Lecture 4
Hash Functions

Stefan Dziembowski
www.crypto.edu.pl/Dziembowski

University of Warsaw

2.11.16 **version 1.0**

Secure communication

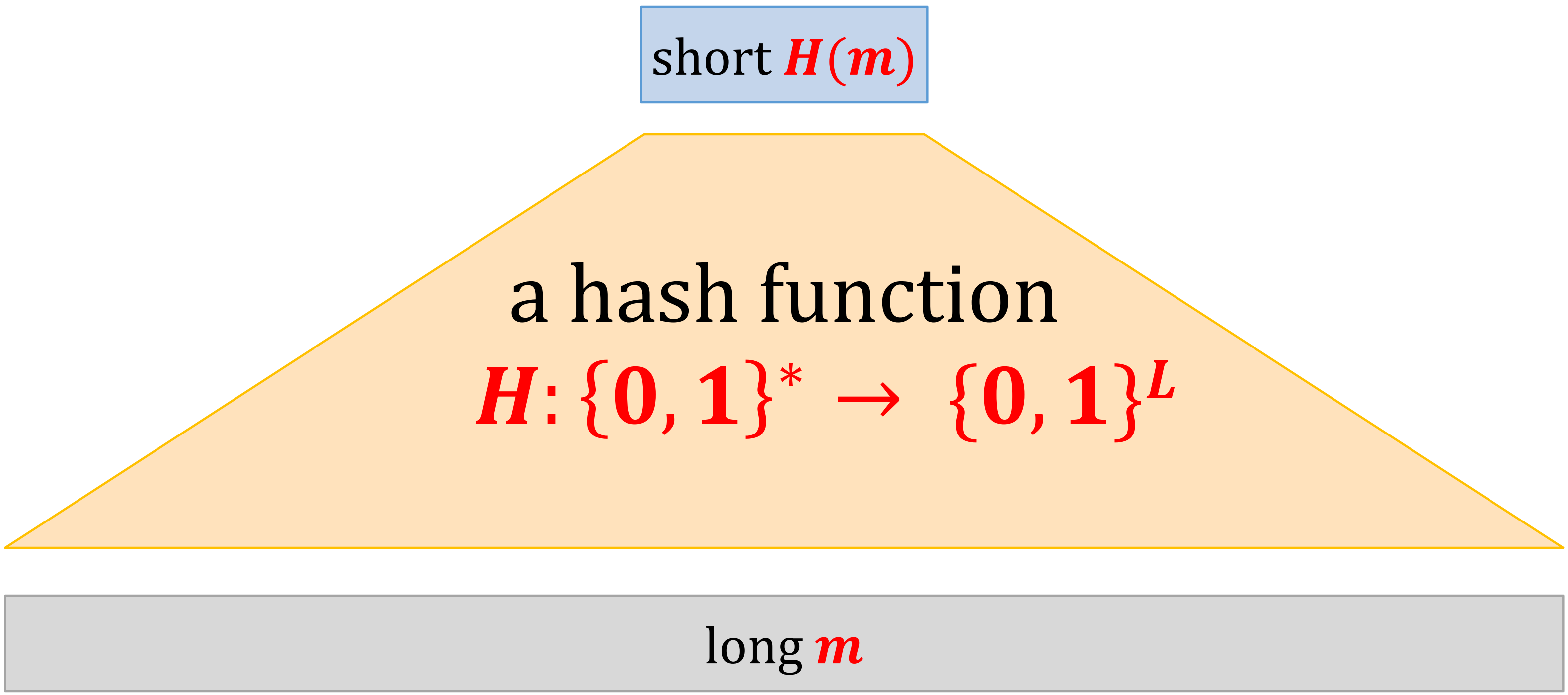
	encryption	authentication
private key	**1** private key encryption	**2** private key authentication
public key	**3** public key encryption	**4** signatures

we will discus this on the next lecture, but first, we talk about the hash functions

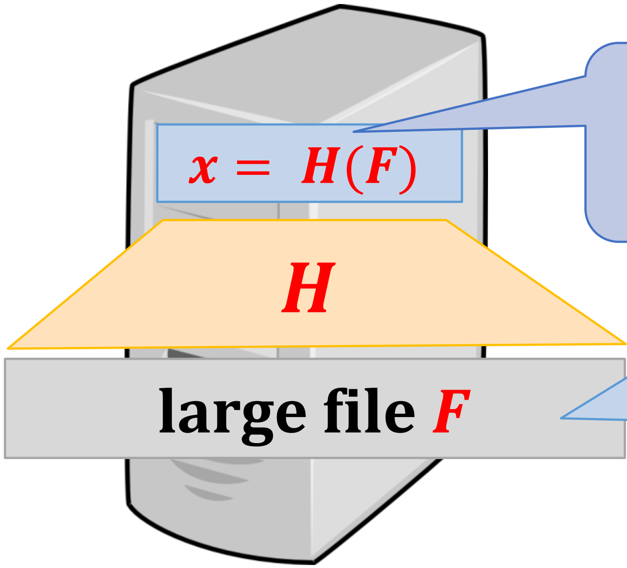
Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

Hash functions



Example of an application: file fingerprinting



"fingerprint"
(publish this through a **secure channel**)

everybody who downloads F can verify its correctness by computing $F(H)$ and comparing with x.

Example
(ubuntu.com):

64-bit PC (amd64, x86_64) (Recommended)

1. Ubuntu 15.10 "Wily Werewolf" 40MB (MD5: d1498749e073ef7fd09f84478f299bea, SHA1: aa659499dc300fe2be00d756180793093cc15014)
2. Ubuntu 15.04 "Vivid Vervet" 40MB (MD5: 3b00a4573b11fb1f85eaa05918971789, SHA1: 97282a3b066de4ee4c9409979737f3911f95ceab)

What properties should a hash function H have?

Minimal requirement: second preimage-resistance:

More precisely, the following problem should be **hard for any efficient adversary A**:

- **given**: "random" $m \in \{0,1\}^*$
- **find**: $m' \neq m$ such that

$$H(m) = H(m')$$

Q: Is it enough?

Second preimage resistance may be in many cases be too weak

What if the adversary can somehow influence the choice of $\boldsymbol{m}$?

<u>**For example**</u>: Ubuntu has many contributors. What if one of them is malicious?

Idea: modify the definition by allowing the adversary to choose $\boldsymbol{m}$ himself.

New game for the adversary:

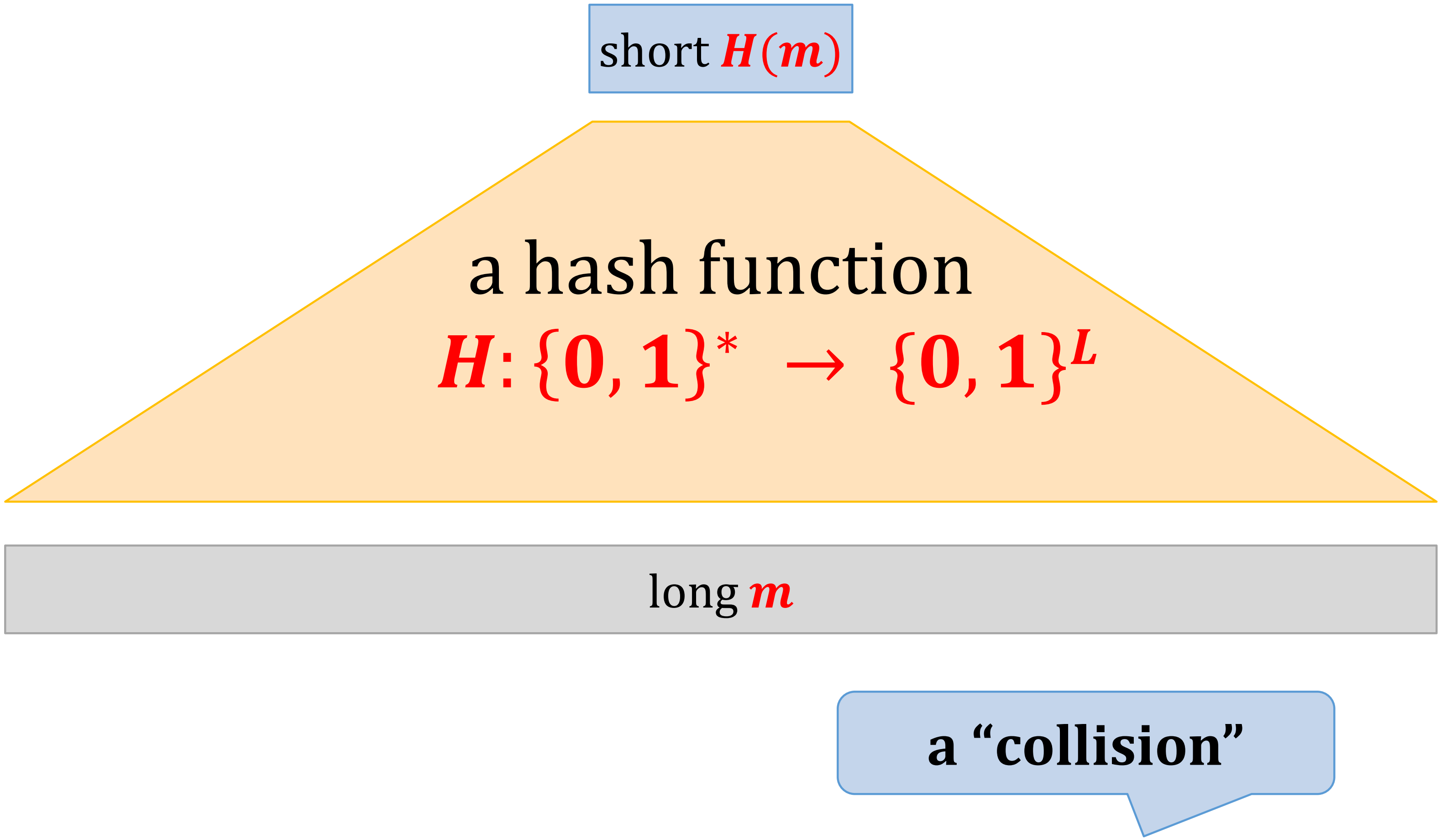
if this problem is hard then a function is called "**collision-resistant**"

find: $\boldsymbol{m}$
find: $\boldsymbol{m'} \neq \boldsymbol{m}$ such that $\boldsymbol{H(m) = H(m')}$

$\Leftrightarrow$

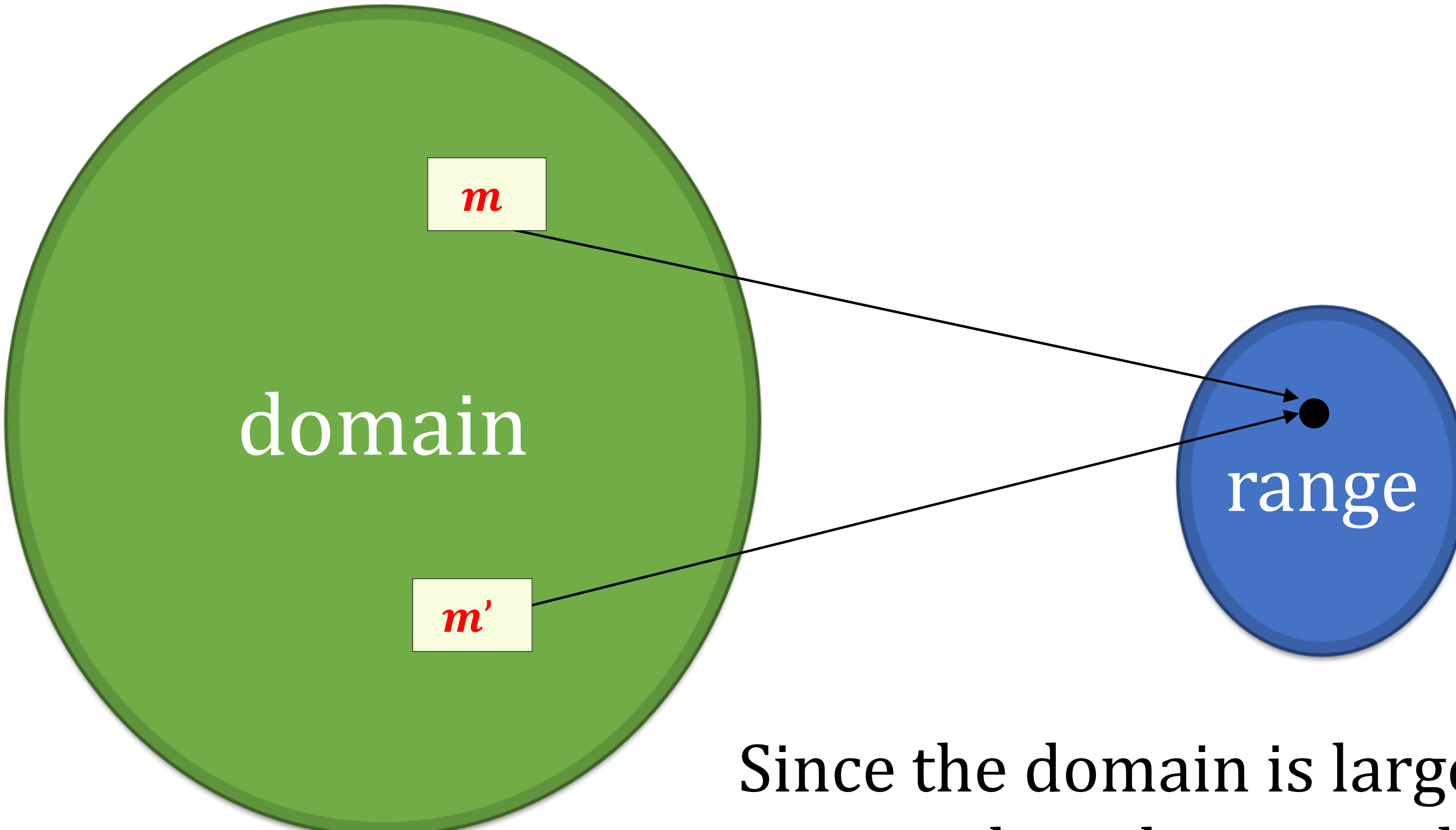
find: $\boldsymbol{m} \neq \boldsymbol{m'}$ such that $\boldsymbol{H(m) = H(m')}$

Collision-resistant hash functions



Requirement: it should be hard to find a pair (m, m') such that $H(m) = H(m')$

Collisions always exist



Since the domain is larger than the range the collisions have to exist.

“Practical definition”

H is a **collision-resistant hash function** if it is “*practically impossible to find collisions in* ***H***”.

Popular hash functions (we will present them in more detail on the next lecture):

- **MD5** (now considered broken
- **SHA1** (also has weaknesses), **SHA256**

based on **the Merkle-Damgård transformation**

- **Keccak**

based on the **sponge construction**

Hash functions can also be constructed using mathematical tools like number theory.

How to formally define “collision resitance”?

<u>**Idea**</u>: Say something like: $\boldsymbol{H}$ is a **collision-resistant hash function** if

$$\forall \quad \boldsymbol{P(A \text{ finds a collision in } H)} \text{ is small}$$

efficient adversary $\boldsymbol{A}$

<u>**Problem**</u>

For a fixed $\boldsymbol{H}$ there **always exist** a constant-time algorithm that “finds a collision in $\boldsymbol{H}$” in **constant time**.

It may be hard to <u>**find**</u> such an algorithm, but it always exists!

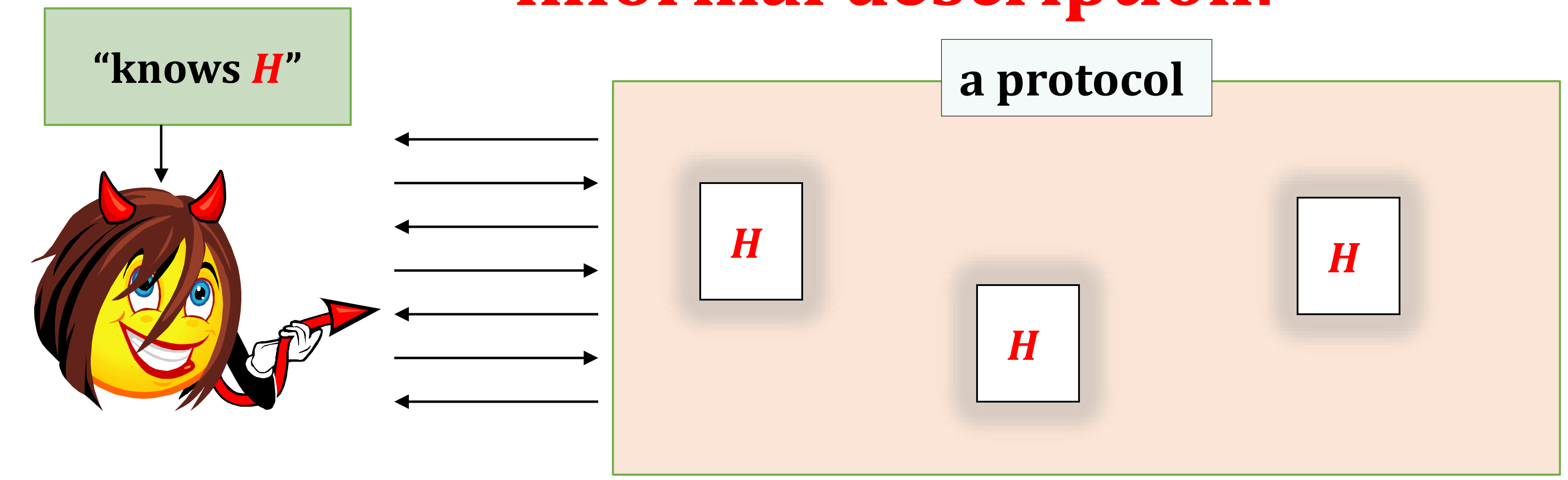
Solution

When we prove theorems we will always consider

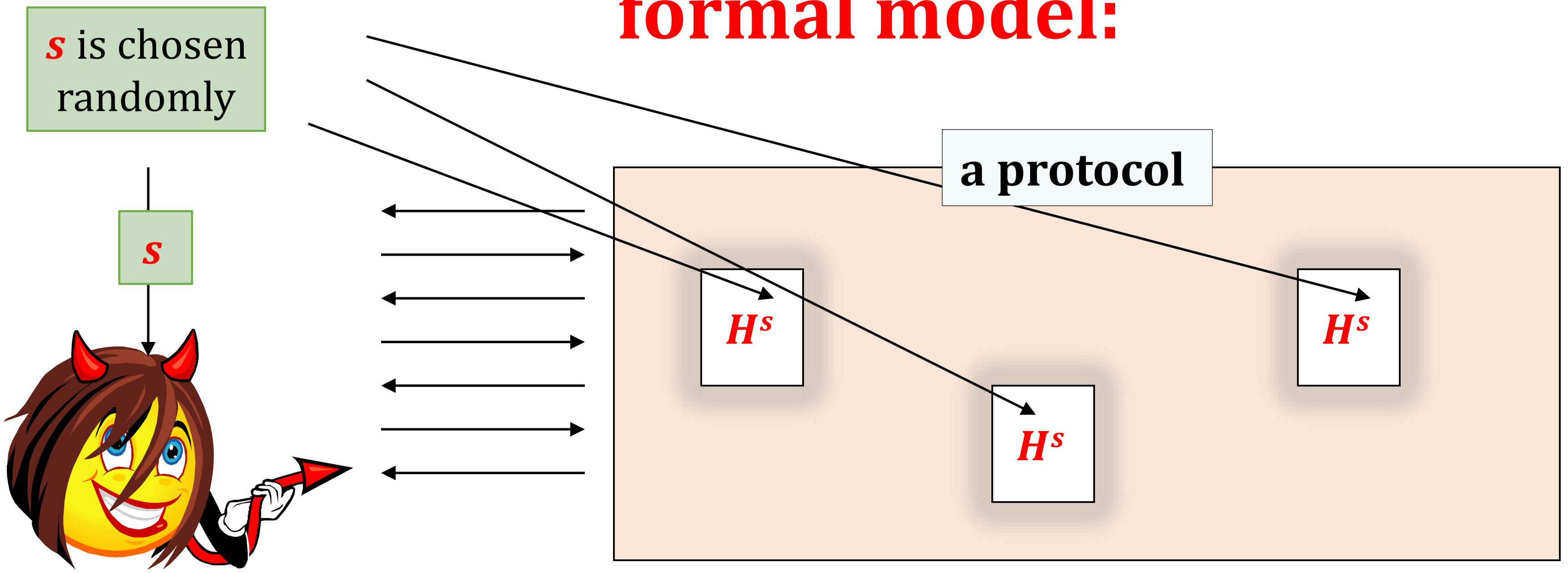
families of hash functions
indexed by a key *s*:

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

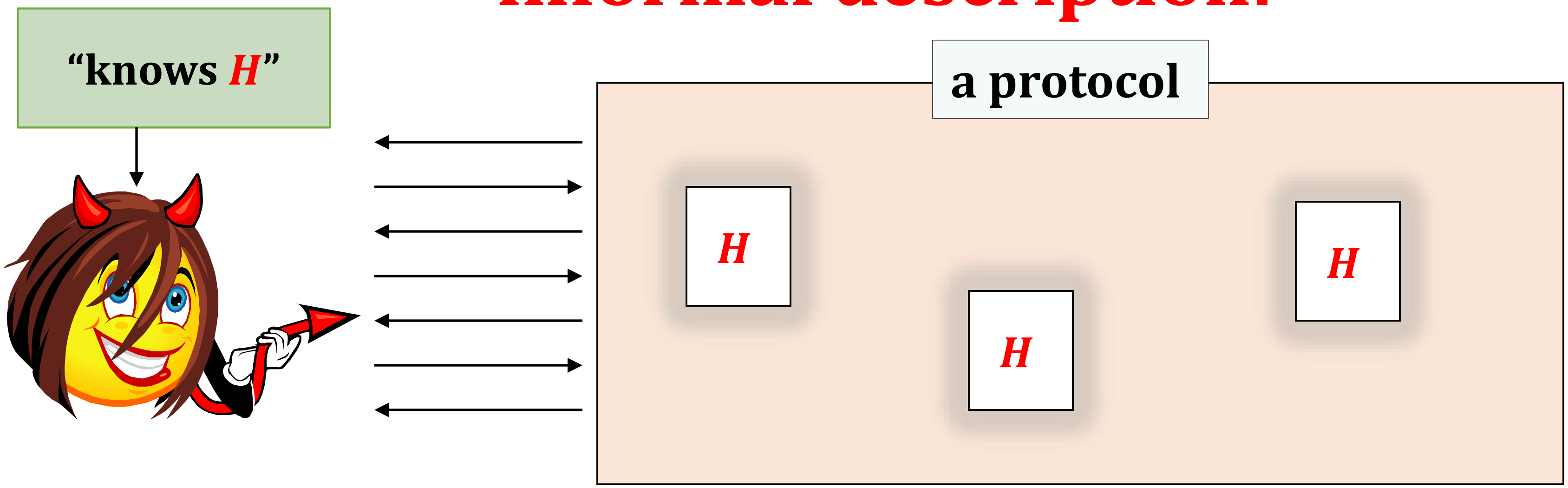
informal description:



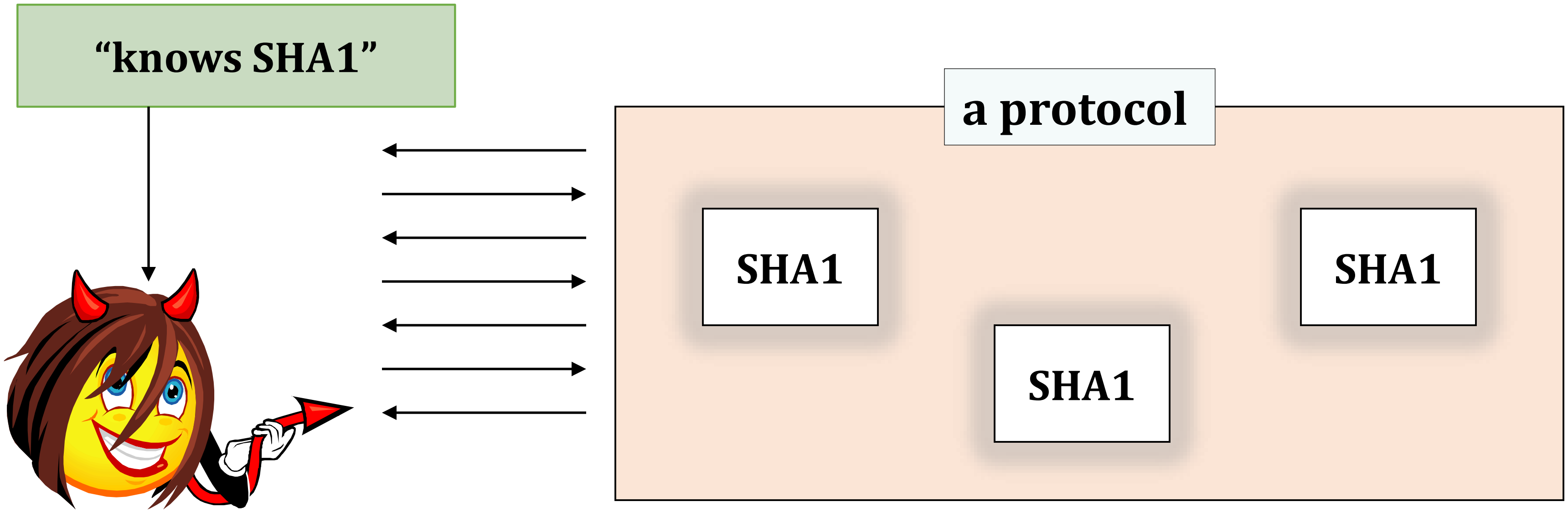
formal model:



informal description:



real-life implementation (example):



Hash functions – the functional definition

A **hash function** is a probabilistic polynomial-time algorithm $\boldsymbol{H}$ such that:

$\boldsymbol{H}$ takes as input a key $\boldsymbol{s} \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$ and a message $\boldsymbol{m} \in \{\mathbf{0}, \mathbf{1}\}^*$ and outputs a string

$$\boldsymbol{H}^{\boldsymbol{s}}(\boldsymbol{m}) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}(\boldsymbol{n})}$$

where $\boldsymbol{L}(\boldsymbol{n})$ is some fixed function.

Hash functions – the security definition [1/2]

1^n

selects a random $s \in \{0,1\}^n$

s

outputs (m, m')

We say that adversary A breaks the function H if

$$H^s(m) = H^s(m').$$

Hash functions – the security definition [2/2]

$\boldsymbol{H}$ is a **collision-resistant hash function** if

$\forall$ polynomial-time adversary $\boldsymbol{A}$

$\boldsymbol{P}(\boldsymbol{A}$ **breaks** $\boldsymbol{H})$ is negligible

A weaker requirement: **pre-image resistance**

Intuitively: “it’s hard to find a pre-image of $\boldsymbol{H^s}$”

Main difference:

$\mathbf{1^n}$

selects random
$\boldsymbol{s \leftarrow \{0,1\}^n}$
$\boldsymbol{m \leftarrow \{0,1\}^n}$

$\boldsymbol{s, H^s(m)}$

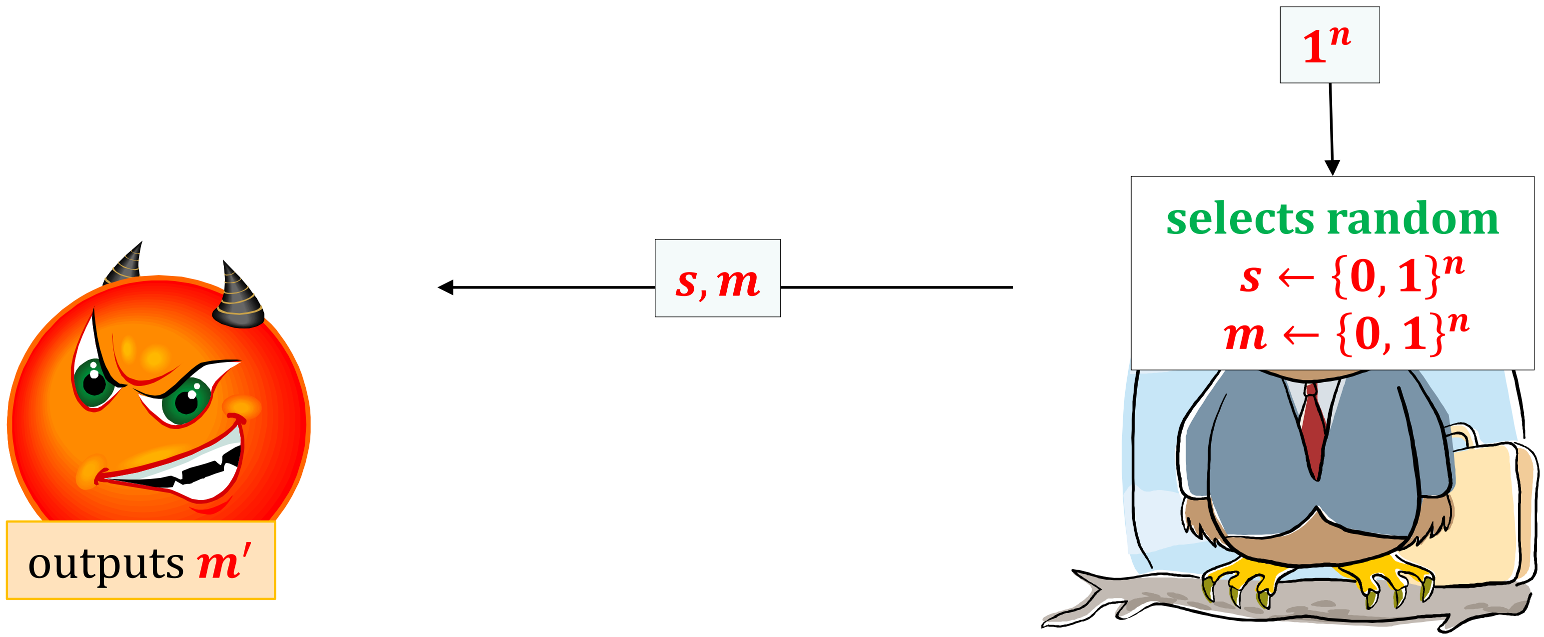
outputs $\boldsymbol{m'}$

We say that adversary $\boldsymbol{A}$ breaks the function $\boldsymbol{H}$ if

$$\boldsymbol{H^s(m) = H^s(m')}$$

Yet another requirement: **second pre-image resistance**

Intuitively: "it's hard to find a second pre-image of H^s"



We say that adversary A breaks the function H if $m' \neq m$ and $H^s(m) = H^s(m')$

Fact

The following implications hold:

H is collision resistant

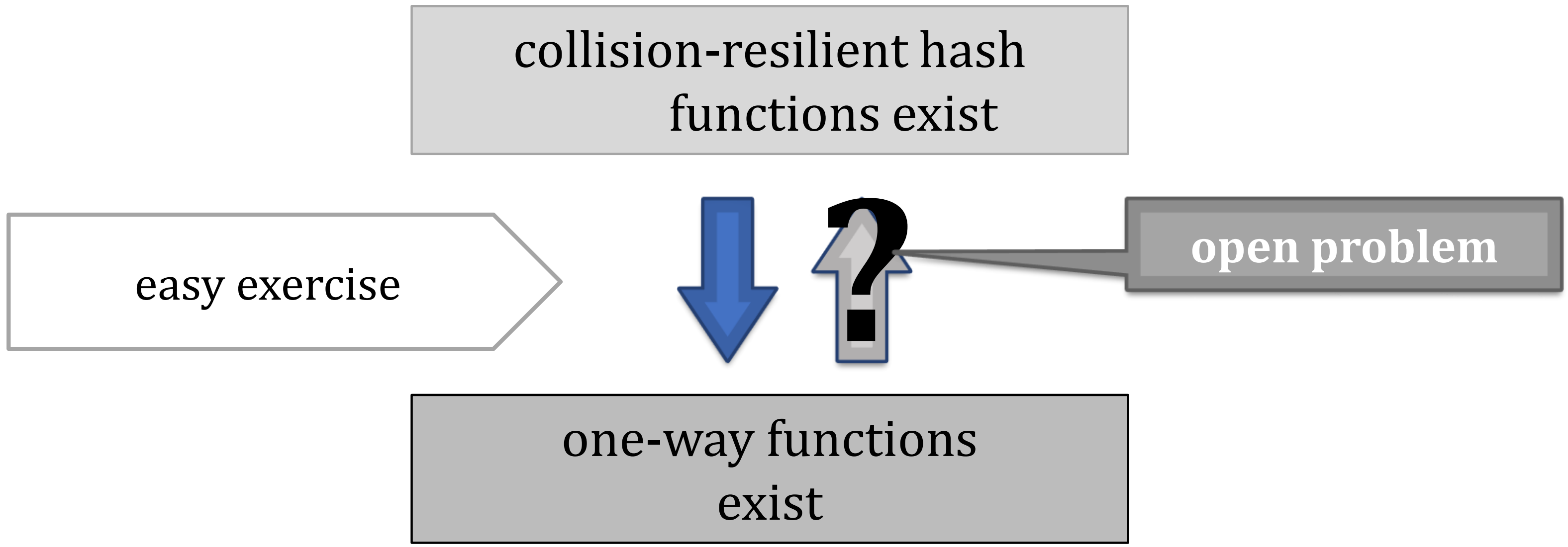
$\Downarrow$

H is second preimage resistant

$\Downarrow$

H is pre-image resistant

Do collision-resilient hash functions belong to minicrypt?



[D. Simon: Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions? 1998]:

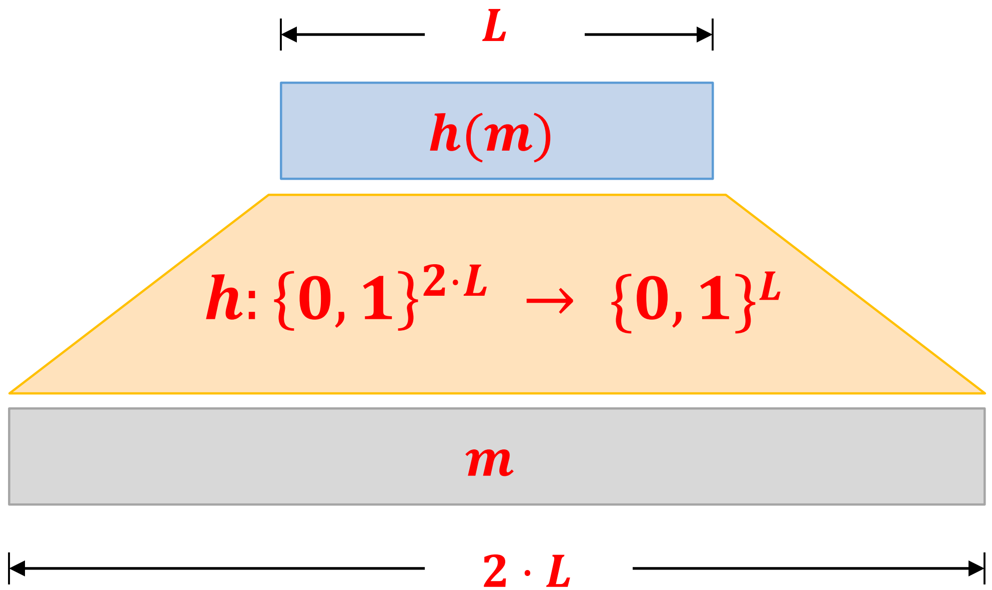
there is no “black-box reduction”.

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

A common method for constructing hash functions

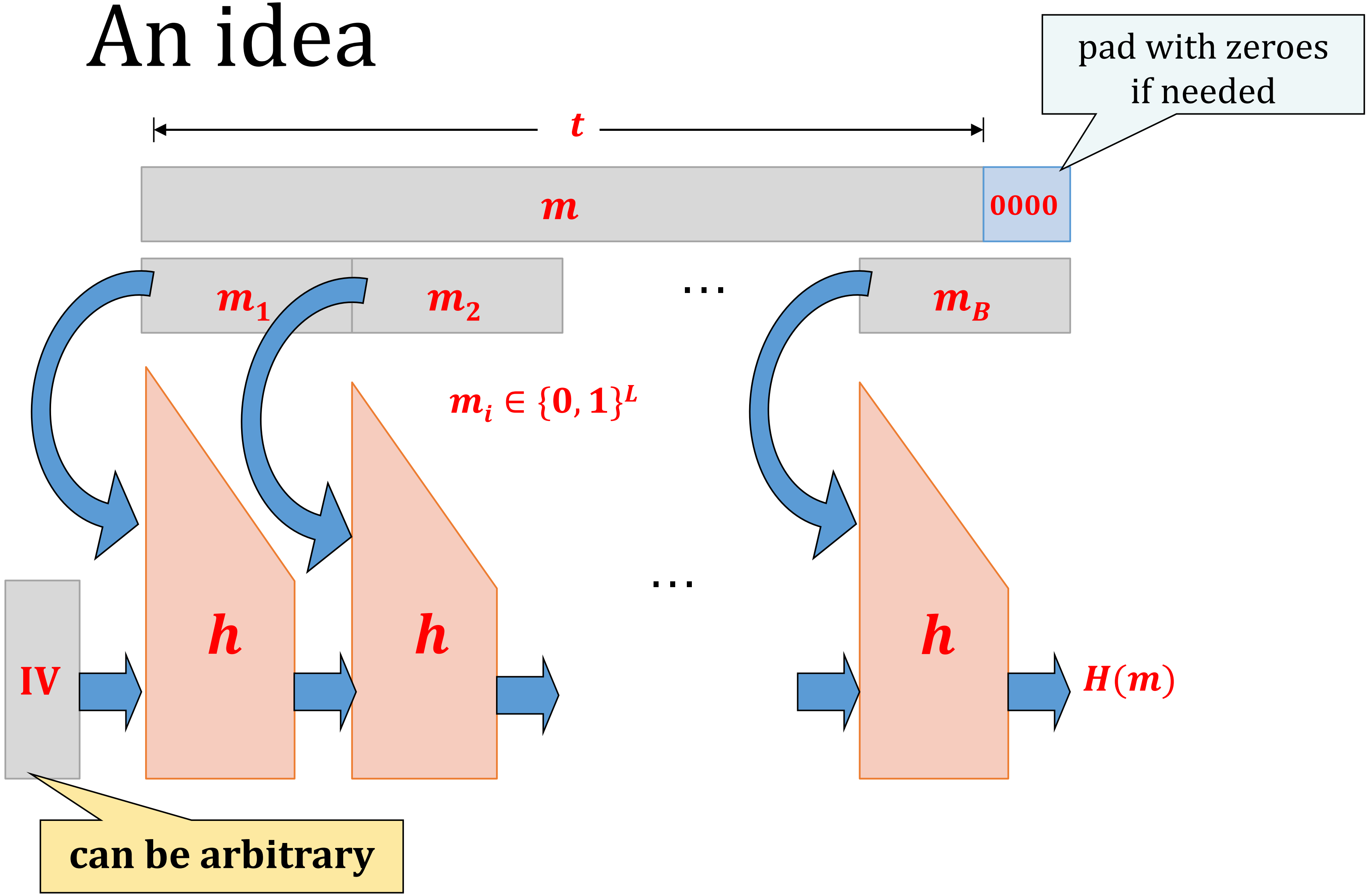
1. Construct a "***fixed-input-length***" collision-resistant hash function



Call it: a collision-resistant **compression function**.

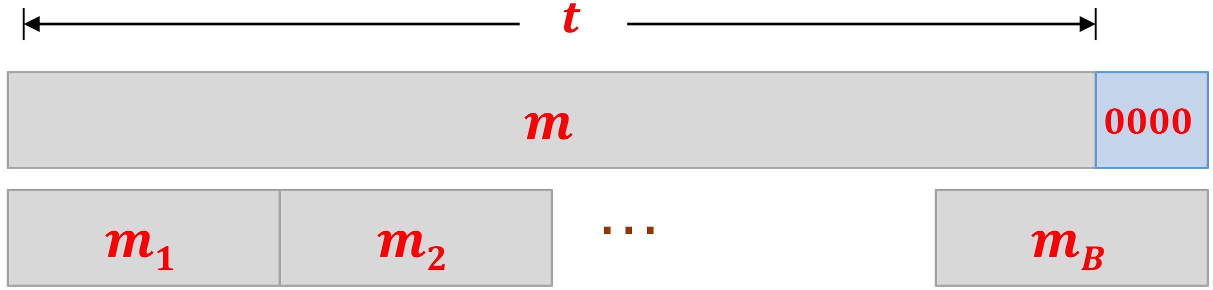
2. Use it to construct a hash function.

An idea



This doesn't work...

Why is it wrong?

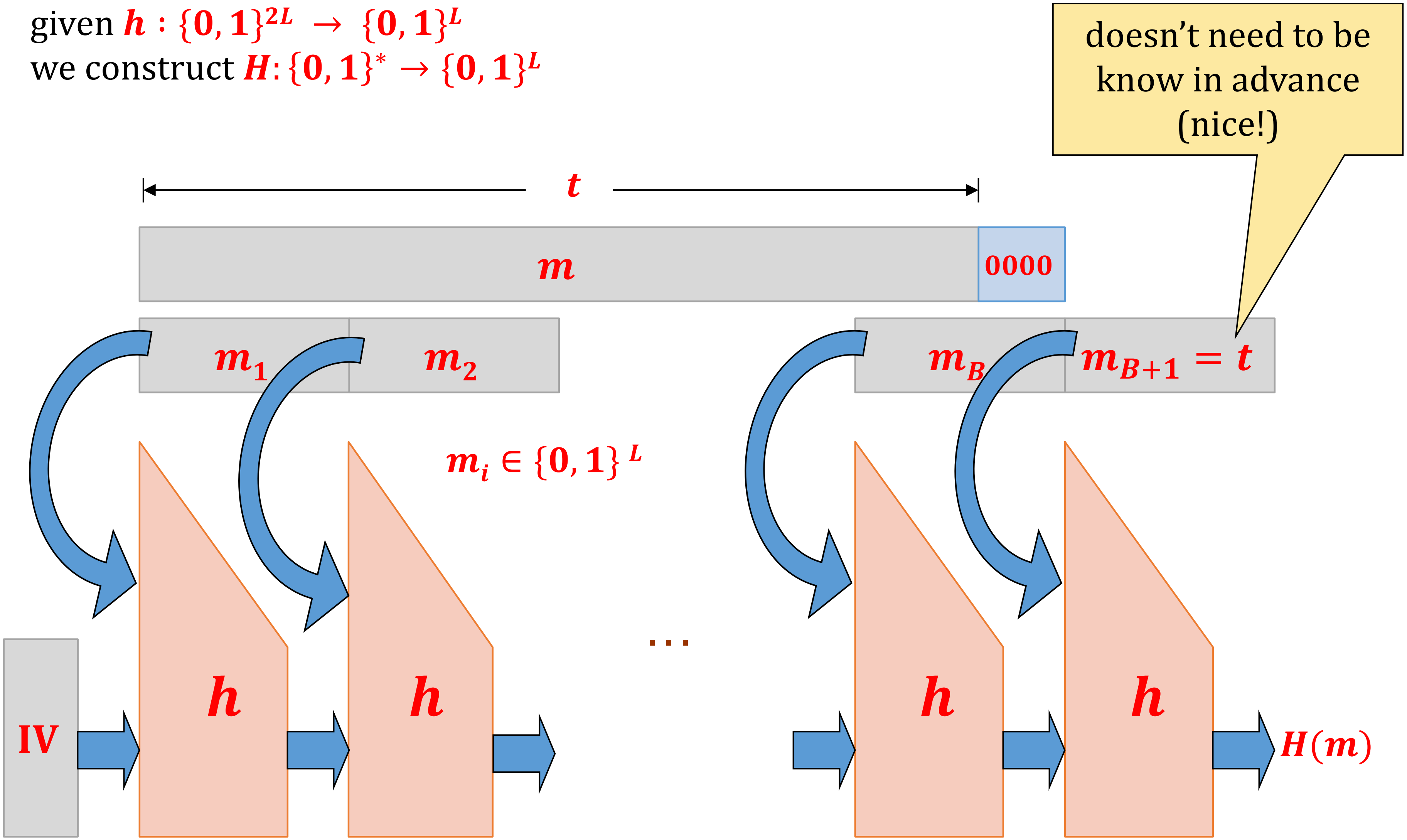


If we set $m' = m \,||\, 0000$ then $H(m') = H(m)$.

Solution: add a block encoding "t".

Merkle-Damgård transform

given $h : \{0,1\}^{2L} \rightarrow \{0,1\}^{L}$
we construct $H: \{0,1\}^{*} \rightarrow \{0,1\}^{L}$



This construction is secure

We would like to prove the following:

Theorem

If

$$\boldsymbol{h}\colon \{\mathbf{0},\mathbf{1}\}^{2L} \rightarrow \{\mathbf{0},\mathbf{1}\}^{L}$$

is a collision-resistant **compression** function

then

$$\boldsymbol{H}\colon \{\mathbf{0},\mathbf{1}\}^{*} \rightarrow \{\mathbf{0},\mathbf{1}\}^{L}$$

is a collision-resistant **hash** function.

But wait….

It doesn't make sense…

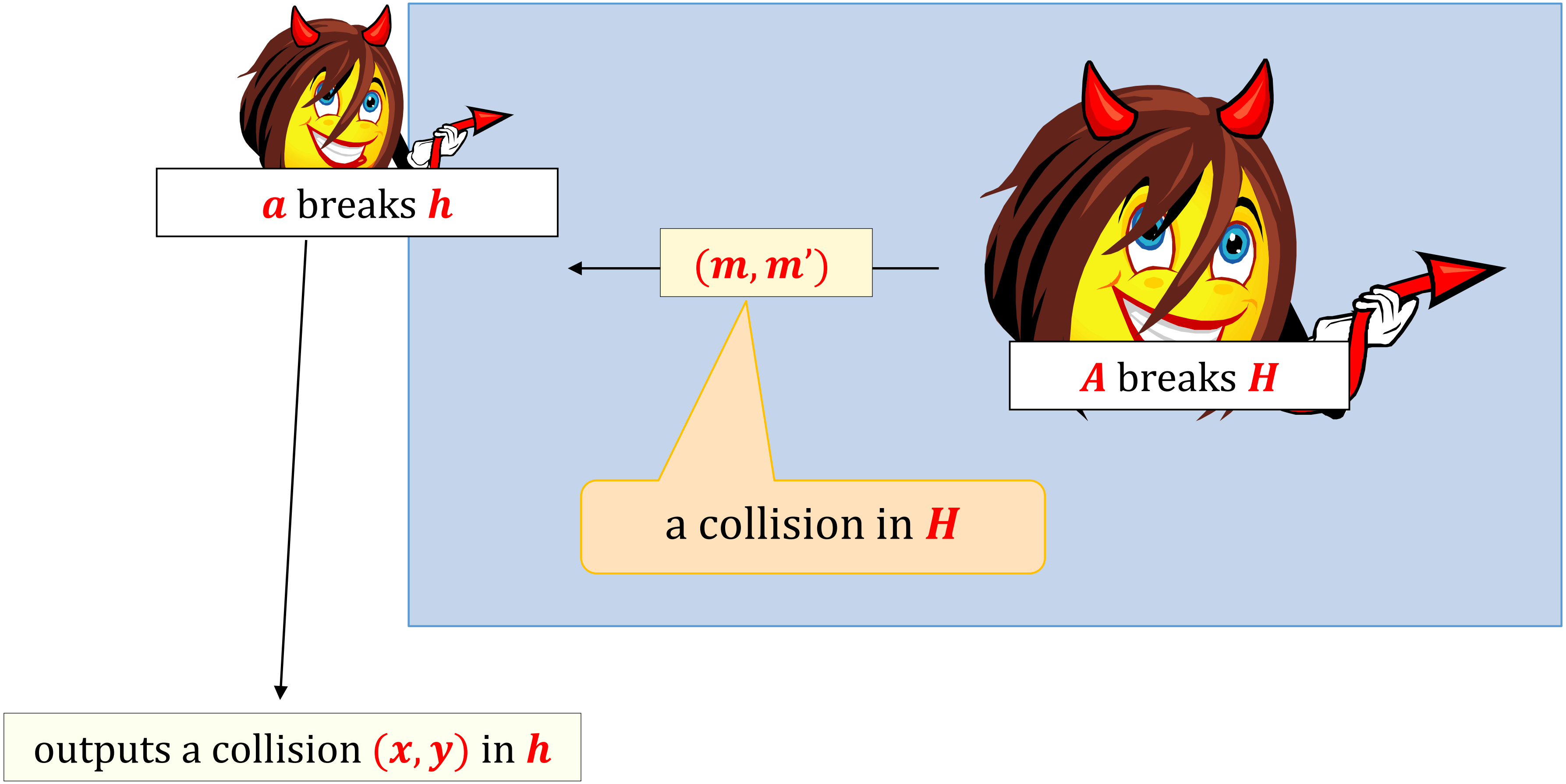
What to do?

To be formal, we would need to consider

families of functions
h and ***H***
indexed by key ***s***

Let's stay on the **informal level** and argue that:

"if one can find a collision in ***H*** then one can find a collision in ***h***"



a breaks h
(m, m')
A breaks H
a collision in H
outputs a collision (x, y) in h

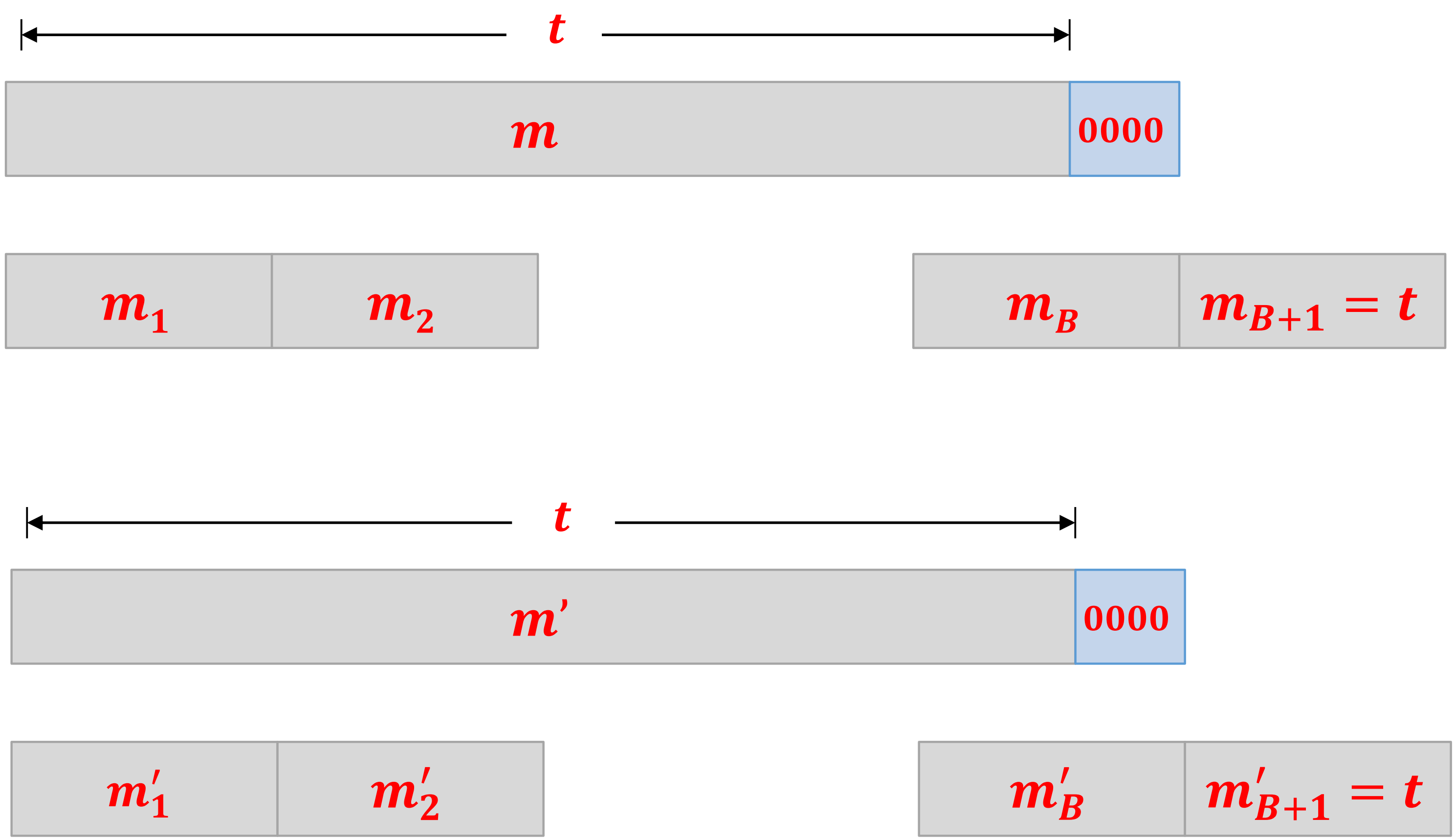
How to compute a collision $(\boldsymbol{x}, \boldsymbol{y})$ in $\boldsymbol{h}$ from a collision $(\boldsymbol{m}, \boldsymbol{m}')$ in $\boldsymbol{H}$?

We consider two cases:

1. $|\boldsymbol{m}| = |\boldsymbol{m}'|$

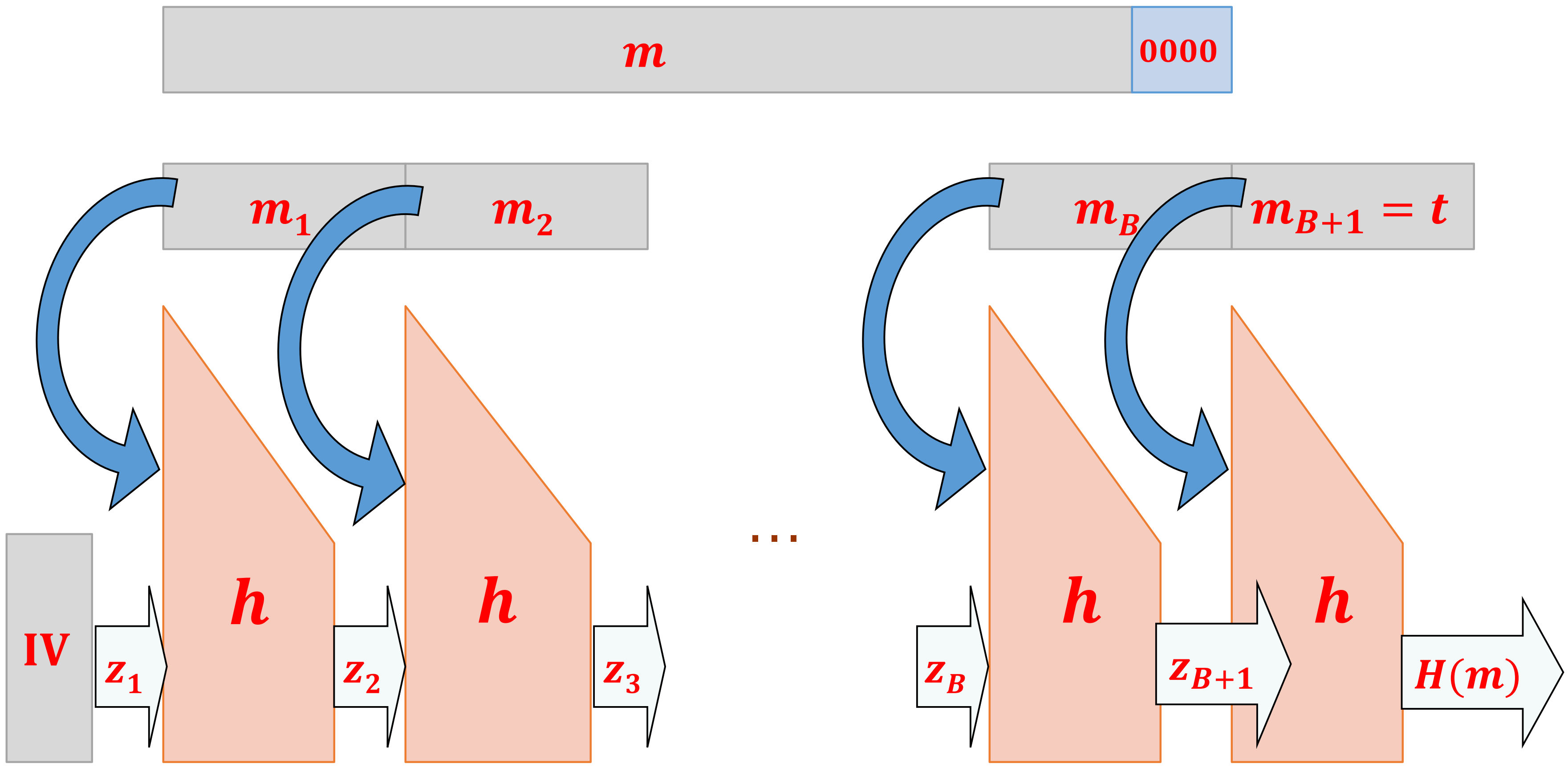
2. $|\boldsymbol{m}| \neq |\boldsymbol{m}'|$

Case 1: $|m| = |m'|$



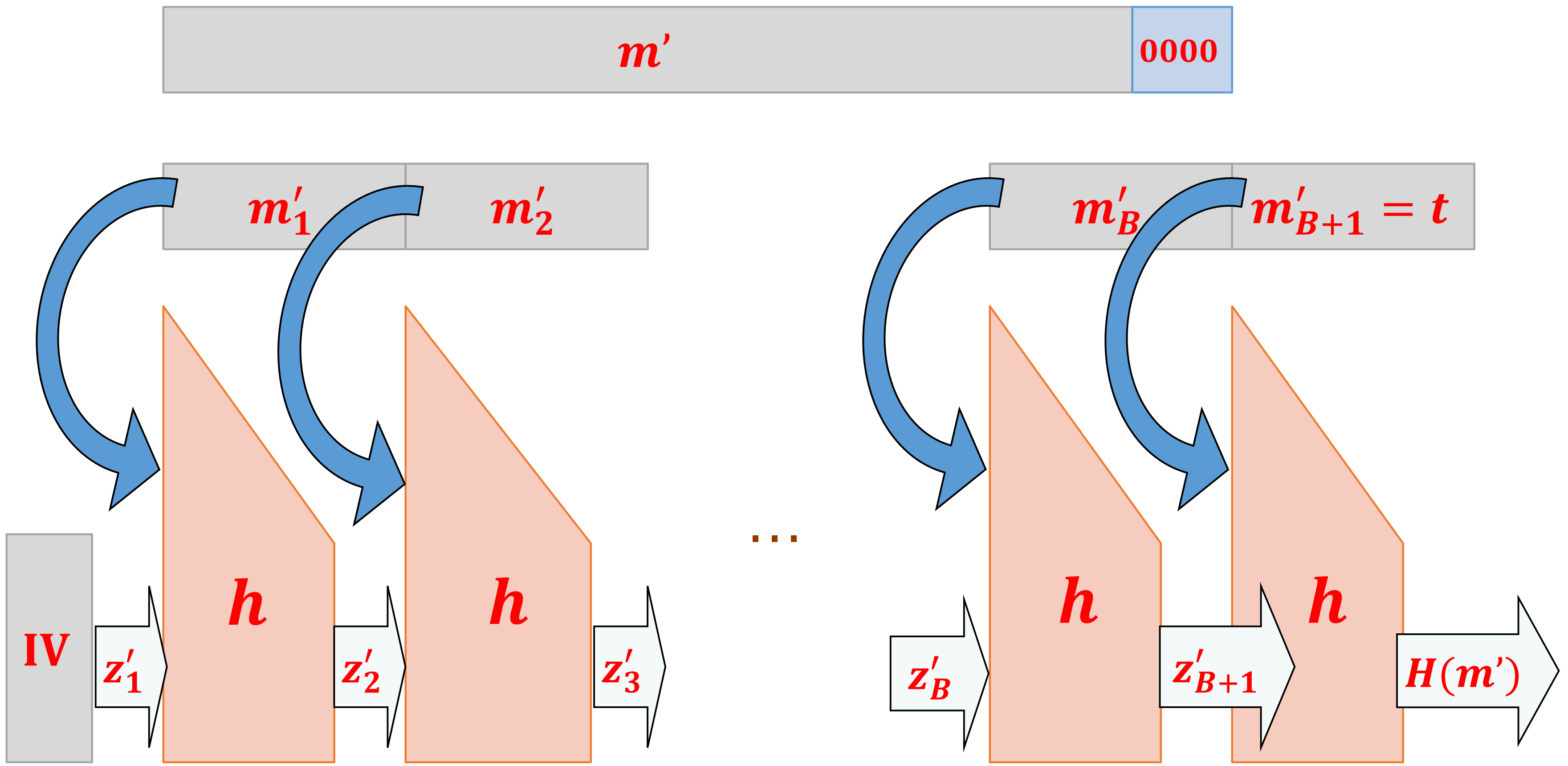
$|m| = |m'|$

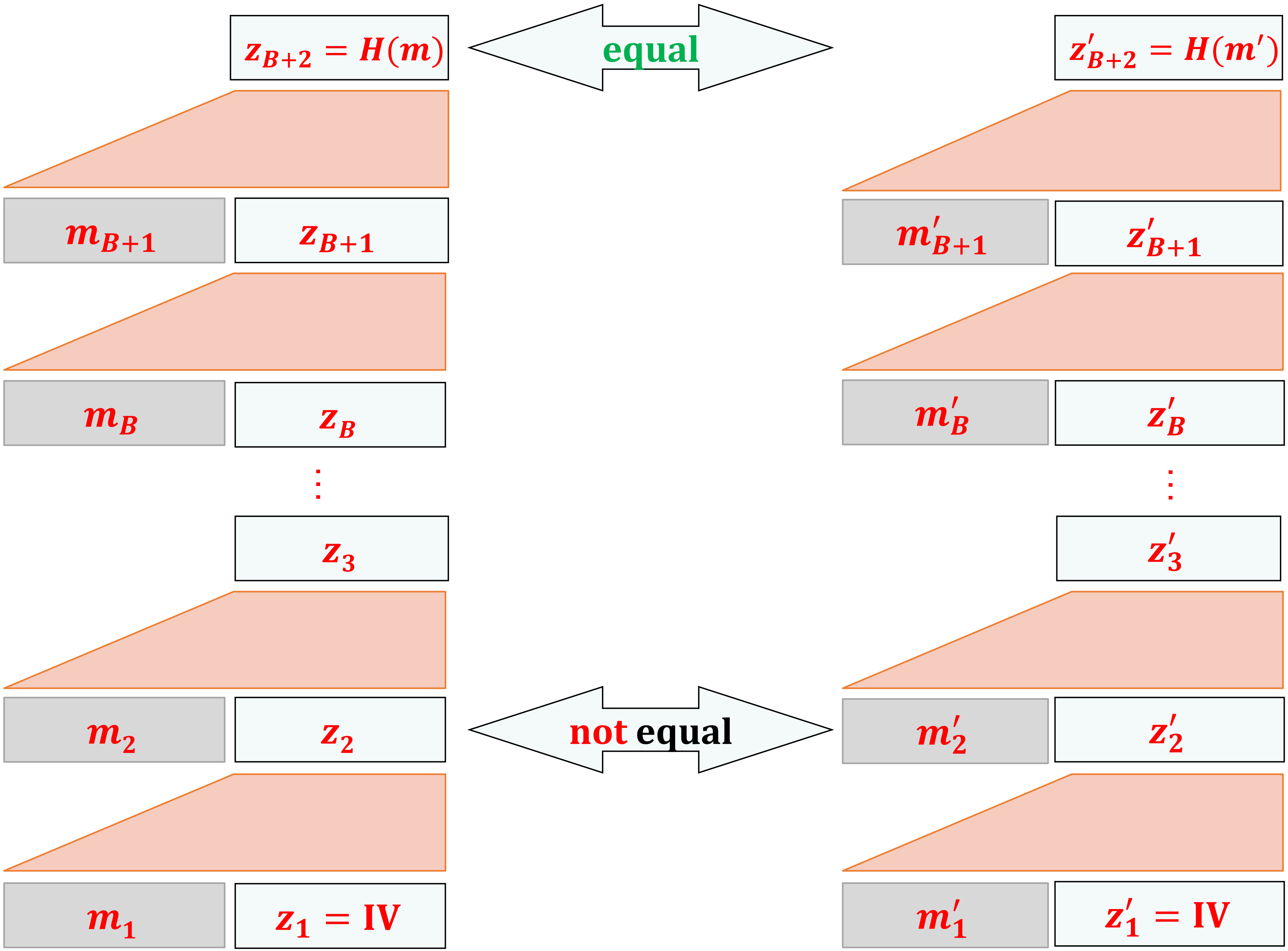
Some notation:

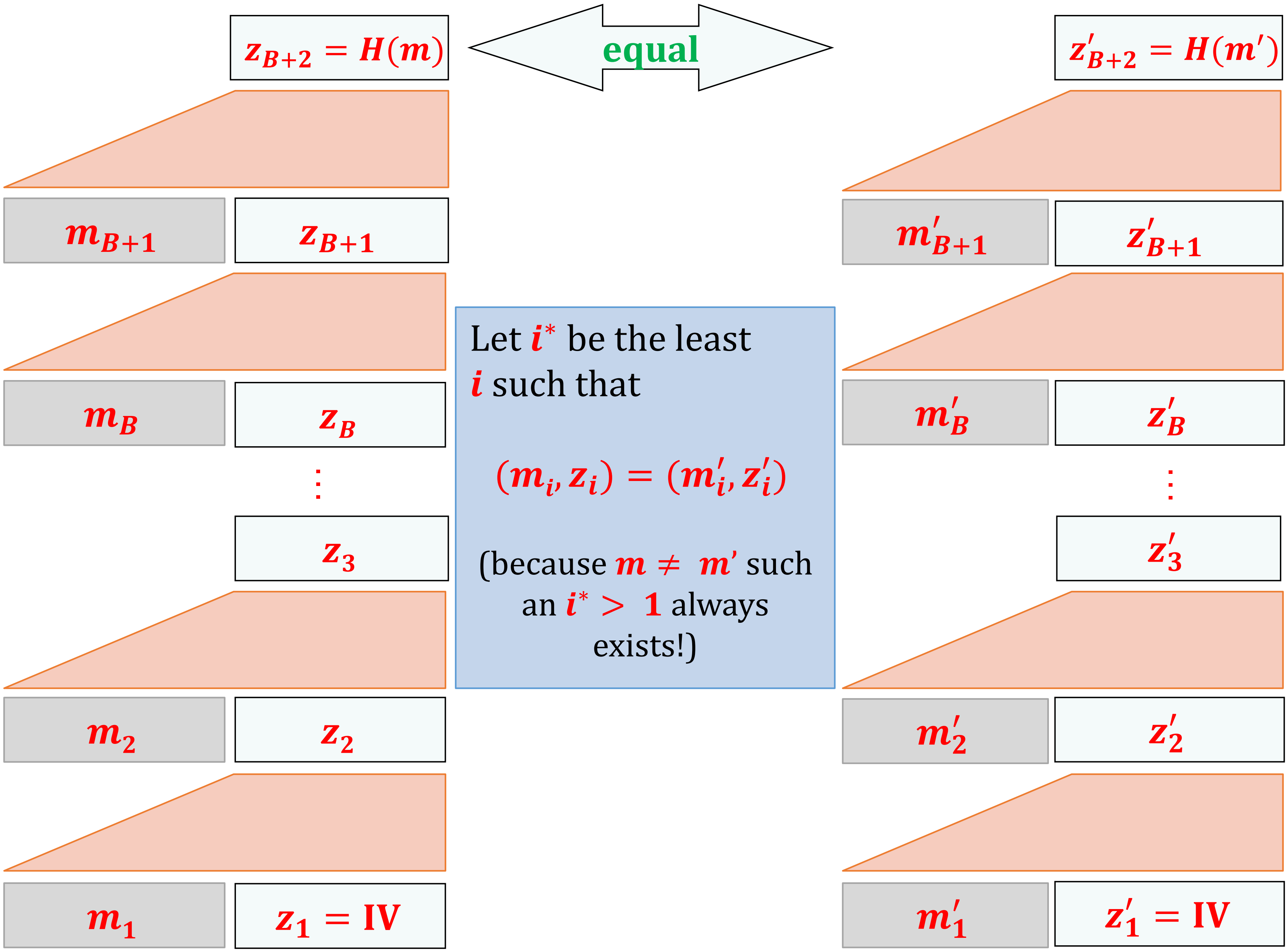


$|m| = |m'|$

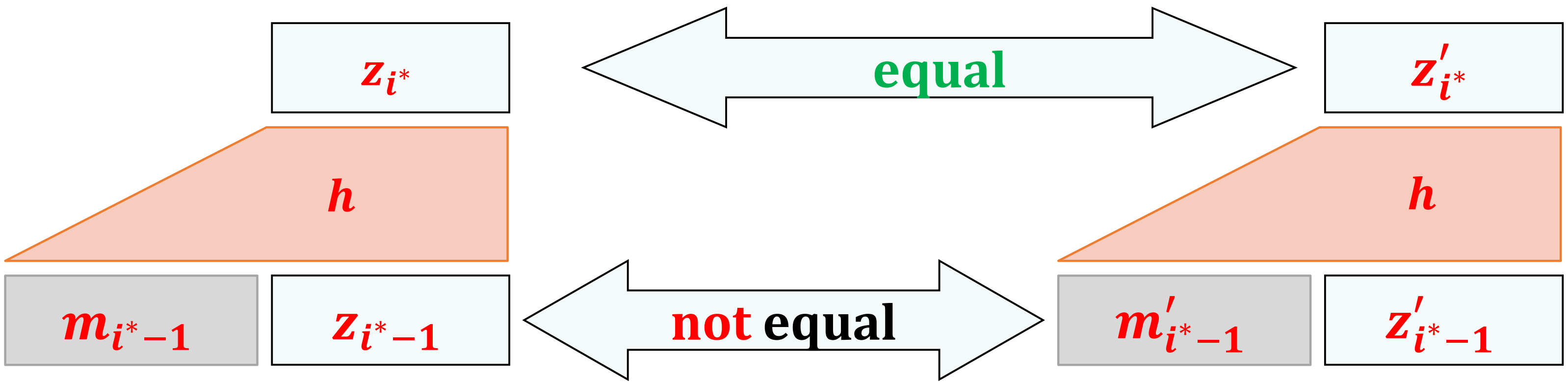
For m':



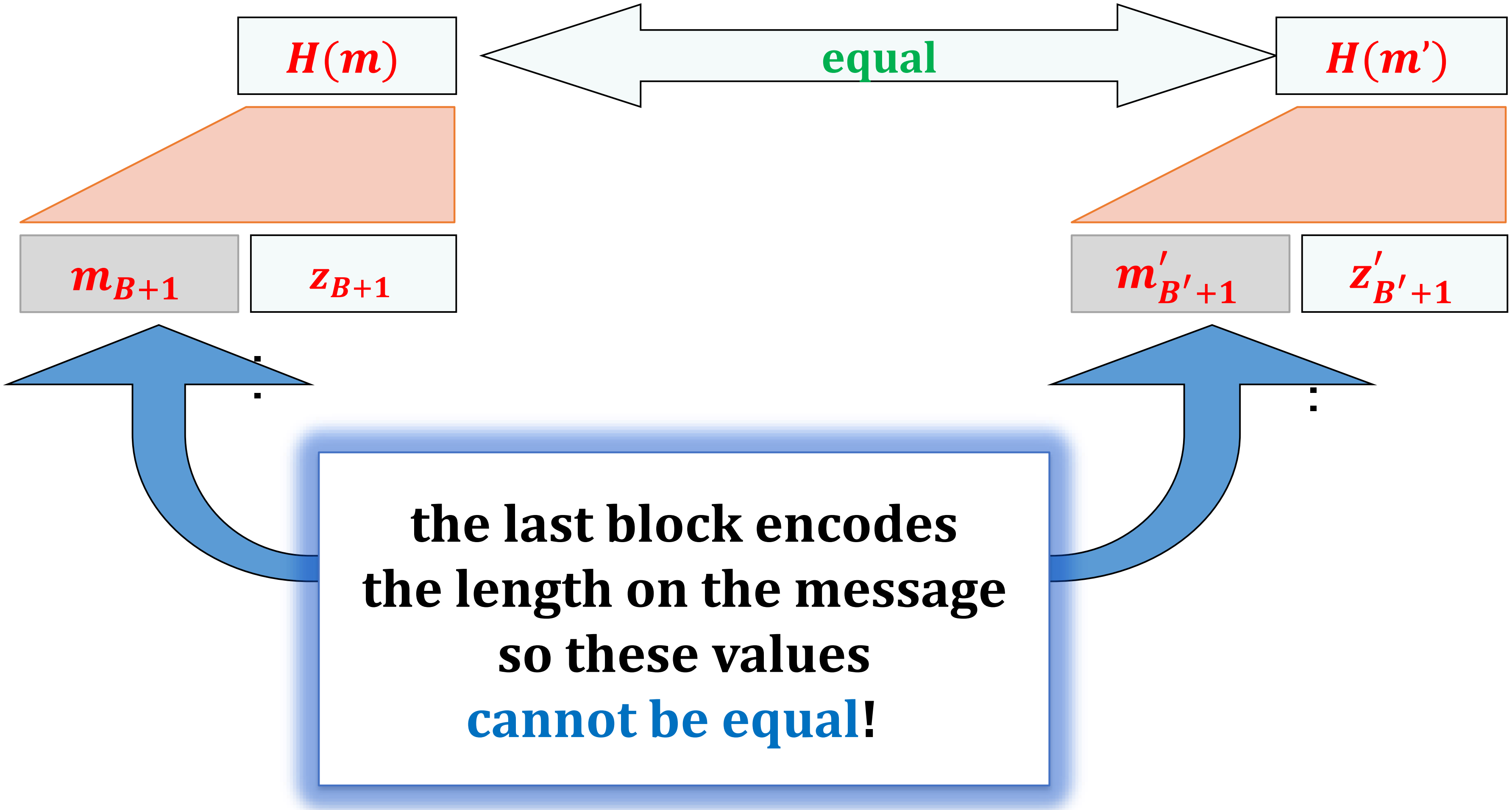

$z_{B+2} = H(m)$
equal
$z'_{B+2} = H(m')$
m_{B+1}
z_{B+1}
m'_{B+1}
z'_{B+1}
m_B
z_B
m'_B
z'_B
z_3
z'_3
m_2
z_2
not equal
m'_2
z'_2
m_1
$z_1 = \mathrm{IV}$
m'_1
$z'_1 = \mathrm{IV}$


$z_{B+2} = H(m)$
equal
$z'_{B+2} = H(m')$
m_{B+1}
z_{B+1}
m'_{B+1}
z'_{B+1}
Let i^* be the least i such that
$(m_i, z_i) = (m'_i, z'_i)$
(because $m \neq m'$ such an $i^* > 1$ always exists!)
m_B
z_B
m'_B
z'_B
z_3
z'_3
m_2
z_2
m'_2
z'_2
m_1
$z_1 = \mathrm{IV}$
m'_1
$z'_1 = \mathrm{IV}$

So, we have found a collision!



Case 2: $|m| \neq |m'|$



So, again we have found a collision!

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

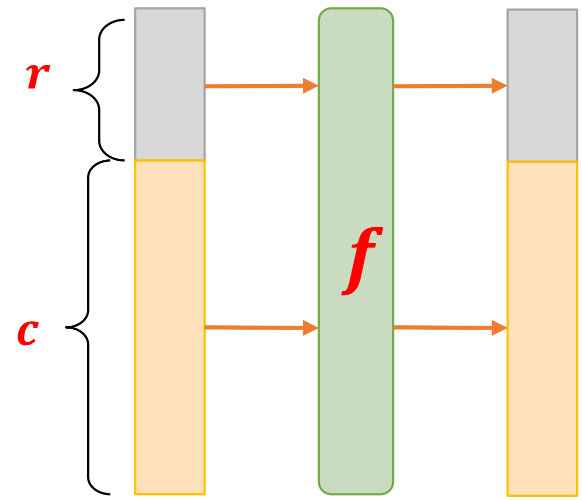
Sponge construction (used in Keccak)

main parameters:

c – "**capacity**"

r – "**rate**"

$b \coloneqq c + r$ – "**state width**"

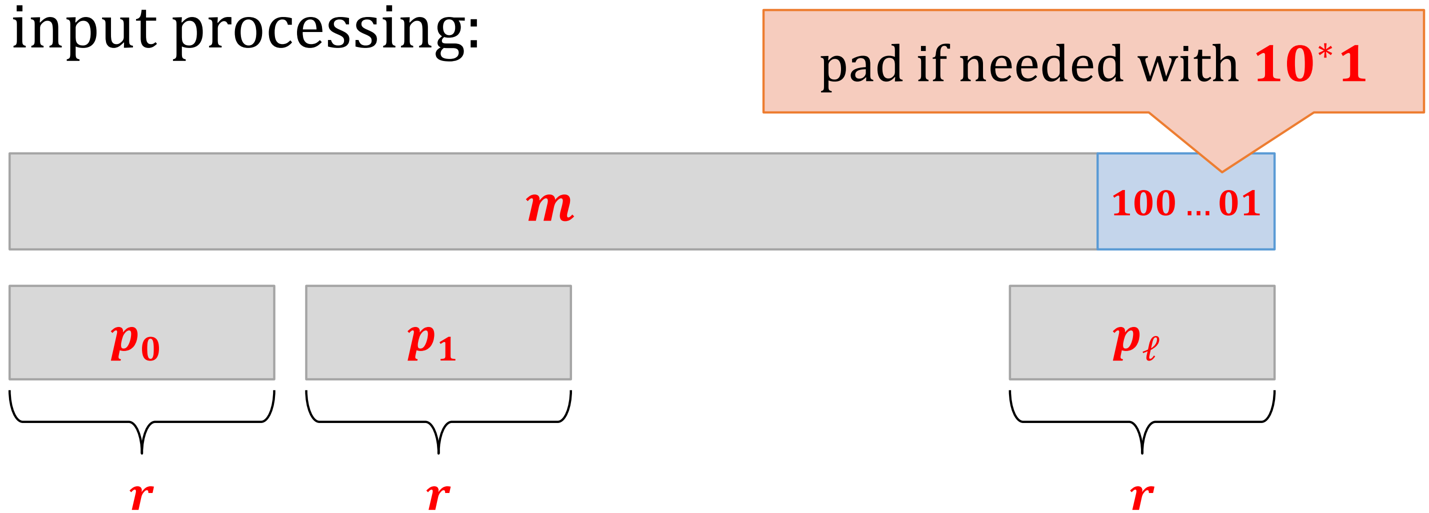


main ingredient:

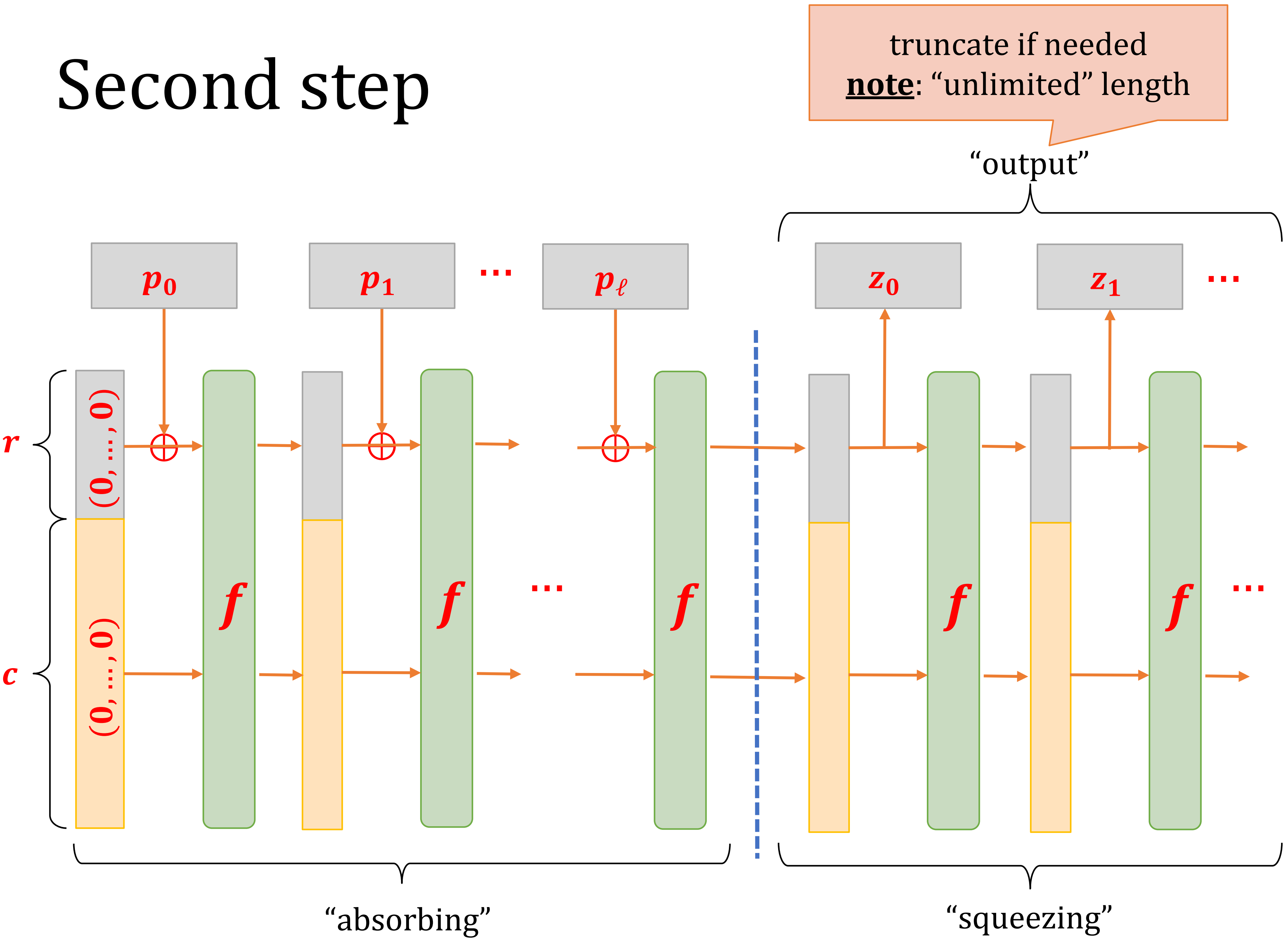
a function $f: \{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$

first step: input processing:

divide m as:



Second step


truncate if needed
note: “unlimited” length
“output”
p_0
p_1
p_ℓ
z_0
z_1
r
c
$(0,\dots,0)$
$(0,\dots,0)$
f
f
f
f
f
“absorbing”
“squeezing”

Plan

1. Introduction and definitions
2. Hash function design paradigms
   1. Merkle-Damgård transform
   2. Sponge construction
3. Real-life constructions
4. Other topics
   1. Merkle trees
   2. Practical randomness extraction and the random oracle model
   3. Password storage and Proofs of Work

Fact

There exists a generic attack on any hash function

$$\boldsymbol{H}: \{\mathbf{0}, \mathbf{1}\}^* \to \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$$

that finds a **collision with probability** $\frac{\mathbf{1}}{\mathbf{2}}$ and works in **time and space** $\boldsymbol{O}\left(\mathbf{2}^{\frac{\boldsymbol{n}}{\mathbf{2}}}\right)$.

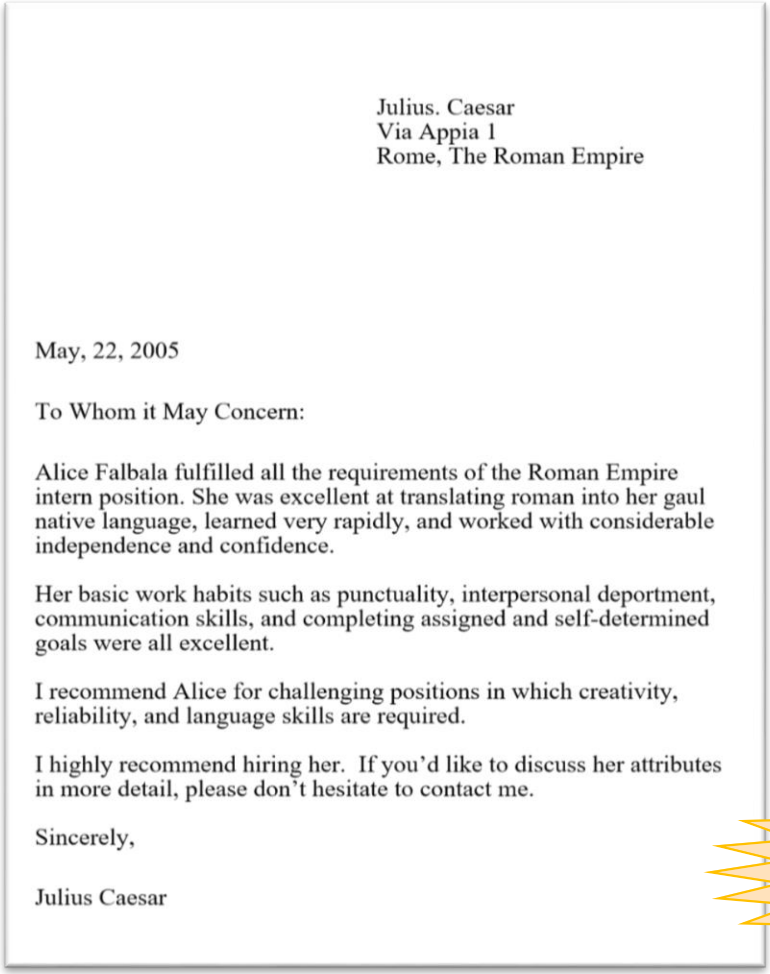
It's called a **birthday attack** and we will discuss it during the exercises.

Consequence: to achieve "$\boldsymbol{m}$ bits of security" one needs to set $\boldsymbol{n} = \mathbf{2} \cdot \boldsymbol{m}$.

MD5 (Message-Digest Algorithm 5)

- based on the **Merkle-Damgard paradigm**
- **output length**: **128 bits**,
- **designed** by **Rivest** in **1991**,
- in **1996**, **Dobbertin** found collisions in the compressing function of **MD5**,
- in **2004** a group of **Chinese mathematicians** designed a method for finding collisions in **MD5**,

June 2005: researchers at the Bochum University produce 2 postscript documents with the same **MD5** hash



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

To Whom it May Concern:

Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence.

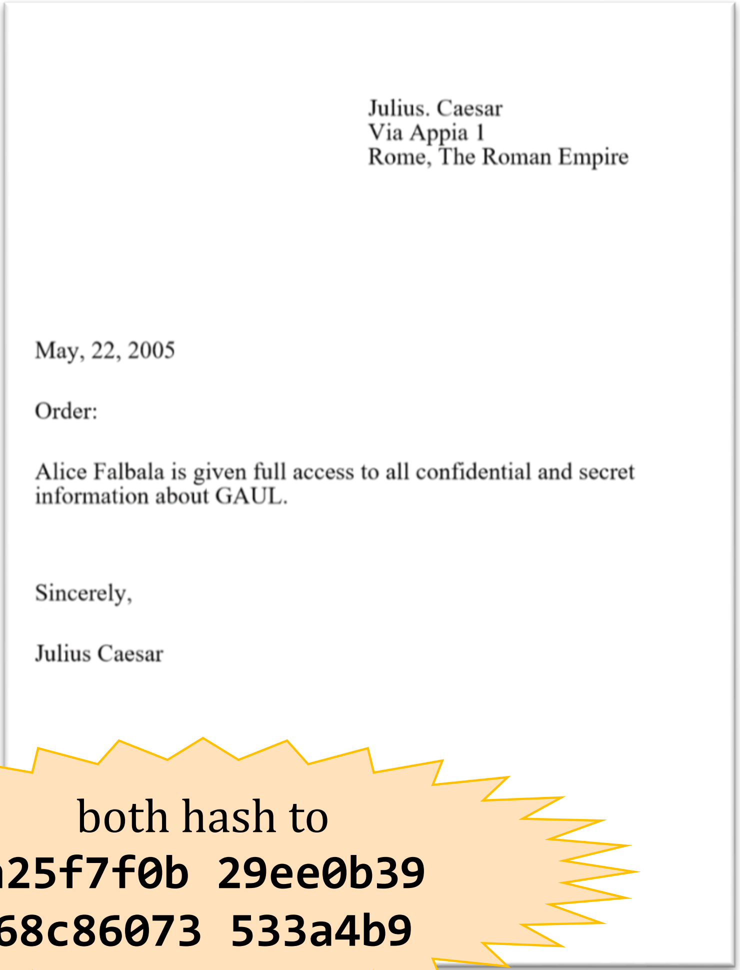
Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent.

I recommend Alice for challenging positions in which creativity, reliability, and language skills are required.

I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me.

Sincerely,

Julius Caesar



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

Order:

Alice Falbala is given full access to all confidential and secret information about GAUL.

Sincerely,

Julius Caesar

both hash to
a25f7f0b 29ee0b39
68c86073 533a4b9

This is done by exploiting the redundancy in postscript.

Colliding certificates

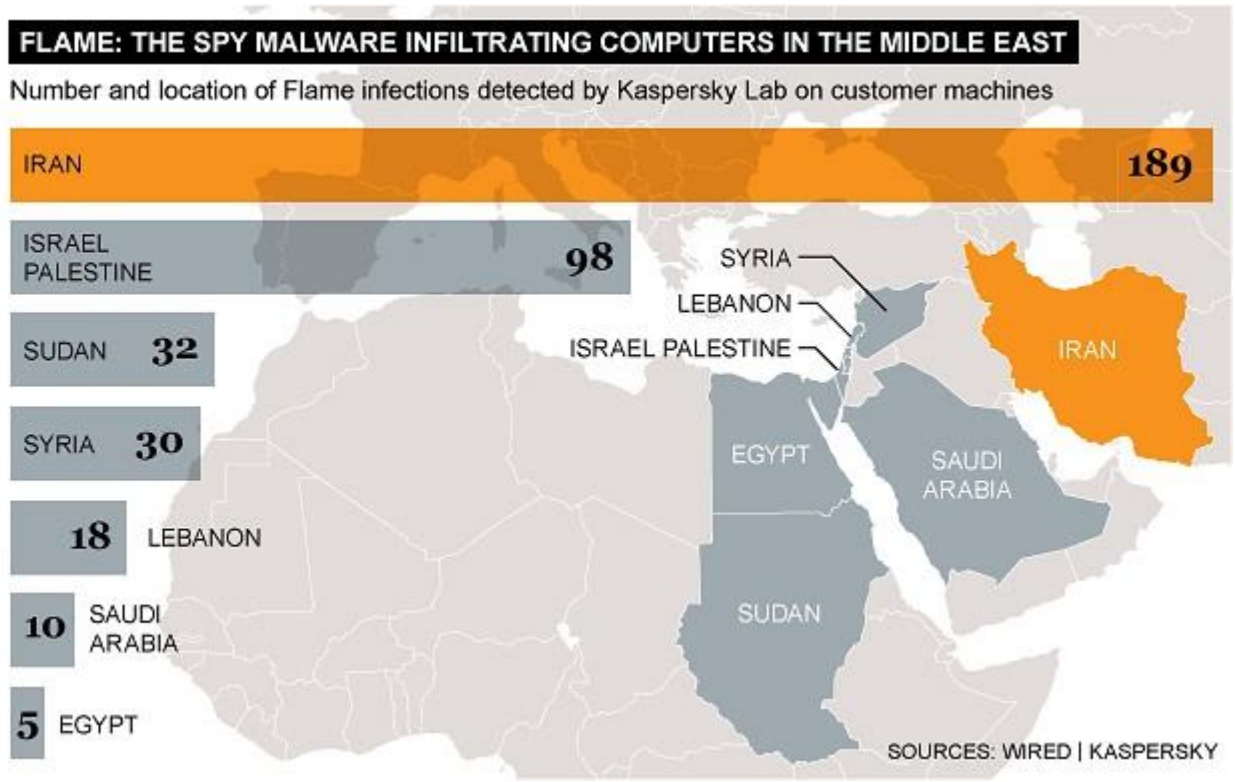
2005 and 2006: A. Lenstra, X. Wang, and B. de Weger found **X.509** certificates with different public keys and the same **MD5** hash.

(we will discuss the **X.509** certificates later)

Two certificates for **different names** ("Arjen K. Lenstra" and "Marc Stevens") **and different public keys**.

Flame mallware

attack on the Microsoft Windows Update Mechanism
exploiting **MD5** collision



SHA-1 (Secure Hash Algorithm)

- based on the **Merkle-Damgard paradigm**
- **output length**: **160 bits**,
- designed in **1993** by the **NSA**,
- in **2005** **Xiaoyun Wang, Andrew Yao and Frances Yao** presented an attack that runs in time 2^{63}.
- A recent attack (Oct 2015): **[Stevens, Karpman, and Peyrin: Freestart collision on full SHA-1]** a collision in the compression function in 2^{57} **SHA-1** evaluations.

The collision that was found

	Input 1
IV1	50 6b 01 78 ff 6d 18 90 20 22 91 fd 3a de 38 71 b2 c6 65 ea
M1	9d 44 38 28 a5 ea 3d f0 86 ea a0 fa 77 83 a7 36 33 24 48 4d af 70 2a aa a3 da b6 79 d8 a6 9e 2d 54 38 20 ed a7 ff fb 52 d3 ff 49 3f c3 ff 55 1e fb ff d9 7f 55 fe ee f2 08 5a f3 12 08 86 88 a9
SHA1_compression_function (**IV1**,**M1**)	f0 20 48 6f 07 1b f1 10 53 54 7a 86 f4 a7 15 3b 3c 95 0f 4b

	Input 2
IV2	50 6b 01 78 ff 6d 18 91 a0 22 91 fd 3a de 38 71 b2 c6 65 ea
M2	3f 44 38 38 81 ea 3d ec a0 ea a0 ee 51 83 a7 2c 33 24 48 5d ab 70 2a b6 6f da b6 6d d4 a6 9e 2f 94 38 20 fd 13 ff fb 4e ef ff 49 3b 7f ff 55 04 db ff d9 6f 71 fe ee ee e4 5a f3 06 04 86 88 ab
SHA1_compression_function (**IV2**,**M2**)	f0 20 48 6f 07 1b f1 10 53 54 7a 86 f4 a7 15 3b 3c 95 0f 4b

Hardware used in **[Stevens, Karpman, and Peyrin: Freestart collision on full SHA-1]**:

*"We have computed the SHA-1 freestart collision on **Kraken**, our 64-GPU cluster. More precisely Kraken is composed of 16 nodes, each node being made of simple, cheap and widely available hardware: 4 GTX-970 GPUs, 1 Haswell i5-4460 processor and 16GB of RAM."*

An estimation

[Stevens, Karpman, and Peyrin: Freestart collision on full SHA-1]

“Concretely, we estimate the SHA-1 collision cost today (i.e., Fall 2015) between 75K$ and 120K$ renting Amazon EC2 cloud computing over a few months.”

Reaction of the industry

Microsoft may block SHA1 certificates sooner than expected

Encrypted sites running old certificates will be inaccessible from modern browsers.

By Zack Whittaker for Zero Day | November 9, 2015 -- 13:16 GMT (13:16 GMT) | Topic: Security

Mozilla Security Blog

OCT 20 2015

Continuing to Phase Out SHA-1 Certificates

Richard Barnes

In our previous blog post about phasing out certificates with SHA-1 based signature algorithms, we said that we planned to take a few actions with regard to SHA-1 certificates:

1. Add a security warning to the Web Console to remind developers that they should not be using a SHA-1 based certificates
2. Show the "Untrusted Connection" error whenever a SHA-1 certificate issued after January 1, 2016, is encountered in Firefox
3. Show the "Untrusted Connection" error whenever a SHA-1 certificate is encountered in Firefox after January 1, 2017

A new hash algorithm: SHA-3

Selected by the **National Institute of Standards and Technology (NIST)** in an open competition.

5 finalists: **BLAKE, Grøstl, JH, Keccak, Skein**.

Winner (2012): **Keccak**.

SHA-3: Keccak

authors: Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche

output lengths: **224**, **256**, **384**, **512**, or unbounded

speed: **12.5** cycles per byte on Core 2

Not based on the **Merkle-Damgard paradigm**.

Instead: it uses the **sponge construction**.

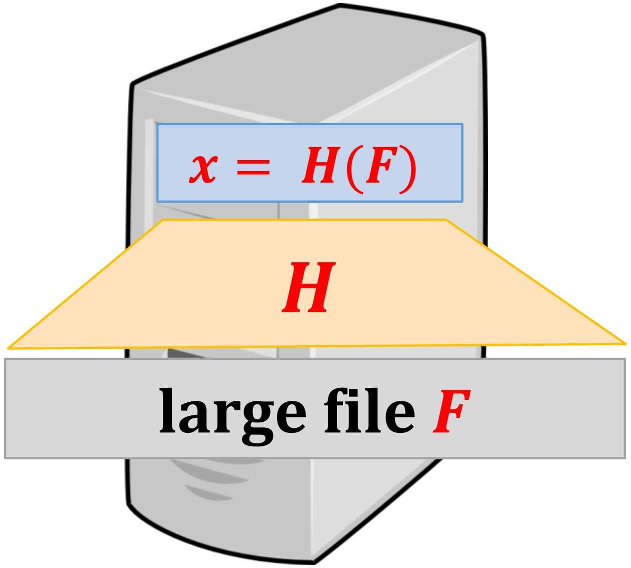
Standardized Keccak's parameters for fixed output length

state width *b*	**rate *r***	**capacity *c***	**output length**
1600	1344	256	224
1600	1344	256	256
1600	1088	512	384
1600	1088	512	512

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

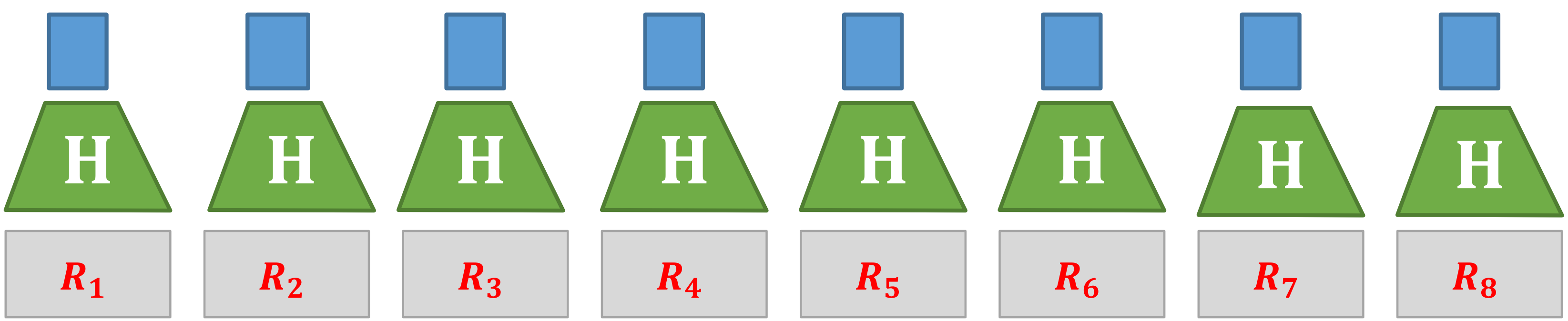
Consider again file fingerprinting



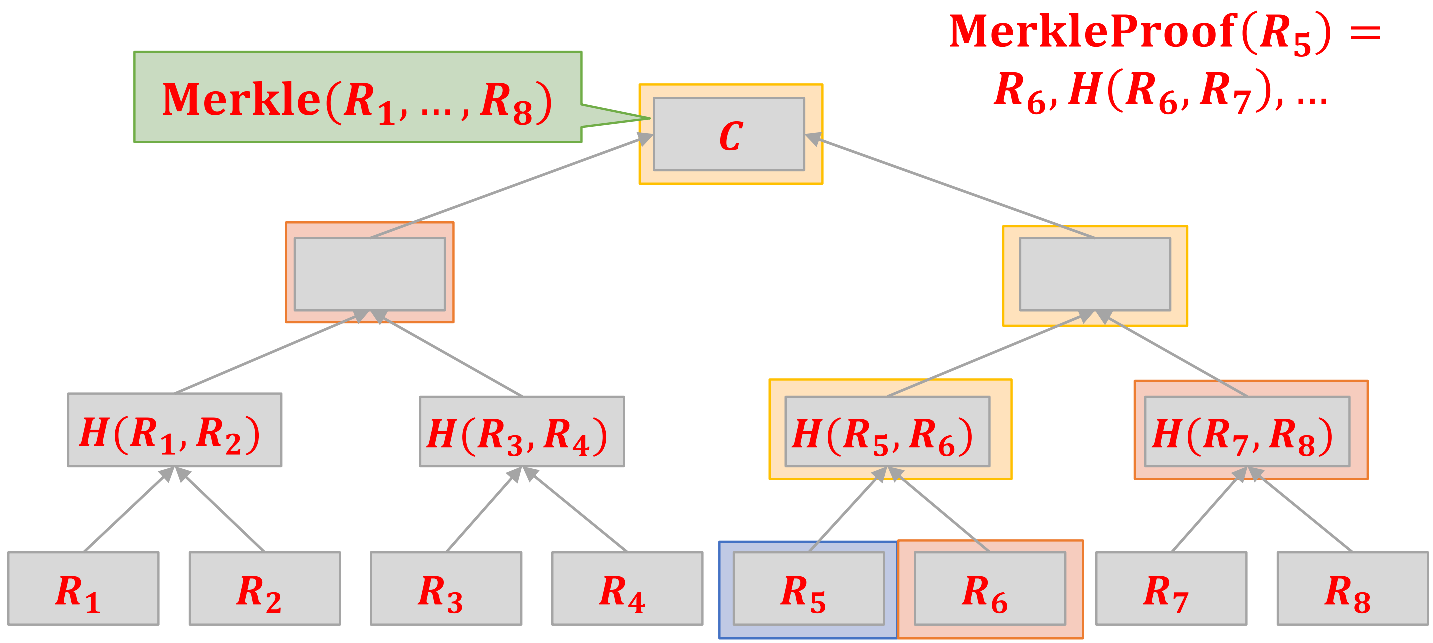
A question

Suppose a file $\boldsymbol{F}$ consists of many smaller blocks $\boldsymbol{R_1, \ldots, R_n}$, and the user may want to access only one of them. How to "fingerprint $\boldsymbol{F}$"?

Naive solution: fingerprint each of them independently.



Better solution: construct a **Merkle tree:**



Recall: Merkle trees allow to efficiently prove that each block R_i was included into the hash C.
This is done by sending $\mathbf{MerkleProof}(R_i)$.
Easy to see: if H is collision resistant then so is **Merkle**.

File sharing

BitTorrent, **Gnutella**, **Gnutella2**, and **Direct Connect P2P**... - a variant of the idea from the previous slides:

- files in the peer-to-peer networks are **identified by their hashes**
- each file consists of "**pieces**"
- the users download the pieces from each other.
- some of them use **Merkle trees.**

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

How the outputs of hash functions look in real life?

```
C:\> echo -n `Wydział Matematyki, Informatyki i
Mechaniki` | openssl sha1
30428440c00bd45d2e2fd93ed980fbd8aa063428

C:\> echo -n `Wydział Matematyki, Informtyki i
Mechaniki` | openssl sha1
9937fe966d988e8163fe07f6a1dbd9caf624e1c8

C:\> echo -n `Wydział Matematyki Informatyki i
Mechaniki` | openssl sha1
456b370c5afe5f45c0af4a6290d02f6d2f557381
```

Observation: the outputs on different inputs are "unrelated" and "completely random".

we will formalize this property in a moment

Example of how this property is used: deriving “uniformly random keys” from “non-uniform randomness"

shorter “uniformly random” $\boldsymbol{H(m)}$

a hash function

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

user generated randomness $\boldsymbol{X}$ (key strokes, mouse movements, passwords, etc.)

Example: password-based encryption

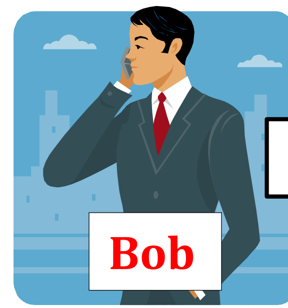
$\boldsymbol{H}$ – hash function
$(\boldsymbol{E}, \boldsymbol{D})$ – encryption scheme

message $\boldsymbol{m}$

Alice

shared password $\boldsymbol{\pi}$

$\boldsymbol{c} = \boldsymbol{E}(\boldsymbol{H}(\boldsymbol{\pi}), \boldsymbol{m})$



Bob

$\boldsymbol{m} = \boldsymbol{D}(\boldsymbol{H}(\boldsymbol{\pi}), \boldsymbol{c})$

shared password $\boldsymbol{\pi}$

Warning:
there exist much better solutions for this problem

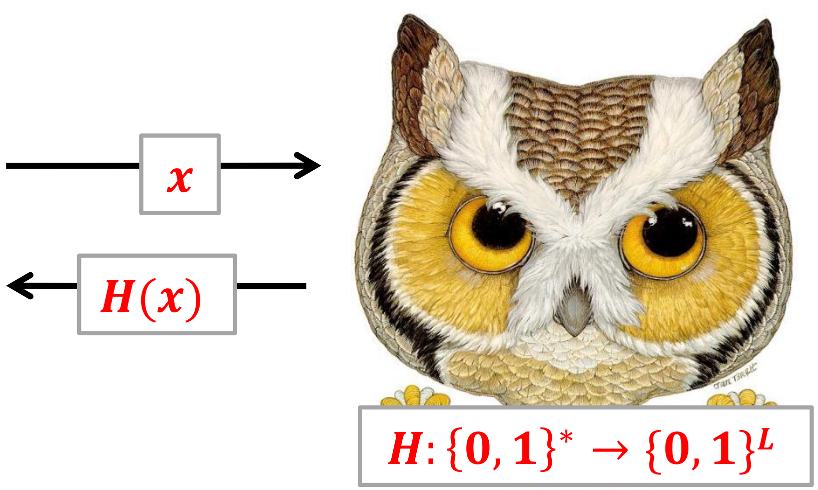
Informally:
The only thing that Eve can do is to examine all possible passwords .

Random oracle model

[Fiat, Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. 1986]

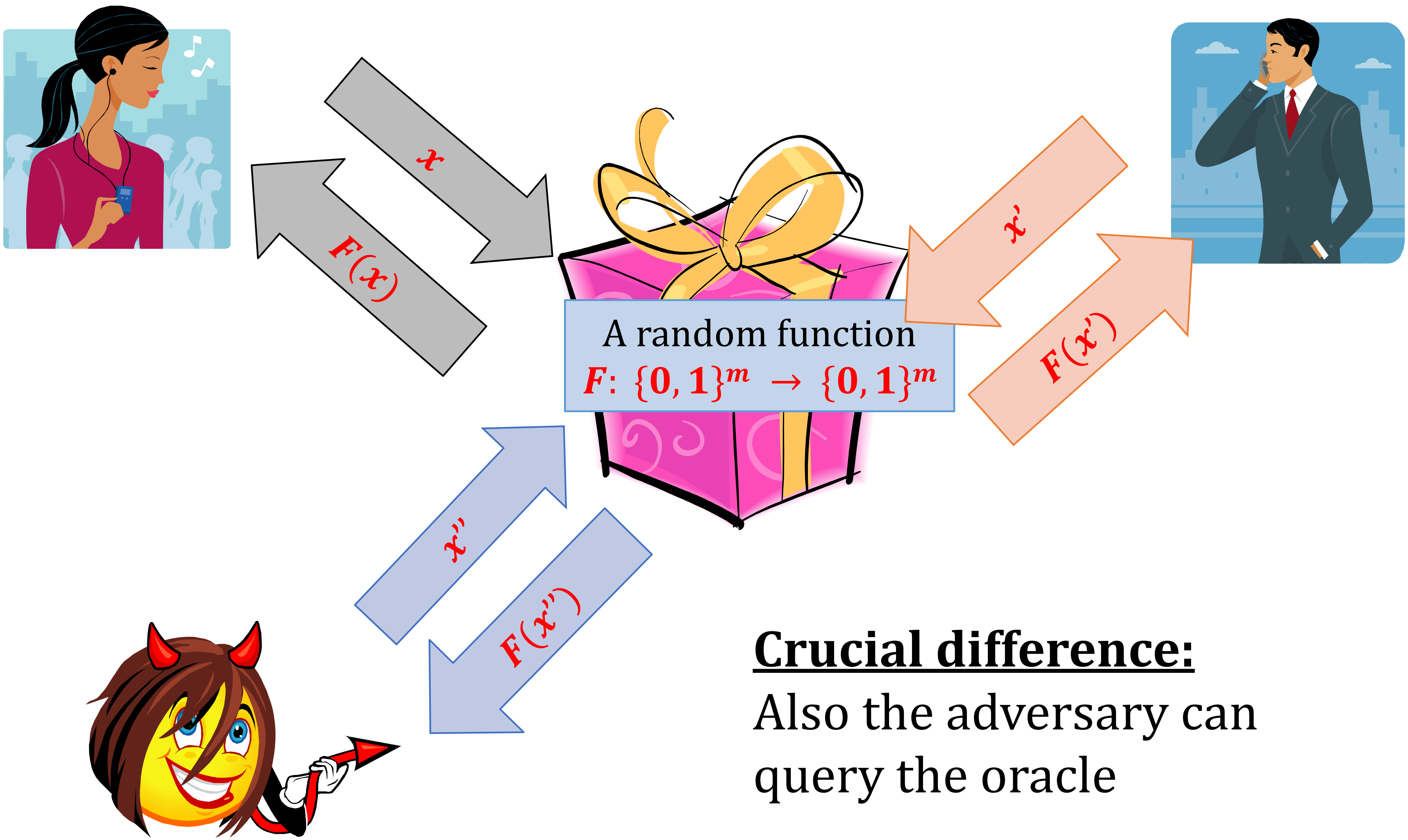
[Bellare, Rogaway: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols, 1993]

Idea: model the hash function as a **random oracle**.



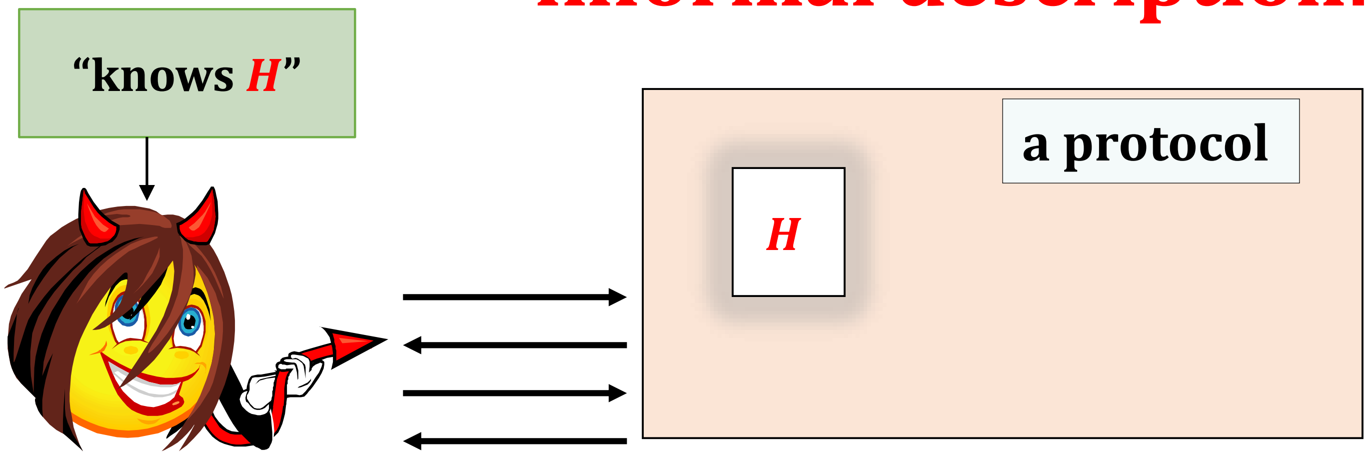
a completely random function

Remember the pseudorandom functions?

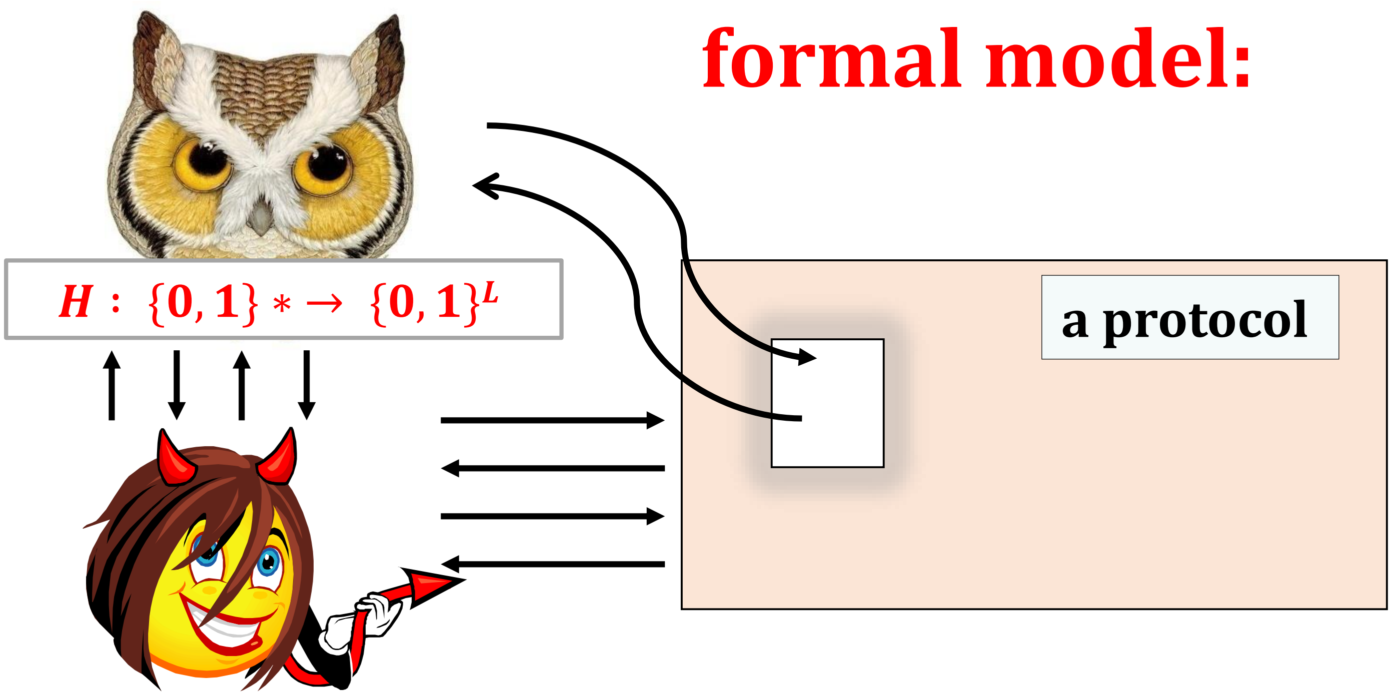


<u>Crucial difference:</u>
Also the adversary can query the oracle

informal description:



formal model:



Every call to ***H*** is replaced with a query to the oracle.

also the adversary is allowed to query the oracle.

How would we use it in the proof?

shorter "uniformly random" $H(X)$

a hash function

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

user generated randomness X

As long as the adversary never queried the oracle on X the value $H(X)$ "looks completely random to him".

Criticism of the Random Oracle Model

[Canetti, Goldreich, Halevi: **The random oracle methodology, revisited**. 1998]

There exists a signature scheme that is

- **secure** in **ROM**

  but

- is **not secure** if the random oracle is replaced with **any** real hash function.

This example is **very artificial**. No "realistic" example of this type is know.

Terminology

Model without the random oracles:
- "**plain model**"
- "**cryptographic model**"

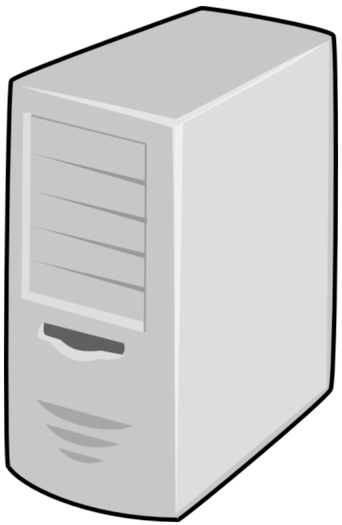
Random Oracle Model is also called:
the "**Random Oracle Heuristic**".

Common view: a proof in **ROM** is better than nothing.

Plan

1. Introduction and definitions
2. Hash function design paradigms
    1. Merkle-Damgård transform
    2. Sponge construction
3. Real-life constructions
4. Other topics
    1. Merkle trees
    2. Practical randomness extraction and the random oracle model
    3. Password storage and Proofs of Work

Password storage



Simple idea: instead of storing user's passwords π in plaintext store their hashes.

Better: "salted hashes" $(s, H(s, \pi))$

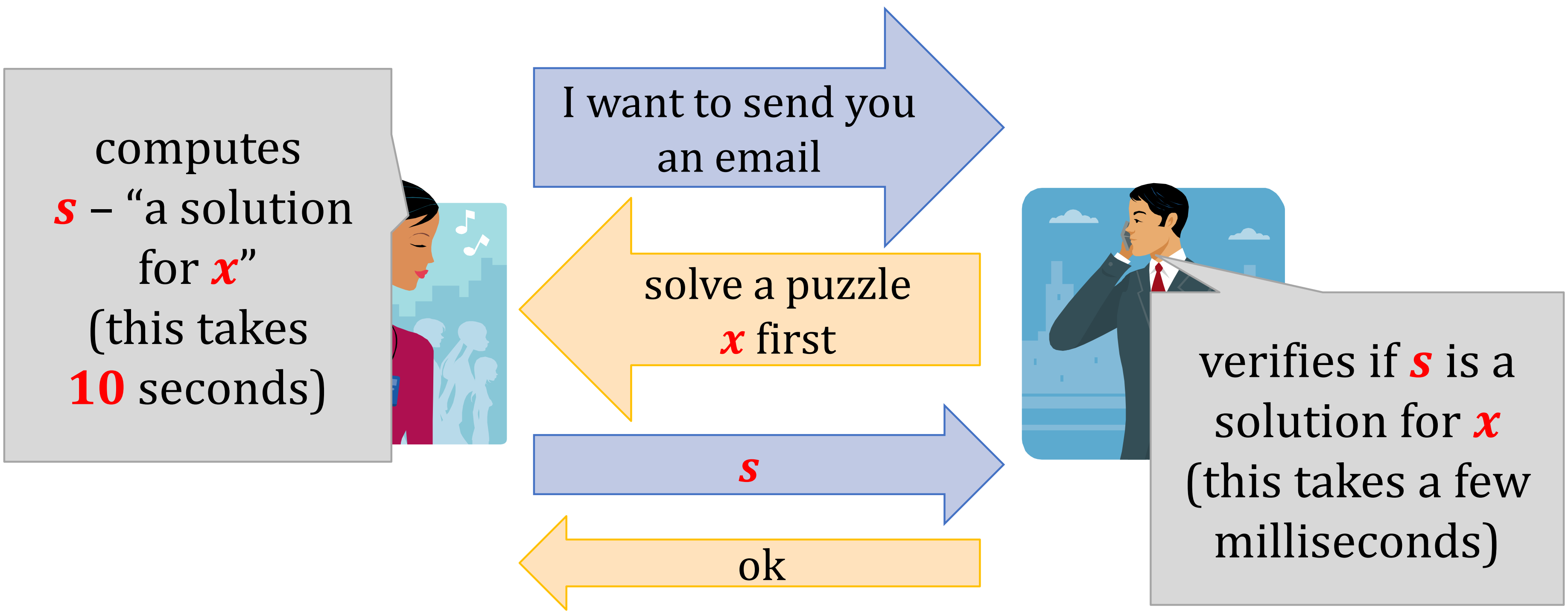
advantages:

1. makes "precomputation attacks" harder (we will discuss these attacks on the exercises)
2. if two users have the same password then the stored values are different.

Proofs of work

Introduced by **Dwork and Naor** [Crypto 1992] as a countermeasure against spam.

Basic idea: Force users to do some computational work: solve a **moderately difficult** "puzzle" (checking correctness of the solution has to be fast).



A simple hash-based PoW

H – a hash function whose computation takes time **TIME(H)**

random **x**

s

Prover
finds s such that
$H(s, x)$ starts with n zeros (in binary)

salt	"hardness parameter

Verifier
checks if
$H(s, x)$ starts with **n** zeros

(in **ROM**) takes expected time $2^n \cdot$ **TIME(H)**

takes time **TIME(H)**

This **PoW** is used in Bitcoin.

Problem

Computing typical hash functions is much faster when done in **parallel** and in **hardware** (this can give advantage to a powerful adversary).

For example “**Bitcoin mining**” is done almost entirely on ASICs

AntMiner S7	Avalon6	SP20 Jackson
		
Advertised Capacity: 4.73 Th/s	**Advertised Capacity:** 3.5 Th/s	**Advertised Capacity:** 1.3-1.7 Th/s
Power Efficiency: 0.25 W/Gh	**Power Efficiency:** 0.29 W/Gh	**Power Efficiency:** 0.65 W/Gh
Weight: 8.8 pounds	**Weight:** 9.5 pounds	**Weight:** 20 pounds
Guide: Yes	**Guide:** No	**Guide:** Yes
Price: $479.95	**Price:** $499.95	**Price:** $248.99

Idea for a solution

Design hash functions whose computation **needs to access memory a lot of times**, so it's hard to implement it efficiently in hardware

Example: **scrypt** hash function introduced in:

> **Colin Percival, *Stronger Key Derivation via Sequential Memory-Hard Functions*, 2009**.

Used in **Litecoin**

Has one practical drawback: it's access pattern is **data-dependent** (hence: it reveals the input).

bad for the side-channel resilience

How **scrypt** works?

computing **scrypt(X)**

init phase: fill-in at table of length $\boldsymbol{N}$ with pseudorandom expansion of $\boldsymbol{X}$.

$$V_0 = X \rightarrow V_1 = H(X) \rightarrow V_2 = H(V_2) \rightarrow \cdots \rightarrow V_{N-1} = H(V_{N-2})$$

result (for N = **10**):

V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9

second phase: compute the output by accessing the table
”pseudorandomly”

$Z := \mathrm{H}(\mathrm{V}_{\mathrm{N}-1})$
for $i = 0$ **to** $N - 1$ **do**
    $j := X \bmod N$
    $X := H(X \oplus V_j)$
output X

What is known about **scrypt**?

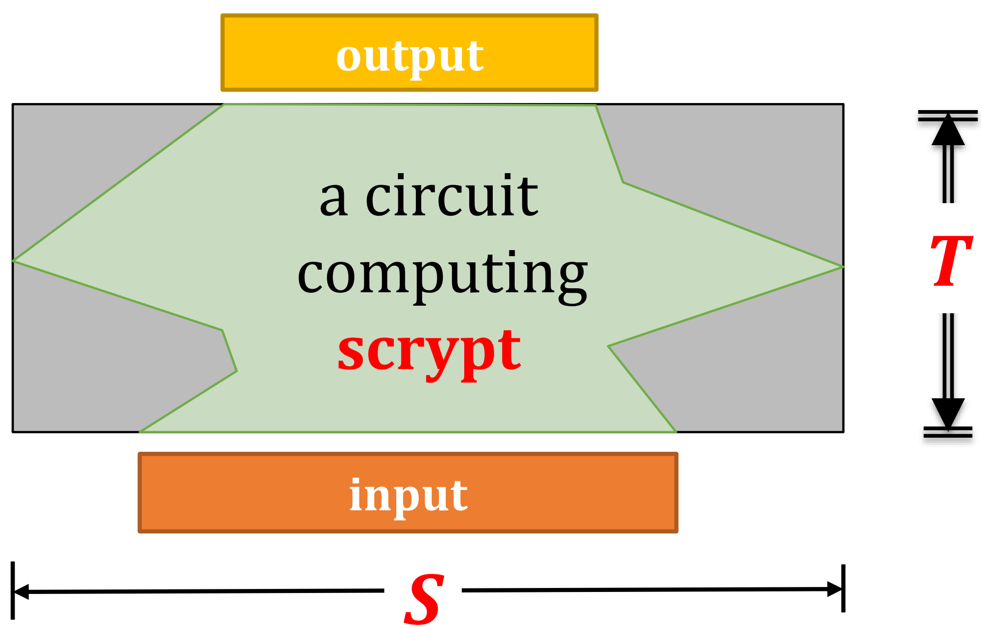
[Percival, 2009]:

- it can be computed in time $\boldsymbol{O}(\mathbf{N})$,
- to compute it one needs time $\boldsymbol{T}$ and space $\boldsymbol{S}$ such that

$$\boldsymbol{S} \times \boldsymbol{T} = \boldsymbol{\Omega}\left(\boldsymbol{N}^{2}\right)$$

this holds even on a parallel machine.

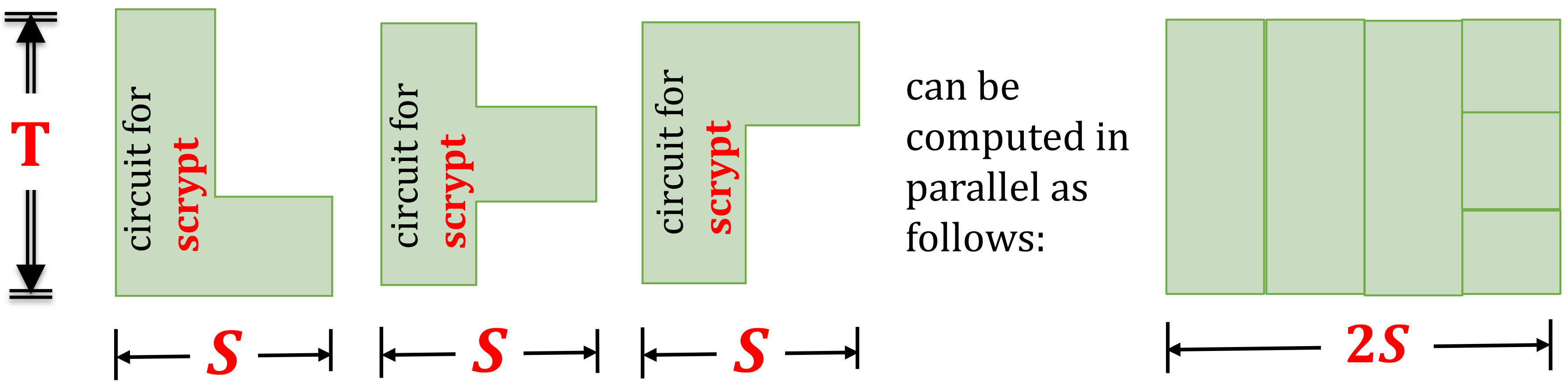
Pictorially:



An observation

[Alwen, Serbinenko, STOC'15]: this definition is **not strong enough**.

The adversary that wants to compute scrypt in parallel can "amortize space". **Example**:

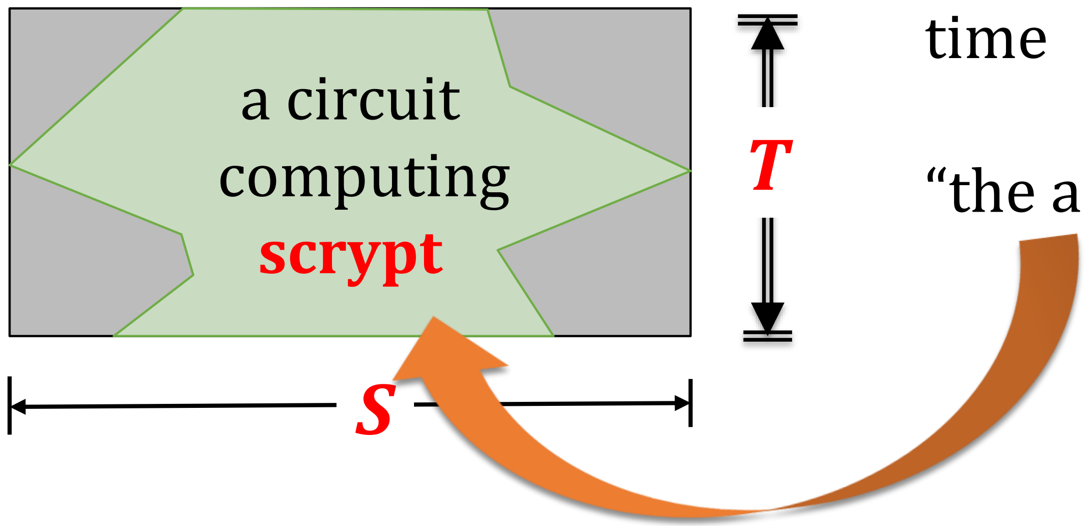


Note: $2S \ll 3S$.

So: the bound provided by Percival is meaningless.

The “right” definition **[Alwen and Serbinenko]**

instead of looking at $S \times T$...



look at the sum of memory cells used over time

“the area on the picture”

A recent result

Alwen, Chen, Pietrzak, Reyzin, and Tessaro: **Scrypt is Maximally Memory-Hard, Cryptology ePrint Archive**, Oct 2016

Password Hashing Competition

- announced in **2013**
- run by an **independent panel** of experts
- **website**: password-hashing.net
- winner (2015): **Argon2** (by Biryukov, Dinu, and Khovratovic)

broken (together with several other competition finalists) by
Alwen, Gaži, Kamath, Klein, Osang, Pietrzak, Reyzin, Rolínek, Rybár: **On the Memory-Hardness of Data-Independent Password-Hashing Functions**, Aug 2016

©2016 by Stefan Dziembowski. Permission to make digital or hard copies of part or all of this material is currently granted without fee *provided that copies are made only for personal or classroom use, are not distributed for profit or commercial advantage, and that new copies bear this notice and the full citation*.