbol{s}$ becomes a part of the public key and the private key.

Can anything be proven about the "hashed RSA" scheme?

In the plain model - not really.

But at least the attacks described before "look infeasible".

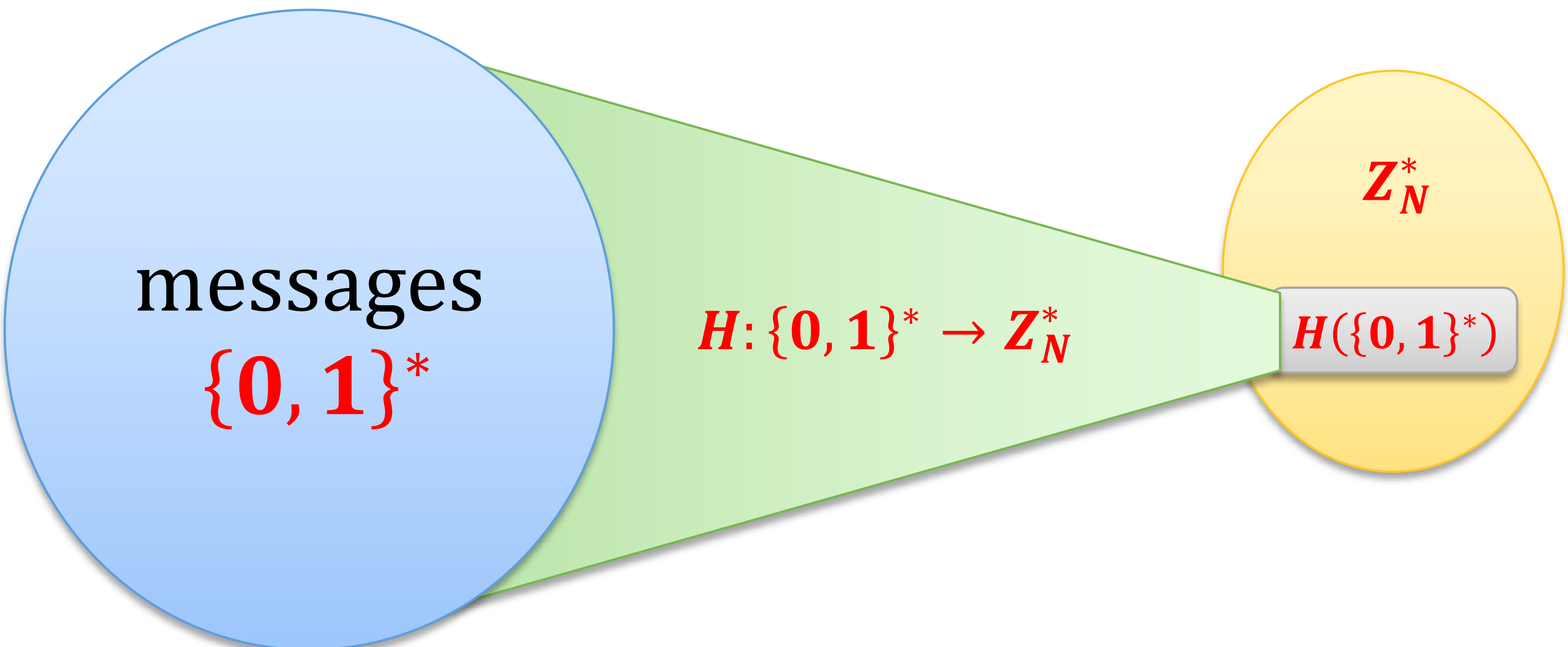
1. **For the "no message attack"**: one would need to invert $\boldsymbol{H}$.

2. **The second ("homomorphic") attack**:
   Looks impossible because the adversary would need to find messages $\boldsymbol{m}, \boldsymbol{m_1}, \boldsymbol{m_2}$ such that

$$\boldsymbol{H}(\boldsymbol{m}) = \boldsymbol{H}(\boldsymbol{m_1}) \cdot \boldsymbol{H}(\boldsymbol{m_2})$$

Why the security proof from the RSA assumption is impossible?

RSA assumption holds for inputs chosen **uniformly at random** from Z_N^*.
But the output of H is **not** "uniformly random"



Solution: **“Full Domain Hash” (FDH)**

provably secure:

- under the **RSA assumption**
- and modelling ***H*** as random oracle.

Introduced in

> **Bellare and Rogaway. *The exact security of digital signatures: How to sign with RSA and Rabin*. EUROCRYPT’96**

Widely used in practice (for example in the **PKCS #1 standard**)

Fact (security of the **Full Domain Hash**)

Lemma (informal)

- Let $\boldsymbol{H}: \{\mathbf{0}, \mathbf{1}\}^* \to \boldsymbol{Z}_N^*$ be a hash function modeled as a **random oracle.**
- Suppose the **RSA assumption** holds

Then the "**hashed RSA**" is **existentially unforgeable under an adaptive chosen-message attack**.

Remember the Random Oracle Model?

x

$H(x)$

$H : \{0,1\}^* \rightarrow \{0,1\}^L$

a completely random function

Why does it help?

RSA assumption

For any randomized polynomial time algorithm A we have:

$$P\left(y^e = x \bmod N : \; y := A(x, N, e)\right)$$

is negligible in $|N|$

where $N = pq$ where p and q are random primes such that $|p| = |q|$, and x is a random element of Z_N^*,
and e is a random element of $Z_{\varphi(N)}^*$.

here we require that x is random

Intuition

If we just use a "normal hash function" then the distribution of

$$H(m_0), H(m_1), H(m_2), \ldots$$

(for any $m_0, m_1, m_2, \ldots$) can be "arbitrary".

If H is a random oracle then

$$H(m_0), H(m_1), H(m_2), \ldots$$

are uniform and independent (for pairwise different m_i's).

This helps a lot in the proof!

Other popular signature schemes

- **Rabin** signatures (based on squaring modulo $\boldsymbol{N = pq}$)

Based on discrete log (usually: in subgroups of $\boldsymbol{Z_N^*}$ or in elliptic curves groups):

- **ElGamal** signatures
- Digital Signature Standard (**DSS**)
- **Schnorr** signatures

can be viewed as **identification schemes** transformed using **Fiat-Shamir transform**.

we will explain it

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, “hash-and-sign”, “full-domain-hash”
3. Constructions based on discrete log
   a) identification schemes
   b) Schnorr signatures
   c) DSA signatures
4. Theoretical constructions

Identification schemes

$(pk, sk) \leftarrow \mathbf{Gen}(1^n)$ – a **(public key, private key)** pair of **prover**

Everybody who knows pk can verify the identity of the **prover**

sk

prover

pk

verifier

verifier's output:
- **yes** if he believes he is talking to the prover
- **no** – otherwise

Definition

We do not define identification schemes formally.

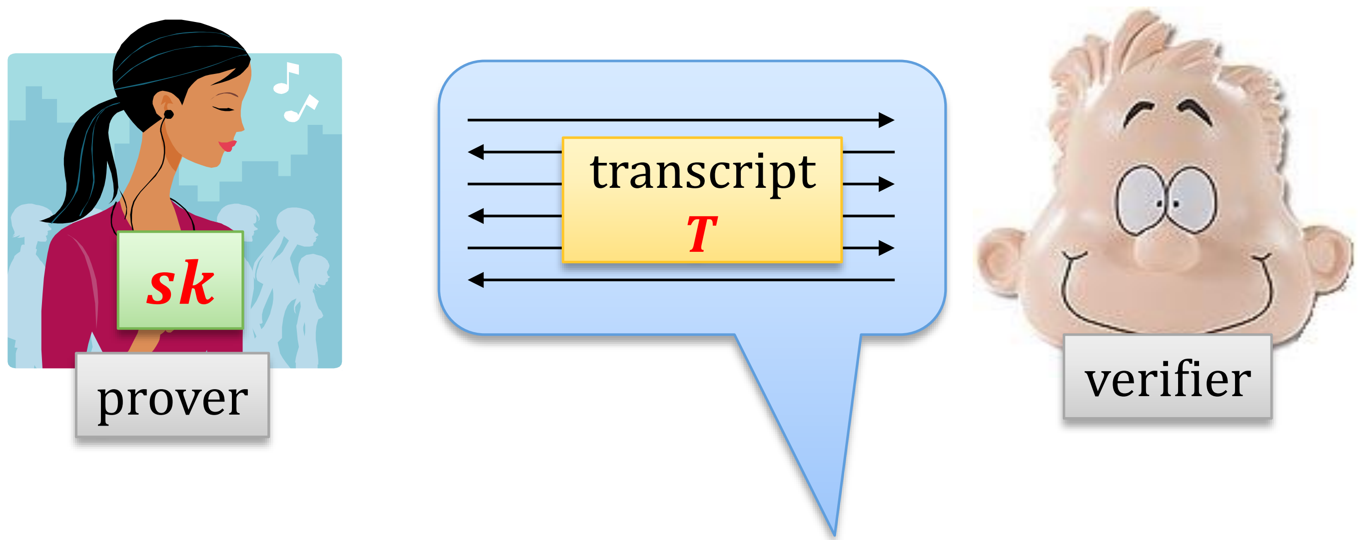
Informally they have to satisfy the following:

- **[correctness]** an honest prover should always convince the verifier
- **[security]** no poly-time adversary should be able to **impersonate the prover** with non-negligible probability.

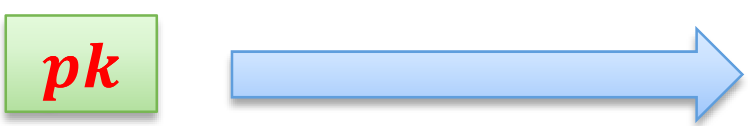
What is the attack model?

Let's assume it's rather weak:

"learning phase":



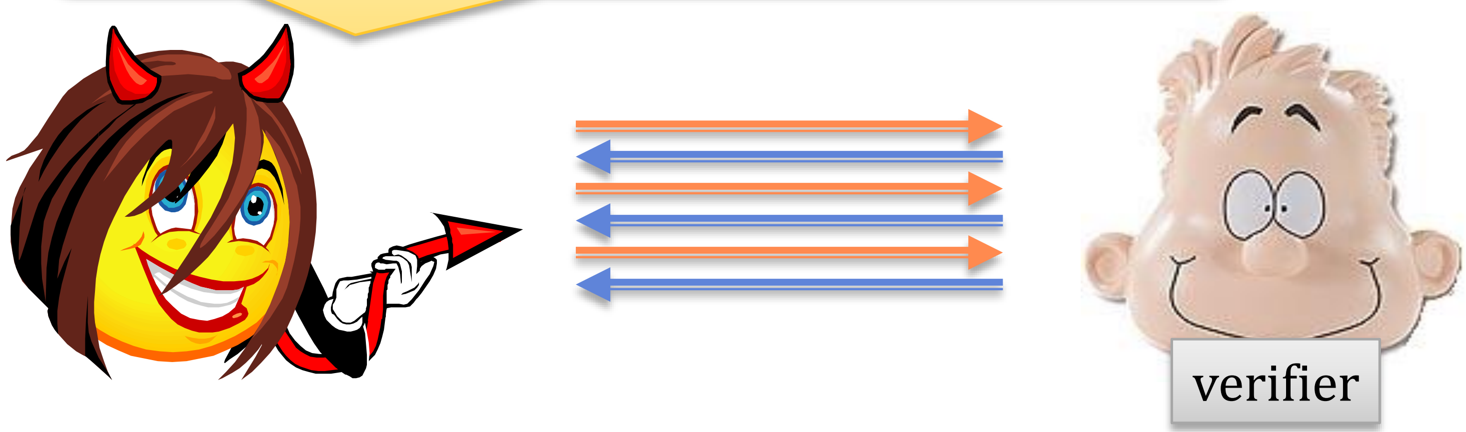
the adversary learns as many transcripts $T_1, T_2, \ldots$ as he wants



Then he has to win in the following game

"challenge phase":

the adversary does not know ***sk*** but can produce any messages that he wants



the adversary wins if the verifier outputs **yes**

Note

1. The adversary **cannot talk to the prover during the learning phase**.

2. The adversary **cannot act as a man-in-the middle**.

(these problems can be solved, but are not relevant today)

We will come back again to these protocols when we talk about **zero-knowledge**.

Schnorr identification scheme

Key generation similar to the one in ElGamal encryption

Let **GenG** be such that **discrete log** is hard w. r. t. **GenG**.

$\mathbf{Gen}(1^n)$ first runs **GenG** to obtain G, g and q (assume q is prime). Then, it chooses $x \leftarrow Z_q$ and computes $y := g^x$.

The public key is (G, g, q, y).
The private key is (G, g, q, x).

The protocol

G – group, $q = |G|$
g – generator

knows x

knows
$y := g^x$

$k \leftarrow Z_q$
$I := g^k$

I →

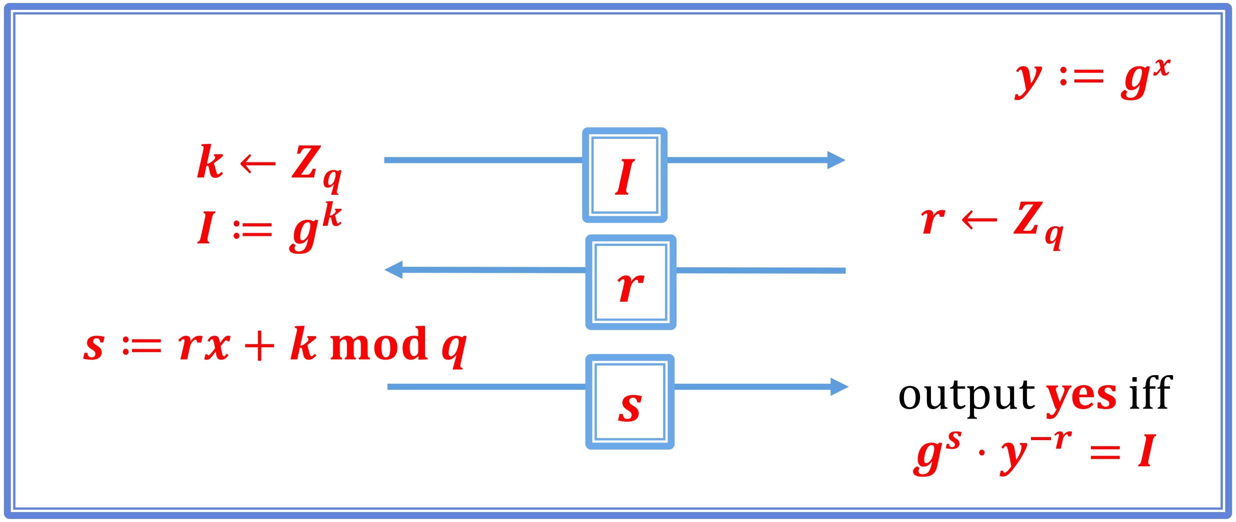
$r \leftarrow Z_q$

← r

$s := rx + k \bmod q$

s →

output **yes** iff
$g^s \cdot y^{-r} = I$



Why is this protocol correct?

$$\begin{aligned} g^s \cdot y^{-r} &= g^{rx+k} \cdot (g^x)^{-r} \\ &= g^{rx+k} \cdot g^{-rx} \\ &= g^k \\ &= I \end{aligned}$$

Security

First, suppose the adversary didn't see any transcript. He has to win the following game.

I →

$r \leftarrow Z_q$

← r

s →

output **yes** iff $g^s \cdot y^{-r} = I$

Lemma

If discrete logarithm is hard with respect to **GenG** then the **probability that any poly-time adversary wins this game is negligible**.

How to prove it?

We show that for every $\boldsymbol{I}$ there exists **at most one** $\boldsymbol{r} \in \boldsymbol{Z_q}$ such that **the adversary can answer it correctly** (if he cannot compute the discrete log).

(so: his probability of winning is at most $\mathbf{1}/\boldsymbol{q}$)

Proof by contradiction

Assume there exist r_0 and r_1 such that $r_0 \neq r_1$ and that the adversary knows answers

- s_0 to r_0 and
- s_1 to r_1

where

$$g^{s_0} \cdot y^{-r_0} = I = g^{s_1} \cdot y^{-r_1}.$$

But then

$$y^{r_1 - r_0} = g^{s_1 - s_0}$$

so

$$y = g^{\frac{s_1 - s_0}{r_1 - r_0}}$$

$= \log_g y$

This finishes the proof of the lemma.

To finish the full security proof we need to show the following

Learning the transcripts

$$(\boldsymbol{I}, \boldsymbol{r}, \boldsymbol{s})$$

doesn't help the adversary.

Q: Why is it true?

A: It turns out that the adversary can "simulate" such transcripts himself (just from $\boldsymbol{pk}$).

We now explain it.

How do the transcripts look like?

$$(\boldsymbol{I}, \boldsymbol{r}, \boldsymbol{s})$$

where

- $\boldsymbol{I} = \boldsymbol{g}^{\boldsymbol{k}}$ where $\boldsymbol{k} \leftarrow \boldsymbol{Z}_{\boldsymbol{q}}$
- $\boldsymbol{r} \leftarrow \boldsymbol{Z}_{\boldsymbol{q}}$
- $\boldsymbol{s} \coloneqq \boldsymbol{rx} + \boldsymbol{k} \bmod \boldsymbol{q}$

We now show that:

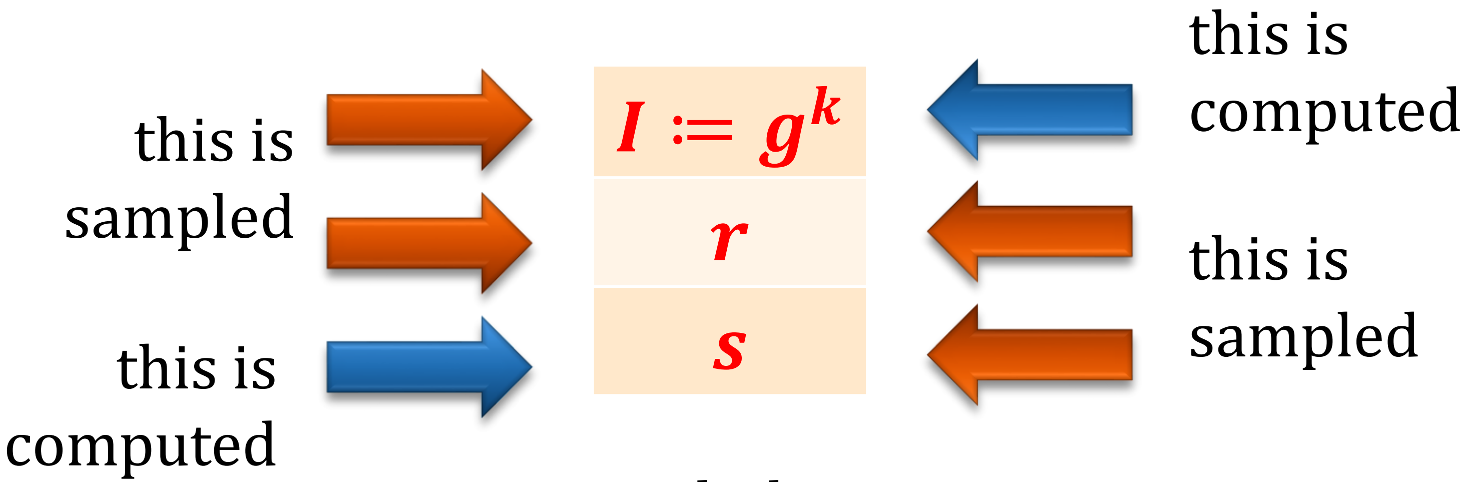
the transcripts with exactly the same distribution can be sampled by the adversary himself!

How can the adversary do it?

- **first** sample $r, s \leftarrow Z_q$ and
- **then** compute I as

$$I = g^s \cdot y^{-r}$$

Why is the distribution the same?

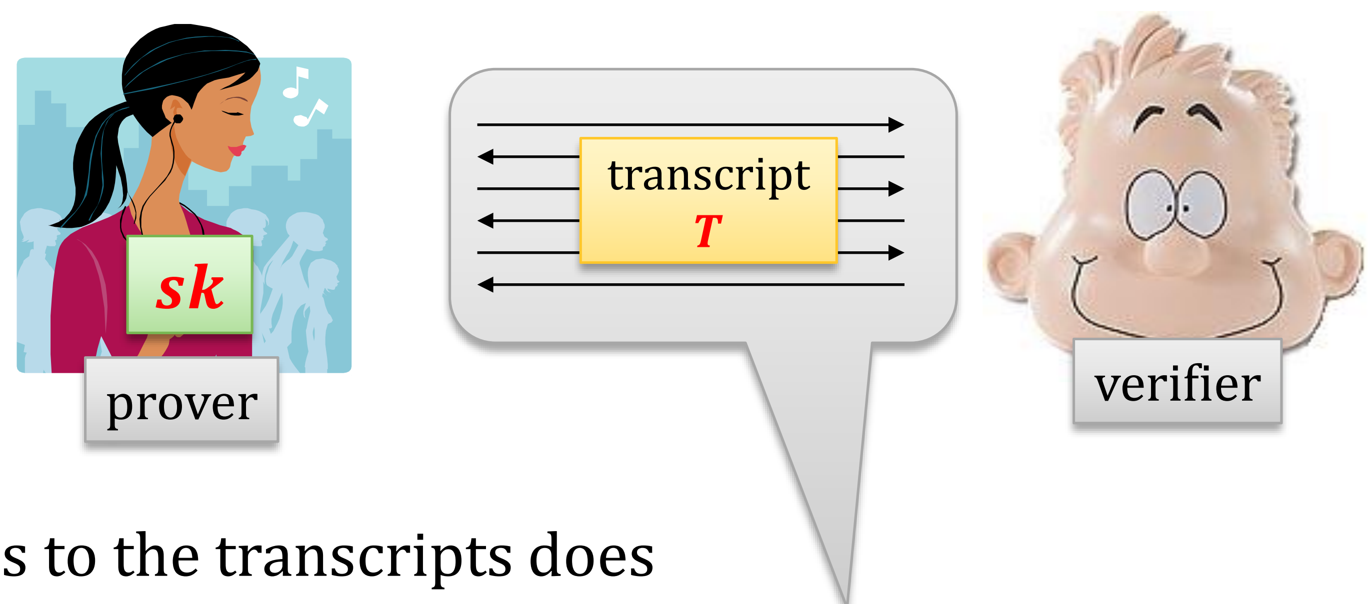


such that

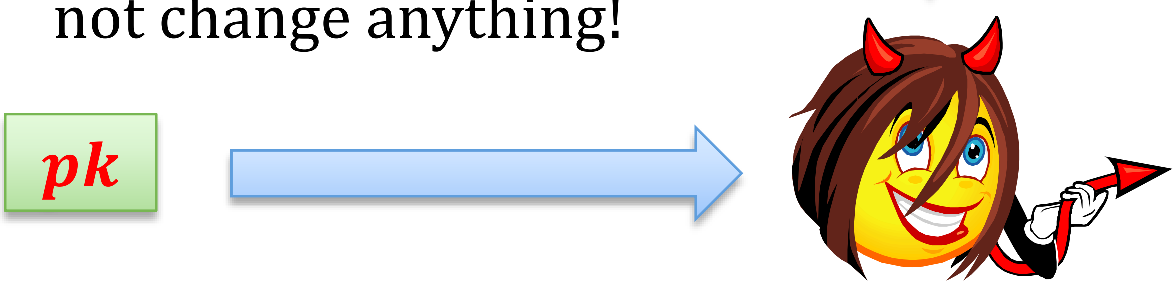
$$s := rx + k \bmod q$$

It’s the same!

Therefore



access to the transcripts does not change anything!



Note the difference

The adversary **can** produce tuples $(\boldsymbol{I}, \boldsymbol{r}, \boldsymbol{s})$ with the right distribution

if he``**starts from** $(\boldsymbol{r}, \boldsymbol{s})$"

but he **cannot do it**

if he has to ``**start from** $\boldsymbol{I}$" and sampling $\boldsymbol{r}$ is out of his control.

Conclusion

The Schnorr protocol is a secure identification scheme.

But how is this related to the signature schemes?

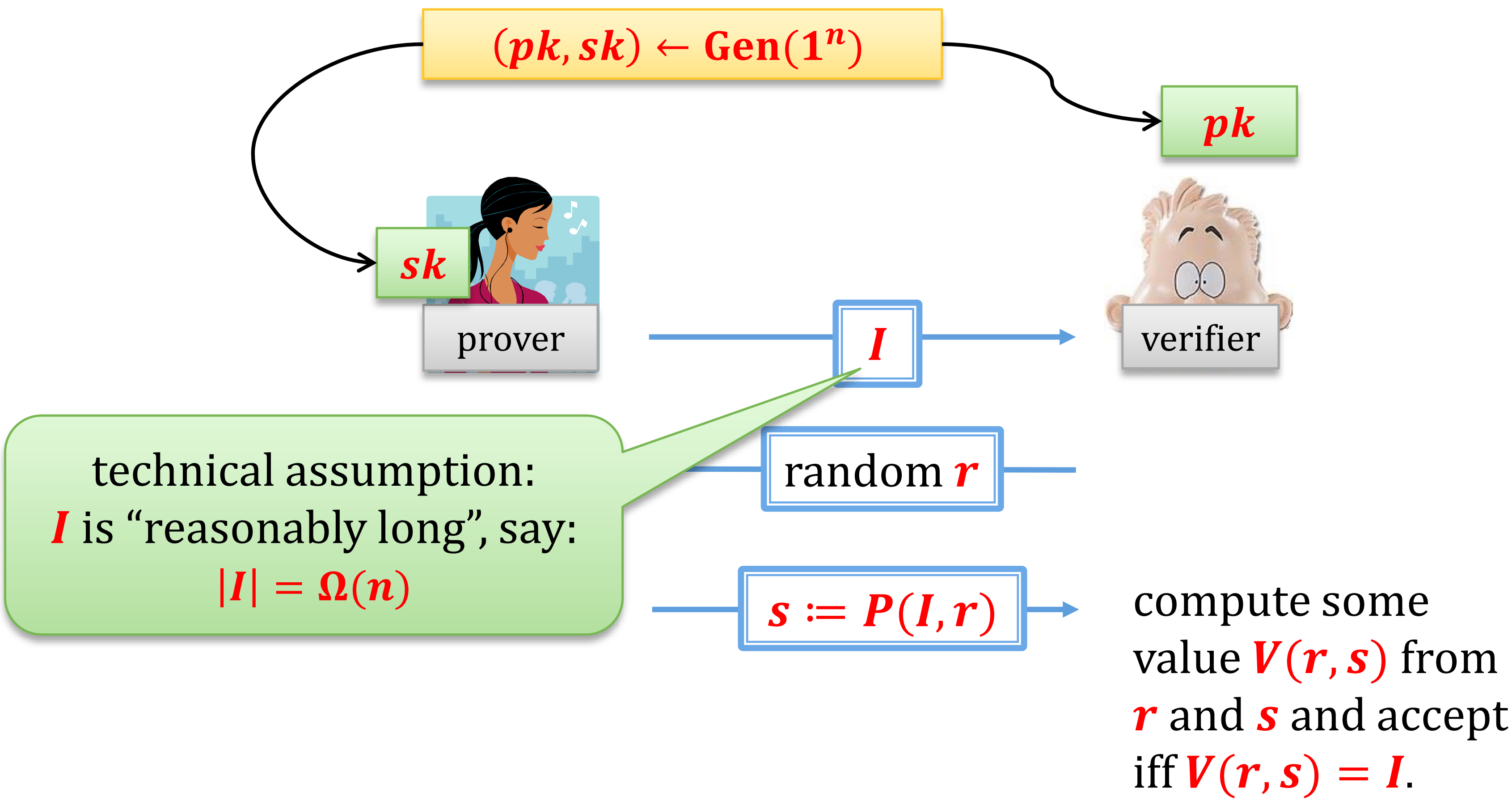
We now show how to transform **any such identification scheme into a signature scheme**.

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, "hash-and-sign", "full-domain-hash"
3. Constructions based on discrete log
    a) identification schemes
    b) Schnorr signatures
    c) DSA signatures
4. Theoretical constructions

Fiat-Shamir transform: main idea

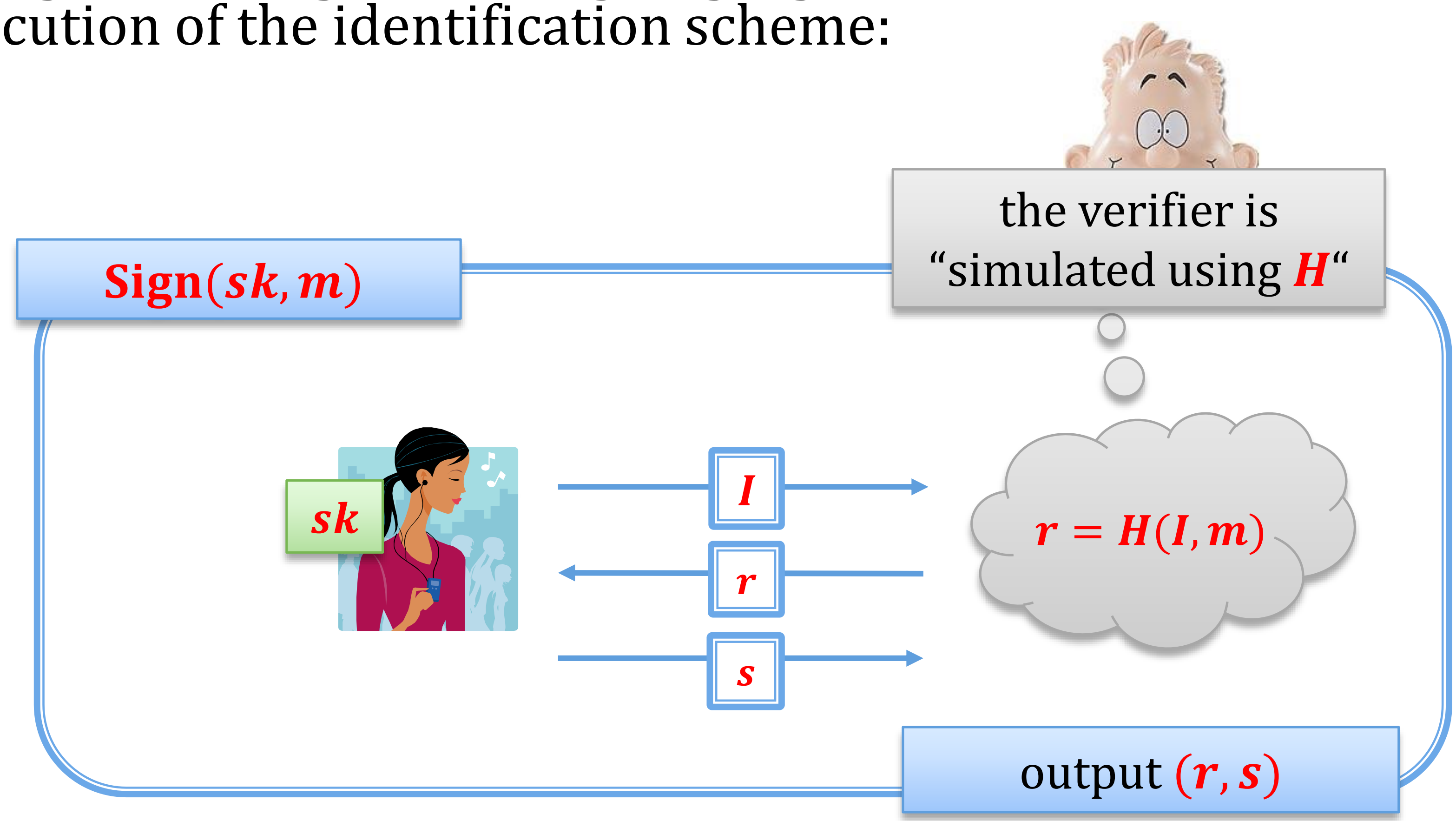
Suppose we have an identification protocol of this form:

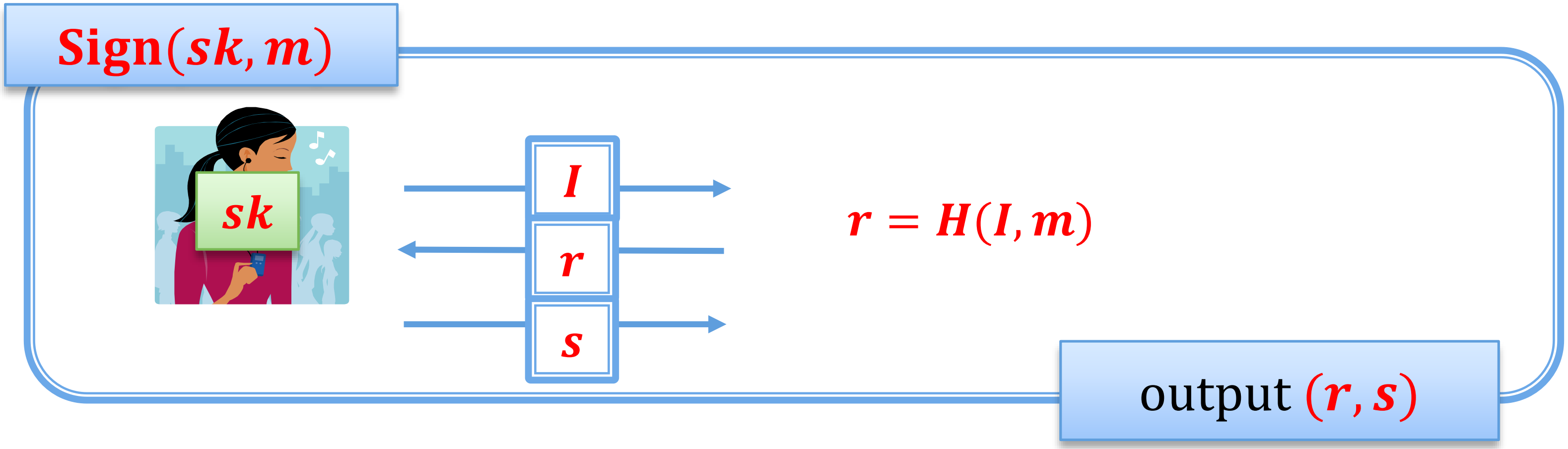


Create a signature scheme as follows

- Let $H: \{0,1\}^* \rightarrow \{0,1\}^{|r|}$ be a **hash function** (modelled as a **random oracle**)
- **key pair generation** – as in the authentication protocol

To sign a message m the signing algorithm simulates the execution of the identification scheme:





How to verify?

$\mathbf{Vrfy}(pk, m, (r, s))$

Assuming I is such that (I, r, s) "is a correct transcript" check if r was computed correctly. That is:

let $I = V(r, s)$
and check if $r = H(I, m)$

equivalent

check if
$r = H(V(r, s), m)$

output **yes** iff the prover outputs **yes**

More formally

Gen – the same as in the identification scheme

Sign$(sk, m) = (r, s)$, computed by simulating the prover or random input as follows:

1. let I be the "first message of the prover"
2. let $r := H(I, m)$
3. let $s := P(I, r)$ be the "second message of the prover" (after receiving r)
4. **output** (r, s)

Vrfy$(pk, m, (r, s))$:

output yes iff $r = H(V(r, s), m)$.

Why does it work?

Correctness is trivial – if the signer is honest then the verifier will always accept.

What about **security**?

Security

First look at the **learning phase**:

In the **identification scheme** the adversary learns $\boldsymbol{pk}$ and can see many tuples of a form:

$$(\boldsymbol{I}, \boldsymbol{r}, \boldsymbol{s})$$

sampled as follows:

- random $\boldsymbol{I}$
- random $\boldsymbol{r}$
- $\boldsymbol{s} := \boldsymbol{P}(\boldsymbol{I}, \boldsymbol{r})$

difference

In the **signature scheme** the adversary learns $\boldsymbol{pk}$ and can see many tuples of a form:

$$(\boldsymbol{I}, \boldsymbol{r}, \boldsymbol{s})$$

sampled as follows:

- random $\boldsymbol{I}$
- $\boldsymbol{r} := \boldsymbol{H}(\boldsymbol{I}, \boldsymbol{m})$
- $\boldsymbol{s} := \boldsymbol{P}(\boldsymbol{I}, \boldsymbol{r})$

Note:

the adversary chooses $\boldsymbol{m}$ but he **cannot choose** $\boldsymbol{I}$, so by the properties of the random oracle: $\boldsymbol{r}$ is **completely random**.

Moral:

these two experiments are identical!

Now look at the challenge phase

To break the **authentication scheme** the adversary has to find I such that
after learning random r
he can find s
such that:

$$V(r,s) = I$$

In the **signature scheme** the adversary has to find (r,s) such that

$$r = H(I,m),$$

where $I = V(r,s)$.

since H is a random oracle

He has to:
choose the value of I first,
then he learns $r = H(I,m)$,
and he has to find s such

$$V(r,s) = I$$

it's the same!

because if he chooses r first he will not be able to find the right I (remember that I is long)

Using this method we can construct signature schemes

For example:

Schnorr's signature scheme

Gen(1^n) run **GenG** to obtain G, g and q (assume q is prime). Then, choose $x \leftarrow Z_q$ and computes $y := g^x$.

- The public key pk is (G, g, q, y).
- The private key sk is (G, g, q, x).

Sign(sk, m):

1. choose uniform $k \leftarrow Z_q$ and let $I := g^k$
2. compute $r := H(I, m)$
3. compute $s := rx + k \bmod q$
4. output (r, s)

Vrfy$\left(pk, m, (r, s)\right)$:

output **yes** if $r = H(g^s \cdot y^{-r}, m)$.

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, “hash-and-sign”, “full-domain-hash”
3. Constructions based on discrete log
   a) identification schemes
   b) Schnorr signatures
   c) DSA signatures
4. Theoretical constructions

DSS signatures (also called DSA)

- based on a paradigm **similar to Schnorr's signatures**
- can also be viewed as a **variant of ElGamal signatures** (**1984**)
- **DSS** was covered by an (expired) **U.S. Patent** 5,231,668 (**1991**) granted to the US government (available worldwide **royalty-free**)
- Schnorr claimed that his **U.S. Patent** 4,995,082 (**1989**) covered **DSA** – this claim is disputed, and anyway it expired in **2008**.
- very widely used in practice!

[we will present this scheme during the exercises]

Note

In **Schnorr** and **DSS signatures** it's **very important** that the signer's **randomness is generated properly** [**exercise**].

Failure to do so can have catastrophic effects:


BBC Sign in News Sport Weather Shop Earth Travel

NEWS

Home | Video | World | UK | Business | Tech | Science | Magazine | Entertainment & Arts

Technology

iPhone hacker publishes secret Sony PlayStation 3 key

By Jonathan Fildes
Technology reporter, BBC News

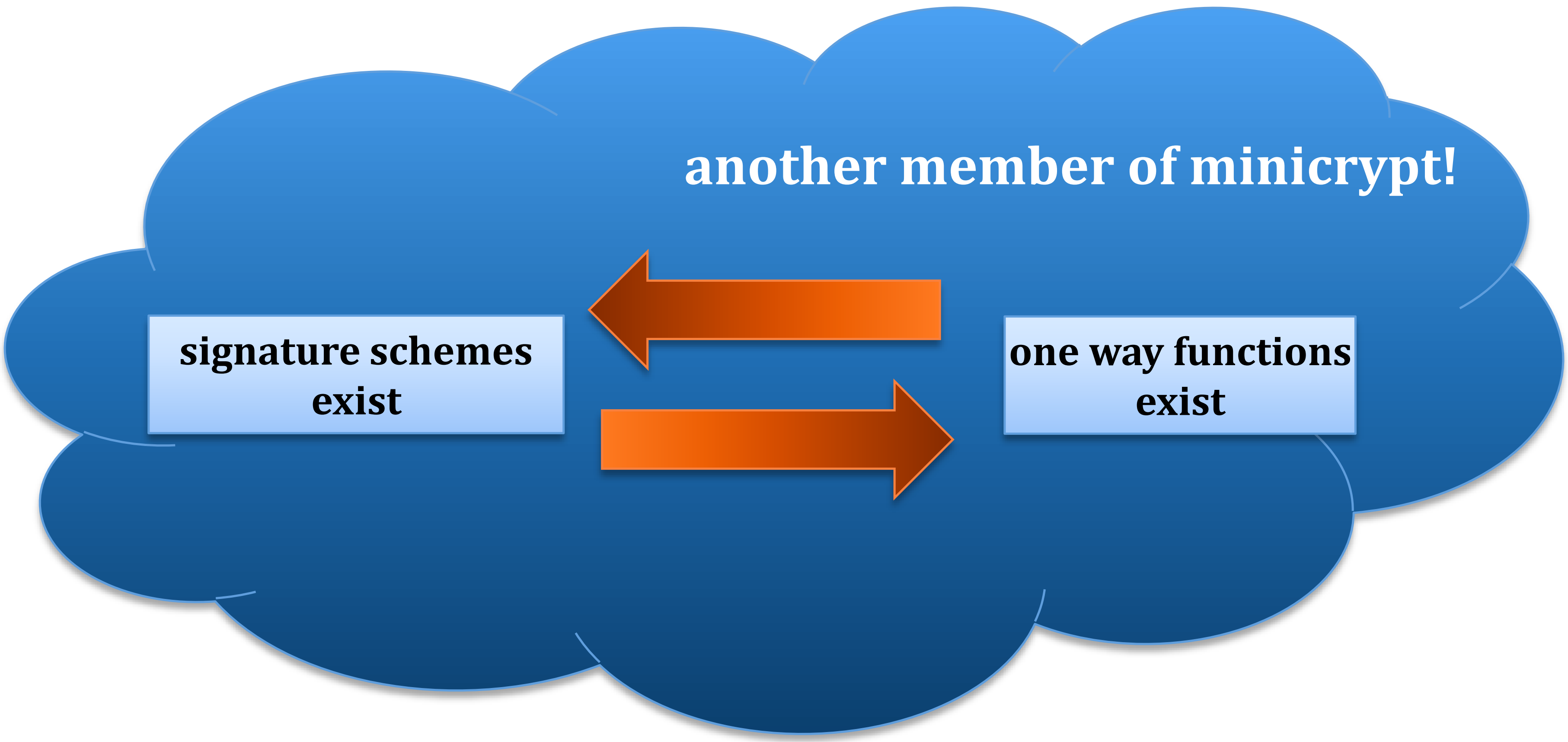
6 January 2011 | Technology Share

The PlayStation 3's security has been broken by hackers, potentially allowing anyone to run any software - including pirated games - on the console.

Plan

1. The definition of secure signature schemes
2. Signatures based on RSA, “hash-and-sign”, “full-domain-hash”
3. Constructions based on discrete log
   a) identification schemes
   b) Schnorr signatures
   c) DSA signatures
4. Theoretical constructions

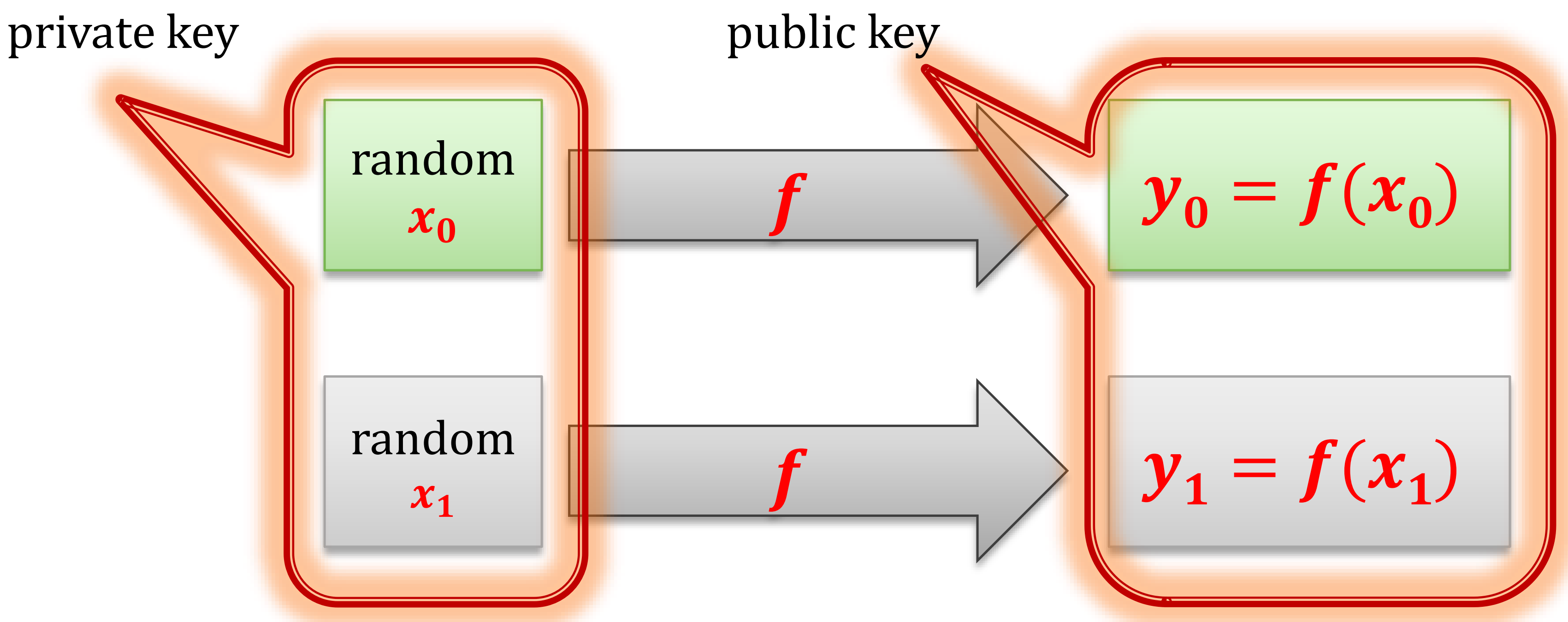
Signatures schemes can be constructed from any one-way function



One-time signatures (**Leslie Lamport**)

How to sign one bit?

f – a one way function

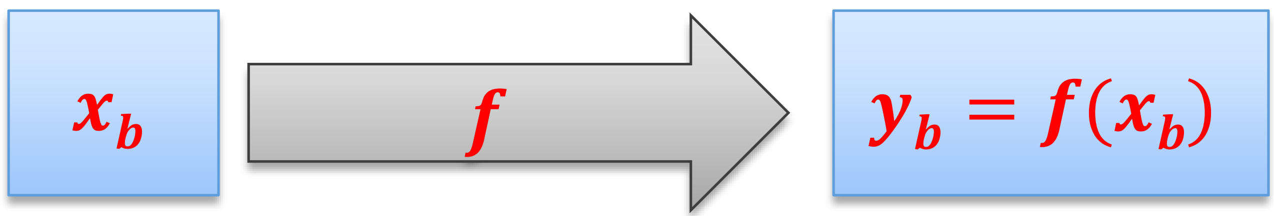


$\mathbf{Sign}\big((x_0, x_1), b\big) \;=\; x_b$

$\mathbf{Vrfy}\big((y_0, y_1), b, x\big) \;=\; \mathbf{yes}$ iff $f(x) \;=\; y_b$

Why is it secure?

To forge a signature on bit $\boldsymbol{b}$ the adversary needs to calculate $\boldsymbol{x_b}$ from $\boldsymbol{y_b}$



This should be infeasible, since $\boldsymbol{f}$ is one-way...

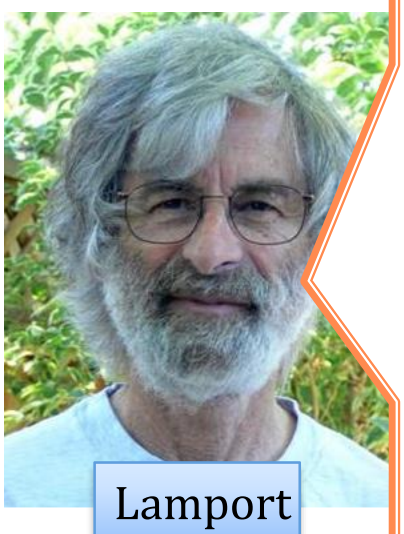
Constructing Digital Signatures from a One Way Function
SRI International Technical Report CSL-98 (October 1979).

At a **coffee house in Berkeley** around **1975**, **Whitfield Diffie** described a problem to me that he had been trying to solve: constructing a digital signature for a document. **I immediately proposed a solution**. Though **not very practical**--it required perhaps 64 bits of published key to sign a single bit--it was the first digital signature algorithm.

In **1978**, **Michael Rabin** published a paper titled *Digitalized Signatures* containing a more practical scheme for generating digital signatures of documents. **(I don't remember what other digital signature algorithms had already been proposed.)** However, his solution had some drawbacks that limited its utility.

[...] I didn't feel that it added much to what Rabin had done. **However, I've been told that this paper is cited in the cryptography literature and is considered significant, so perhaps I was wrong**.

from: research.microsoft.com/en-us/um/people/lamport/



How to sign longer messages?

We show a **one-time signature** scheme (one public key can be used at most once).

f – one way function

n – length of the message

private key sk:

$x_{0,1}$	$\ldots$	$x_{0,n}$
$x_{1,1}$	$\ldots$	$x_{1,n}$

all x_{ij}'s are random strings

public key pk:

$y_{0,1} = f(x_{0,1})$	$\ldots$	$y_{0,n} = f(x_{0,n})$
$y_{1,1} = f(x_{1,1})$	$\ldots$	$y_{1,n} = f(x_{1,n})$

$$\mathbf{Sign}_{\mathbf{Lamport}}\left(sk, (m_0, \ldots, m_n)\right) := \left(x_{m_0}, \ldots, x_{m_n}\right)$$

$$\mathbf{Vrfy}_{\mathbf{Lamport}}\left(pk, (m_0, \ldots, m_n), \left(x_{m_0}, \ldots, x_{m_n}\right)\right) :=$$

$$\textbf{check if } \left(f\left(x_{m_0}\right), \ldots, f\left(x_{m_n}\right)\right) = \left(y_{m_0}, \ldots, y_{m_n}\right)$$

Example

$n = 6$

$m = (1, 0, 1, 1, 0, 0)$

private key:

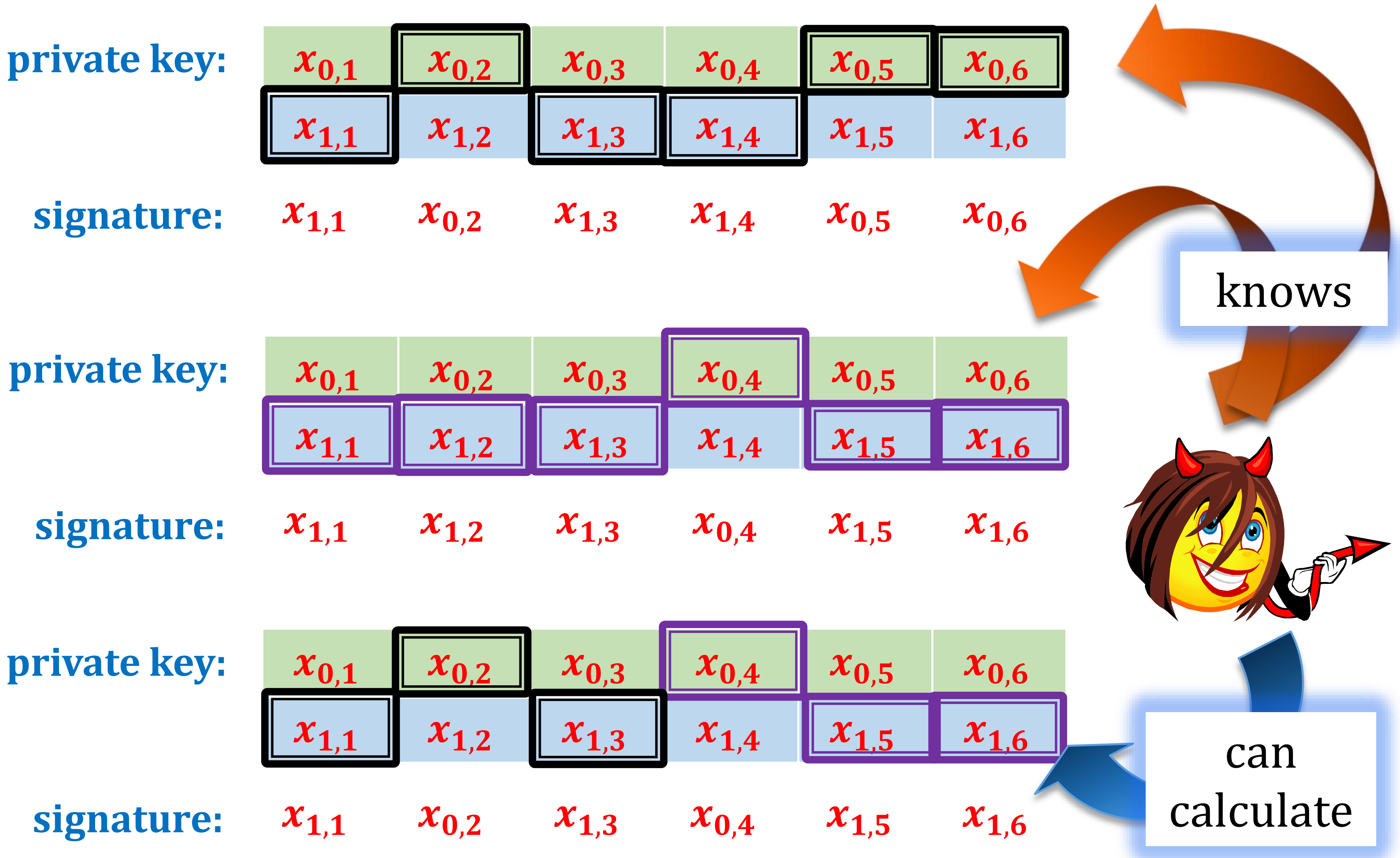
$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{0,4}$	$x_{0,5}$	$x_{0,6}$
$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$

public key:

$f(x_{0,1})$	$f(x_{0,2})$	$f(x_{0,3})$	$f(x_{0,4})$	$f(x_{0,5})$	$f(x_{0,6})$
$f(x_{1,1})$	$f(x_{1,2})$	$f(x_{1,3})$	$f(x_{1,4})$	$f(x_{1,5})$	$f(x_{1,6})$

signature: $x_{1,1}$ $x_{0,2}$ $x_{1,3}$ $x_{1,4}$ $x_{0,5}$ $x_{0,6}$

Why each key can be used at most once?



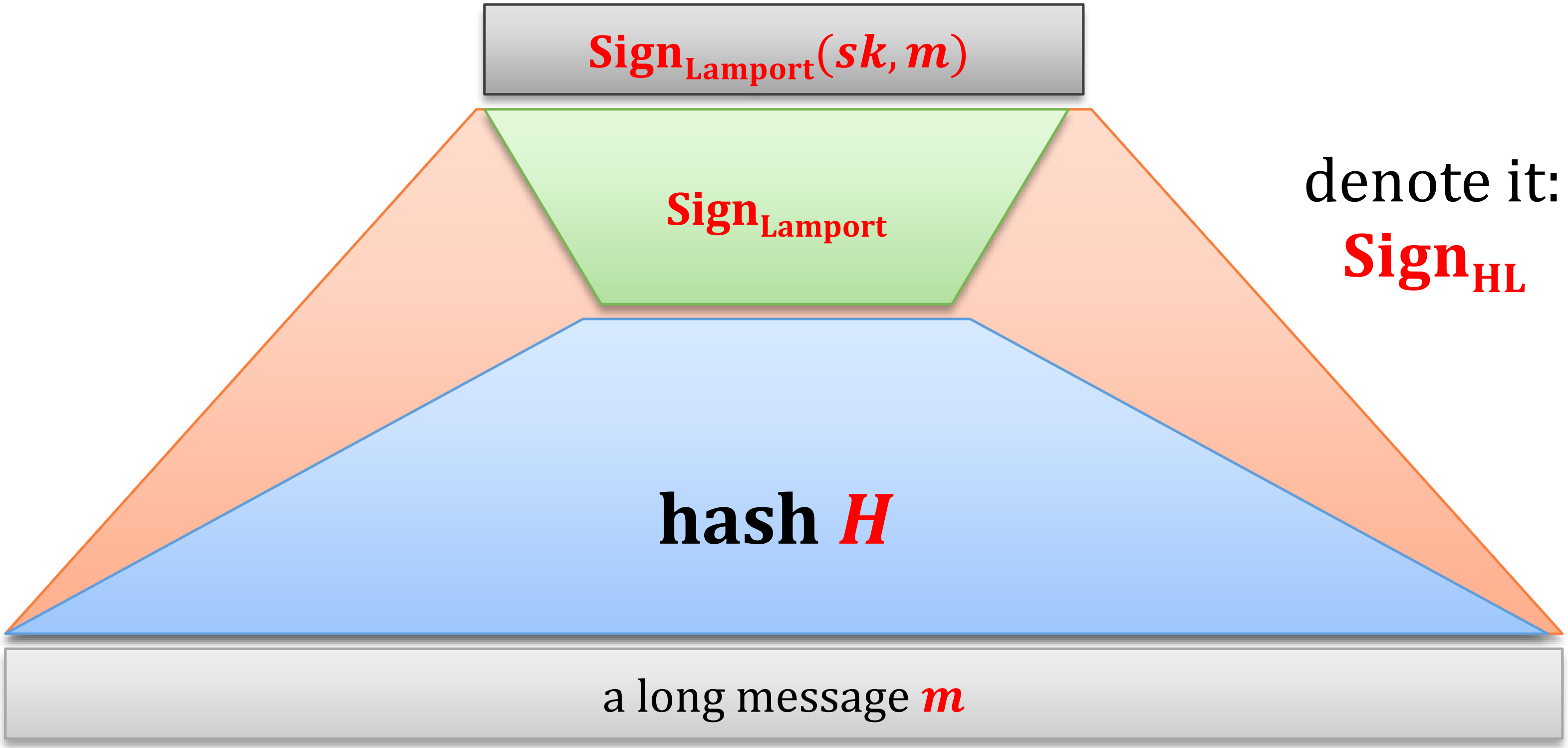
Problem

Signature is much **longer than the message**!

(and can be used **only once**)

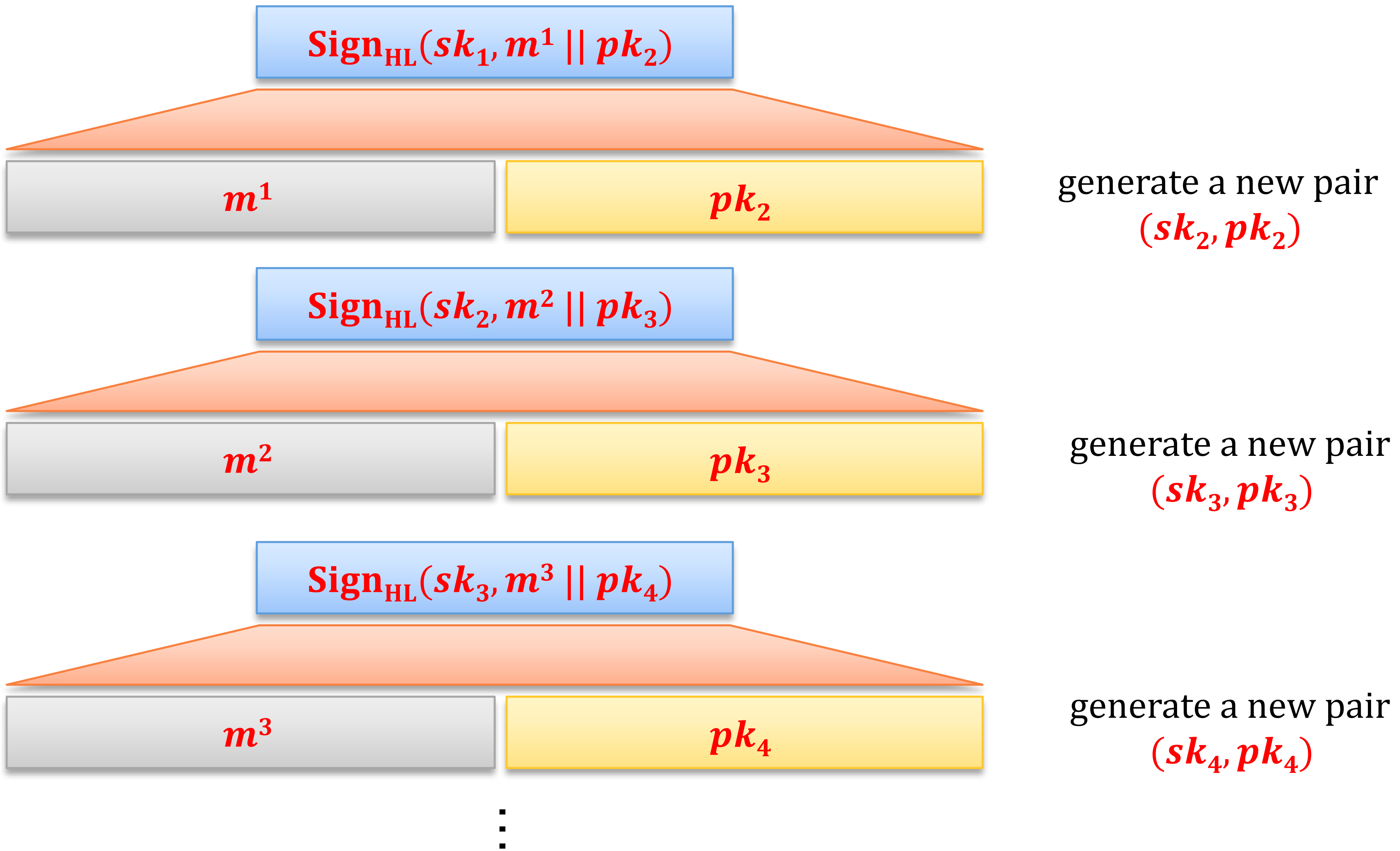
How to sign long messages?

Use hash functions



denote it:
$\mathbf{Sign}_{\mathbf{HL}}$

Idea: to sign multiple messages use "certification"



How to verify?

The signer needs to include the "certificate chain" in the signature.

$\mathbf{Sign}(sk_1, m^1) =$	$\mathbf{Sign}_{\mathrm{HL}}(sk_1, m^1 \,\|\, pk_2)$		← verify using pk_1
$\mathbf{Sign}(sk_1, m^2) =$	$\mathbf{Sign}_{\mathrm{HL}}(sk_1, m^1 \,\|\, pk_2)$	m^1	← verify using pk_1
	$\mathbf{Sign}_{\mathrm{HL}}(sk_2, m^2 \,\|\, pk_3)$		← verify using pk_2
$\mathbf{Sign}(sk_1, m^3) =$	$\mathbf{Sign}_{\mathrm{HL}}(sk_1, m^1 \,\|\, pk_2)$	m^1	← verify using pk_1
	$\mathbf{Sign}_{\mathrm{HL}}(sk_2, m^2 \,\|\, pk_3)$	m^2	← verify using pk_2
	$\mathbf{Sign}_{\mathrm{HL}}(sk_3, m^3 \,\|\, pk_4)$		← verify using pk_3

Problems

1. The length of the signature **grows linearly**

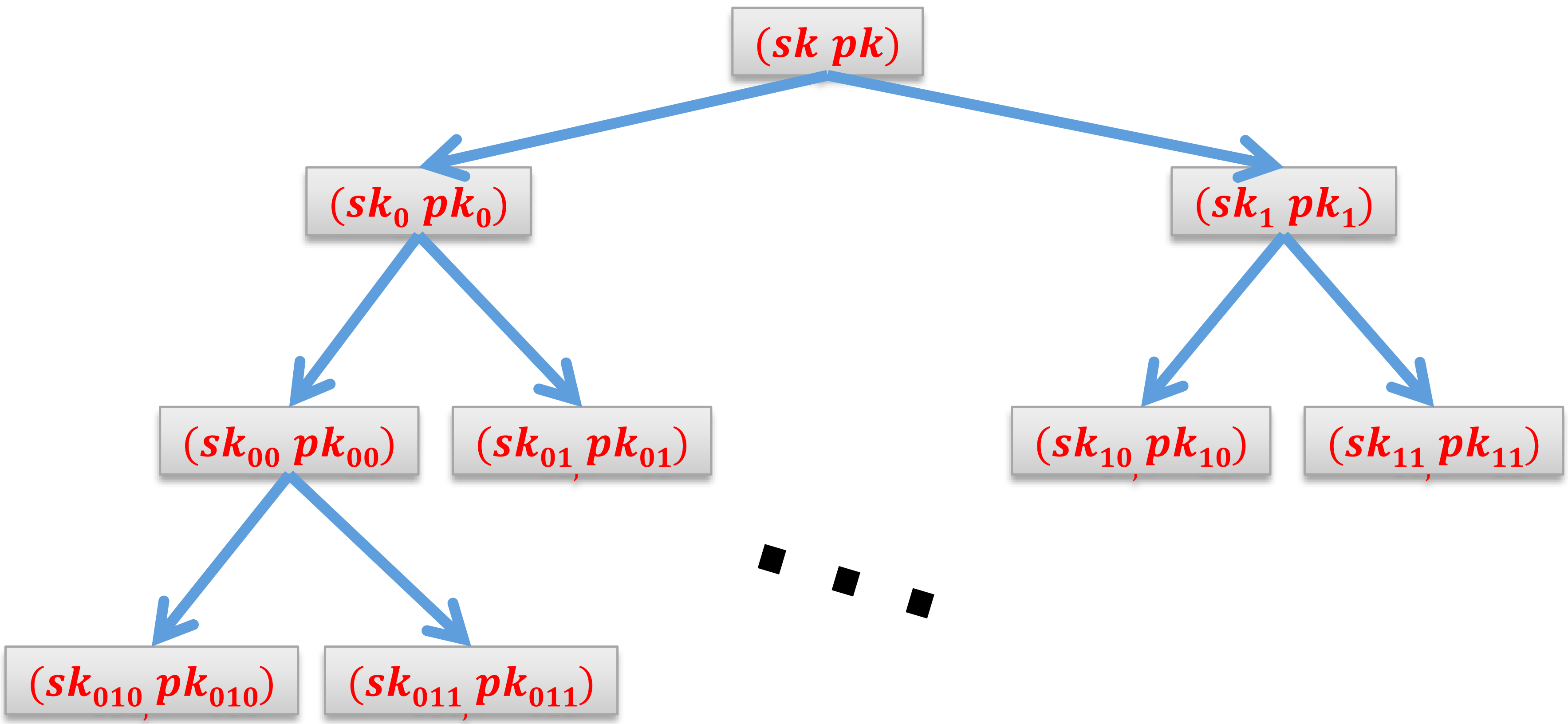
2. The signing algorithm needs to **have a state** ("memory")

Solution to the first problem

Instead of a **chain** use a **binary tree**:

"certify each time **two** public keys"

The tree:



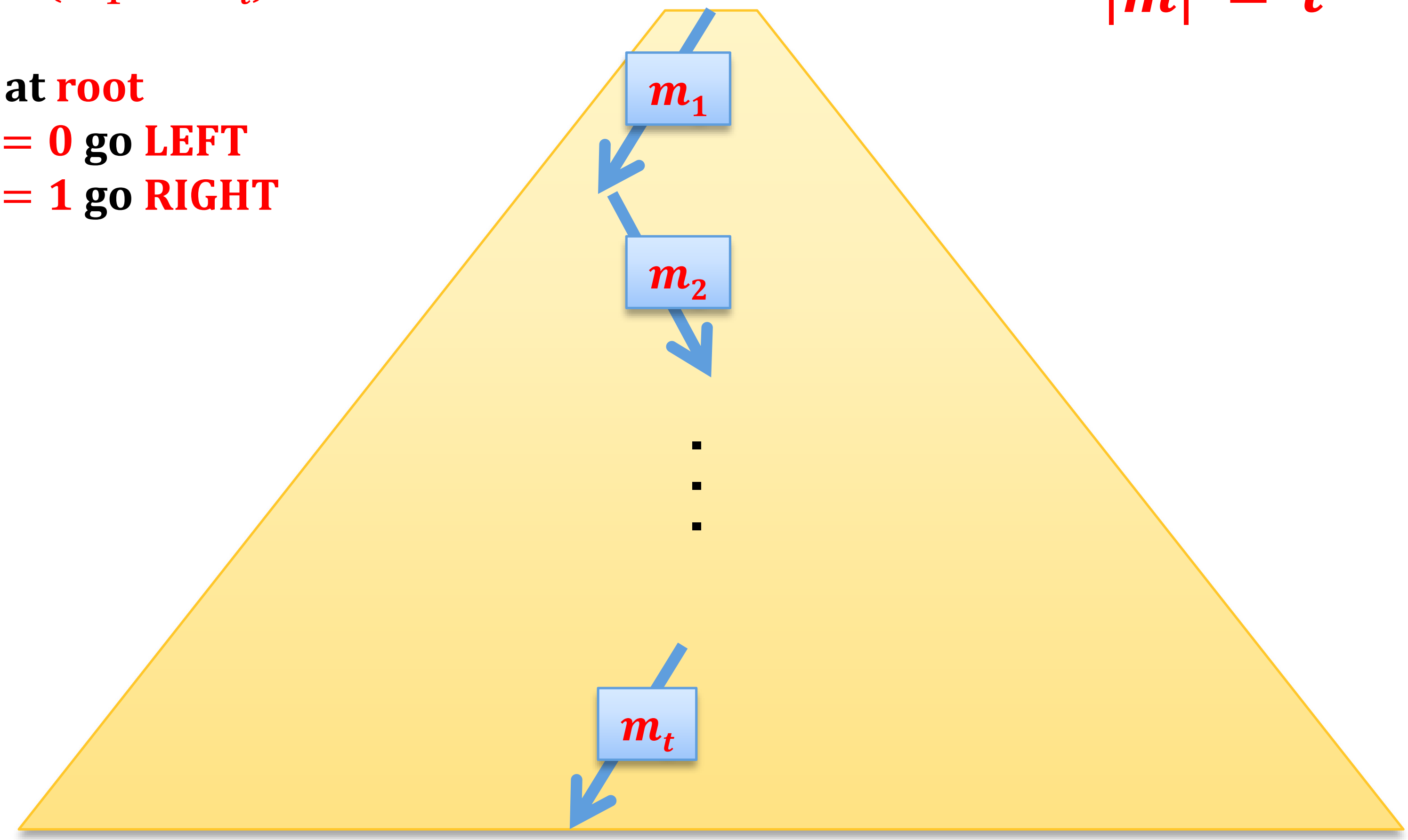
$(sk\ pk)$
$(sk_0\ pk_0)$
$(sk_1\ pk_1)$
$(sk_{00}\ pk_{00})$
(sk_{01},pk_{01})
(sk_{10},pk_{10})
(sk_{11},pk_{11})
(sk_{010},pk_{010})
(sk_{011},pk_{011})

The details

$m = (m_1, \ldots, m_t)$

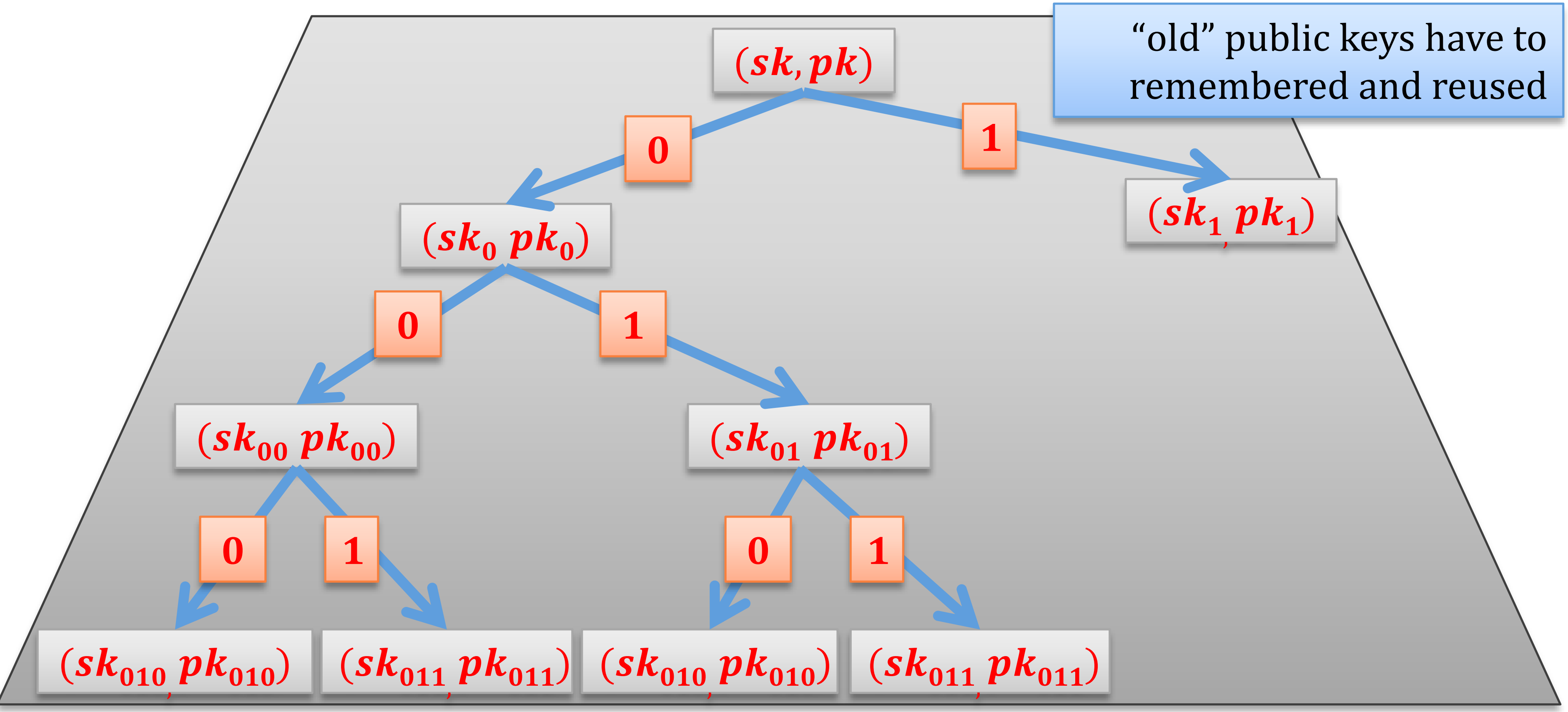
start at root
if $m_i = 0$ go LEFT
if $m_i = 1$ go RIGHT

now the "chain" has length
$|m| = t$



use the key in the **LEAF** to sign m

The key pairs are generated on-fly



$\mathbf{Sign}(sk, (0,1,0)) =$

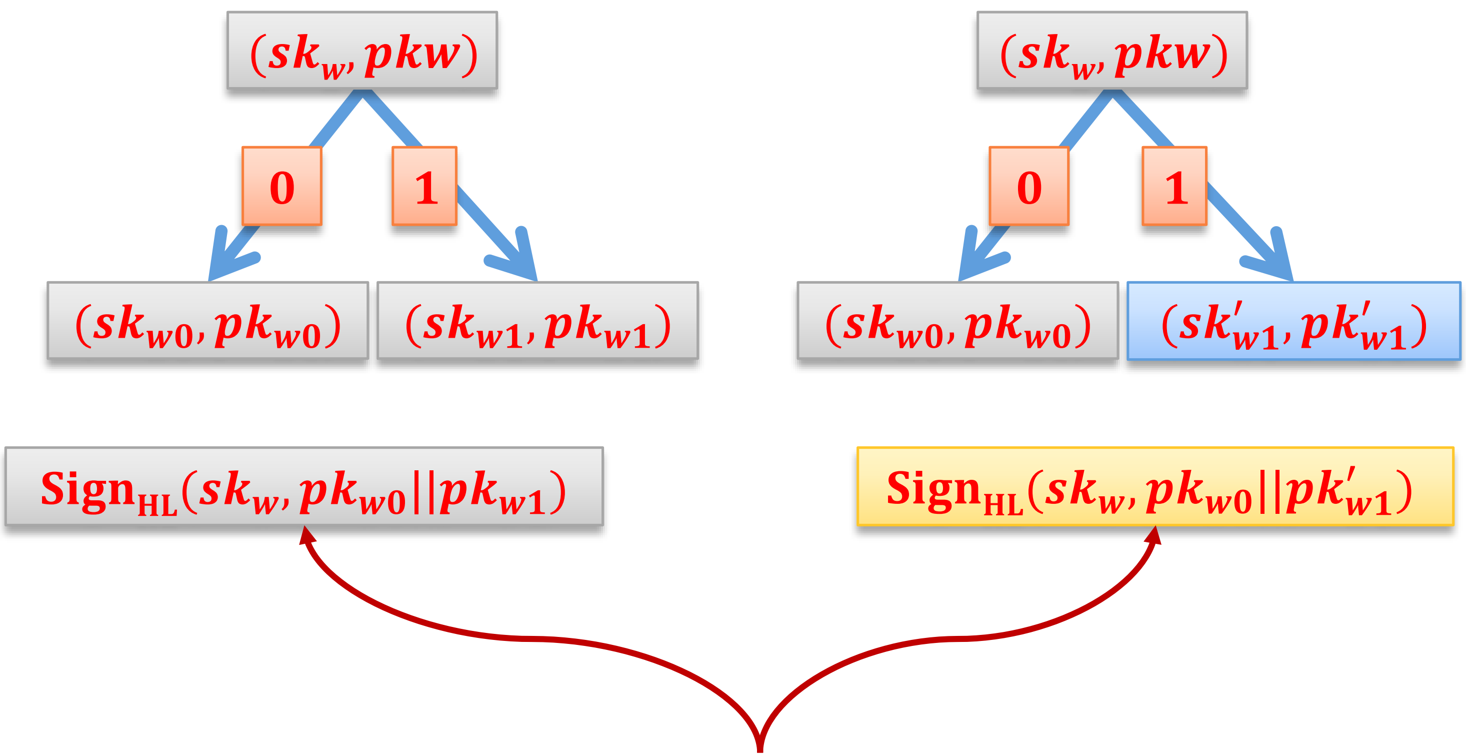
$\mathbf{Sign}_{\mathrm{HL}}(sk, pk_0 \| pk_1)$	$\mathbf{Sign}_{\mathrm{HL}}(sk_0, pk_{00} \| pk_{01})$	$\mathbf{Sign}_{\mathrm{HL}}(sk_{01}, pk_{010} \| pk_{011})$	$\mathbf{Sign}_{\mathrm{HL}}(sk_{010}, (0,1,0))$

$\mathbf{Sign}(sk, (0,0,0)) =$

$\mathbf{Sign}_{\mathrm{HL}}(sk, pk_0 \| pk_1)$	$\mathbf{Sign}_{\mathrm{HL}}(sk_0, pk_{00} \| pk_{01})$	$\mathbf{Sign}_{\mathrm{HL}}(sk_{00}, pk_{000} \| pk_{001})$	$\mathbf{Sign}_{\mathrm{HL}}(sk_{000}, (0,0,0))$

Why we have to remember the old keys?

Suppose we don't:



so we signed two different messages with the same key!

Problem

The tree is constructed on-fly, so we need to remember the state.

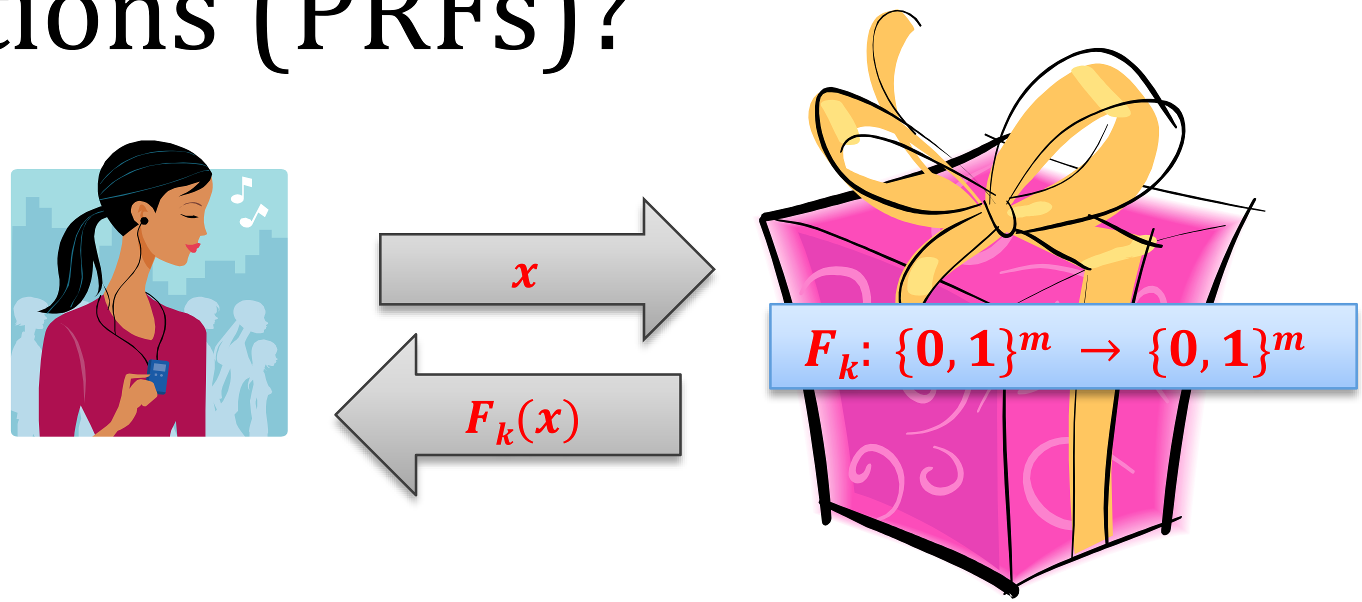
A stupid solution:

generate the whole tree beforehand.

A better solution:

generate the whole tree pseudorandomly and just remember the seed.

Remember the pseudorandom functions (PRFs)?



For a random key k and any $x_1, \ldots, x_t$ the values

$F_k(x_1), \ldots, F_k(x_t)$

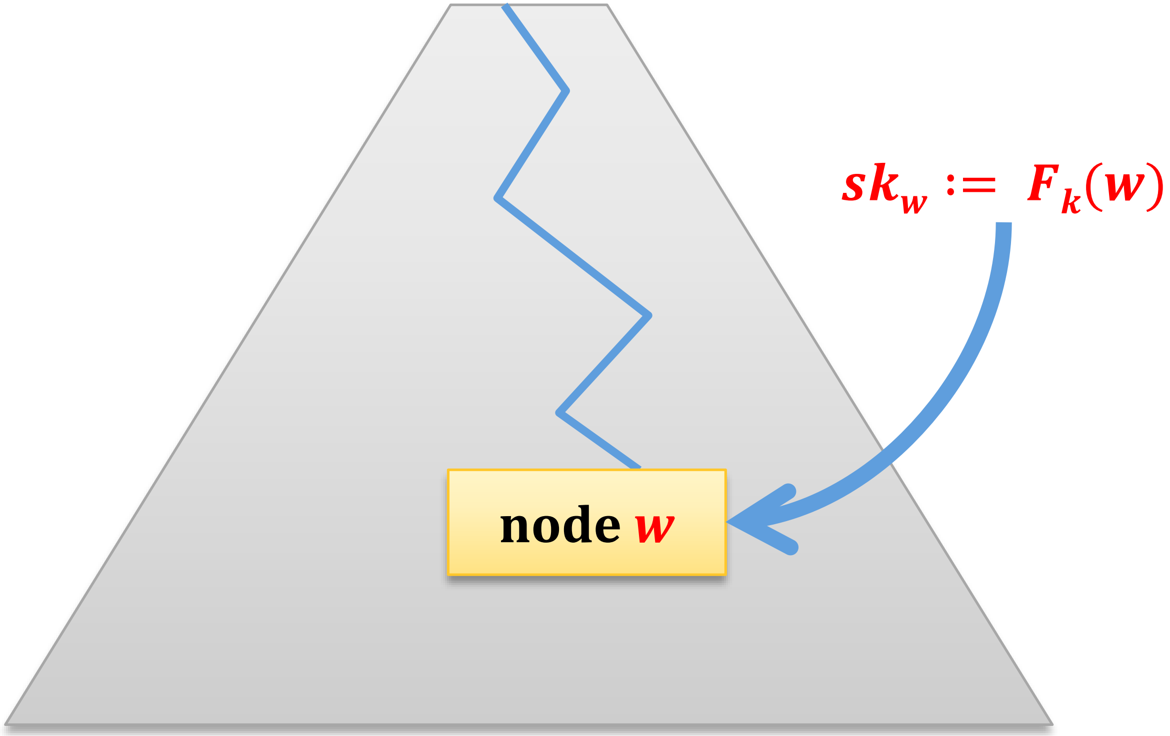
"look random"

Solution

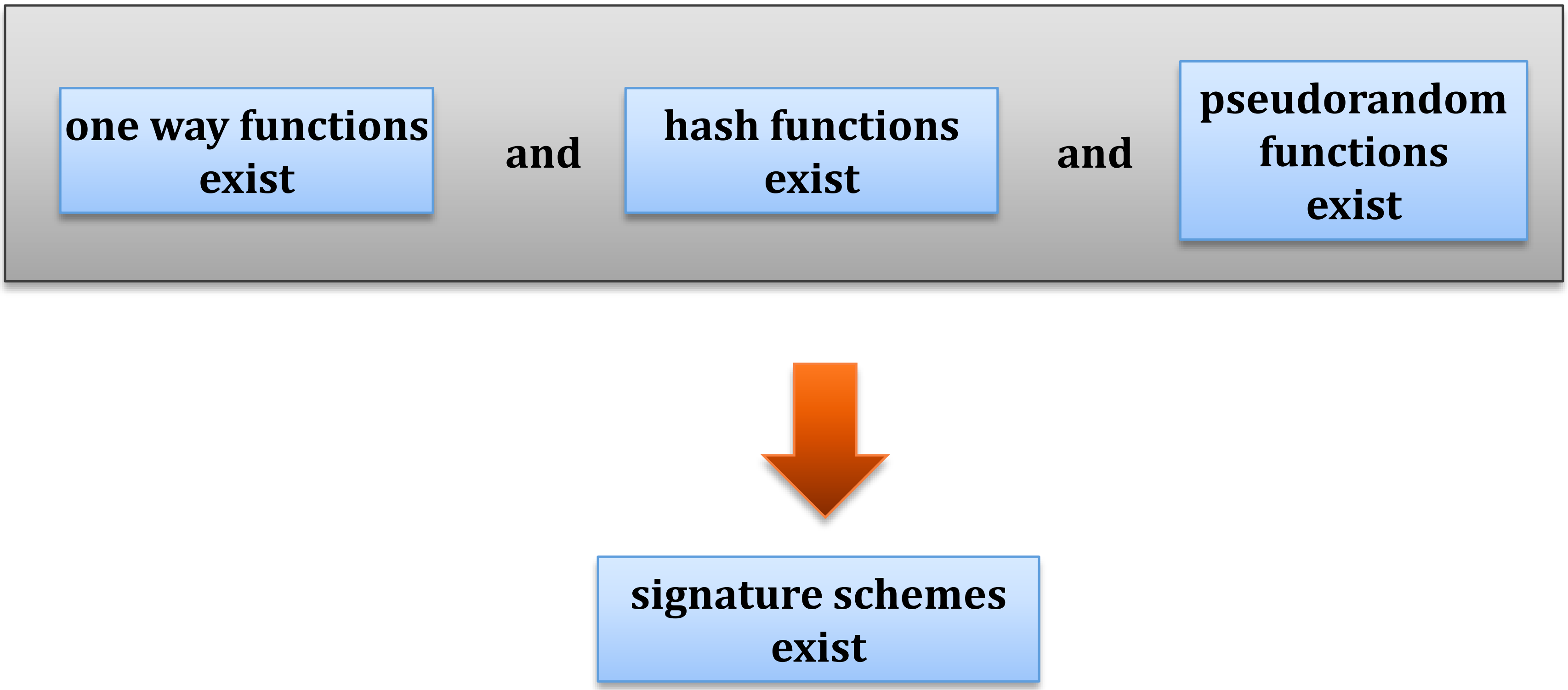
Take some PRF F

private key: (sk, k)
sk – a private key for **hashed Lamport**
k – a key for **PRF** F

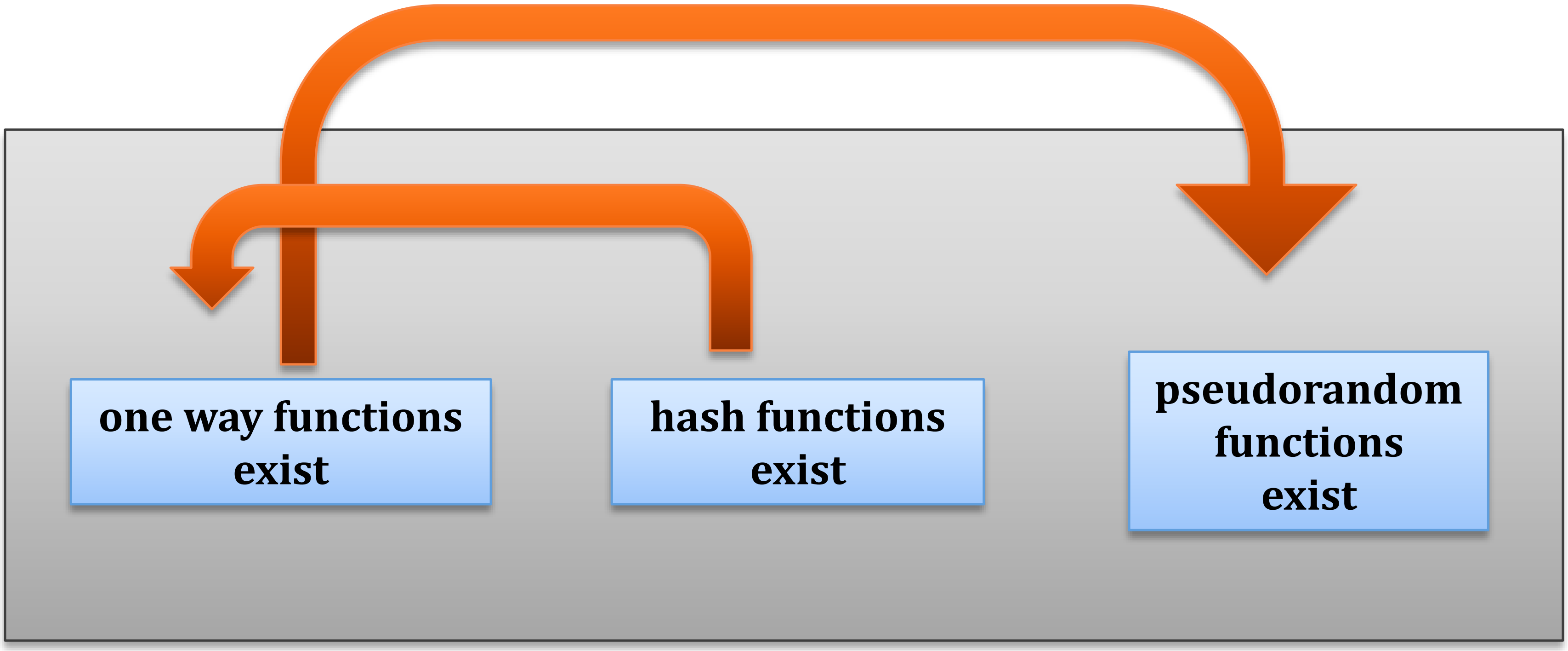
public key:
pk – a public key for **hashed Lamport**



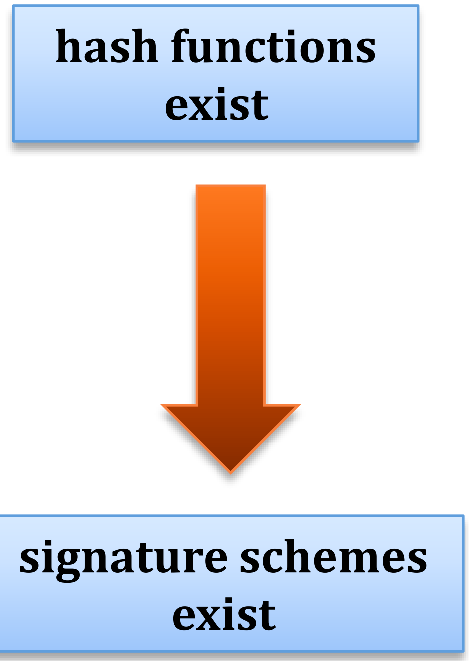
We have shown that



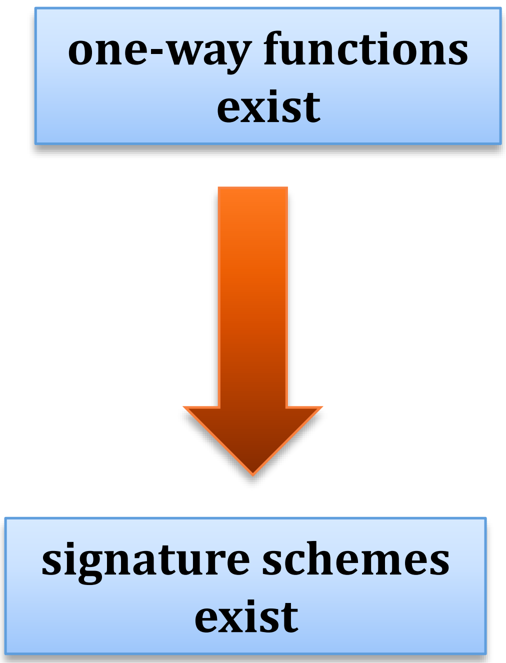
But we know that



Therefore we have shown that



The proof that



is more complicated