KEYLESS CRYPTOGRAPHY

PHONG NGUYEN

http://www.di.ens.fr/~pnguyen

KEYLESS CRYPTO

- This is a type of cryptography that does not require keys!
- It is very efficient because it is based on symmetric techniques.
- It is widely used.

KEYLESS CRYPTO

- The basic object is a one-way function f:
  - Easy to compute.
  - Impossible to invert: given a random output y , it is too expensive to find an input x s.t. y=f(x).
- Two special cases:
  - Hash functions: arbitrarily-long input, small (fixed) output size.
  - Pseudo-random number generators (PRNG): small (fixed) input size, arbitrarily-long output.

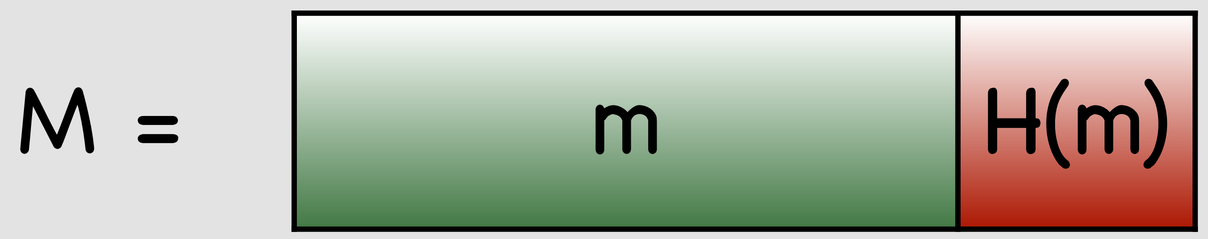
ONE-WAY FUNCTIONS (OWF)

- The input set of a OWF must be large enough to prevent exhaustive search.
- If we want a 128-bit security level, the input set must have at least 2^{128} elements. Is this the best attack?

HASH FUNCTIONS

CLASSICAL HASH FUNCTIONS

- Many numbers have a special format to detect errors.



- H is a hash function chosen to detect small modifications of m: for all m, $H(m) \neq H(m')$ for any $m' \neq m$ close to m.
- One checks the value of H(m).

WHICH ERRORS?

- The case of decimal digits:
  - Single-digit errors: 5 → 7
  - Transposition errors: 38 → 83
  - Twin errors: 66 → 88

EX: BOOK NUMBERS

ISBN 978-3-16-148410-0

m_{13}	m_{12}	…	m_4	m_3	m_2	$H(m) = m_1$

- Recent books = 13-digit number ISBN
- The digit $H(m)=m_1$ is defined by $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} 3m_i \equiv 0 \pmod{10}$.
- Ex: $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} 3m_i = (0+1+8+1+1+8+9)+3(0+4+4+6+3+7) \equiv 0 \pmod{10}$

9	7	8	3	1	6	1	4	8	4	1	0	0

EX: BOOK NUMBERS

ISBN 978-3-16-148410-0

m_{13}	m_{12}	…	m_4	m_3	m_2	$H(m) = m_1$

- The equation $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} 3m_i \equiv 0 \pmod{10}$ becomes incorrect if a single m_i is changed.
- Indeed, $f(x)=3x \pmod{10}$ is a permutation: why?

9	7	8	3	1	6	1	4	8	4	1	0	0

EX: BOOK NUMBERS

m_{13}	m_{12}	…	m_4	m_3	m_2	$H(m) = m_1$

- However, the equation $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} 3m_i \equiv 0 \pmod{10}$ may be preserved if two consecutive m_i are swapped.

9	7	8	3	1	6	1	4	8	4	1	0	0
9	7	**3**	**8**	1	6	1	4	8	4	1	0	0
9	7	8	3	**6**	**1**	1	4	8	4	1	0	0
9	7	8	3	1	**1**	**6**	4	8	4	1	0	0

EX: BOOK NUMBERS

ISBN 978-3-16-148410-0

m_{13}	m_{12}	…	m_4	m_3	m_2	$H(m) = m_1$

- Write a python program which checks if an ISBN number given as a string is correct: $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} 3m_i \equiv 0 \pmod{10}$

9	7	8	3	1	6	1	4	8	4	1	0	0
9	7	**3**	**8**	1	6	1	4	8	4	1	0	0
9	7	8	3	**6**	**1**	1	4	8	4	1	0	0
9	7	8	3	1	**1**	**6**	4	8	4	1	0	0

ISBN 978-3-16-148410-0

CAN WE DO BETTER?

m_{13}	m_{12}	…	m_4	m_3	m_2	$H(m) = m_1$

- The ISBN hash H(m) has the form $H(m)=\sum_{i\geq 2} a_i m_i \pmod{10}$ for fixed digits $a_2,\ldots,a_{13}$.
- **Th**: If such a hash detects all single-digit errors, it cannot detect the transposition errors (0,5), (1,6), (2,7), (3,8) and (4,9).

9	7	8	3	1	6	1	4	8	4	1	0	0
9	7	**3**	**8**	1	6	1	4	8	4	1	0	0
9	7	8	3	**6**	**1**	1	4	8	4	1	0	0
9	7	8	3	1	**1**	**6**	4	8	4	1	0	0

EX: CREDIT CARD NUMBERS

m_n	m_{n-1}	…	m_4	m_3	m_2	$H(m)$

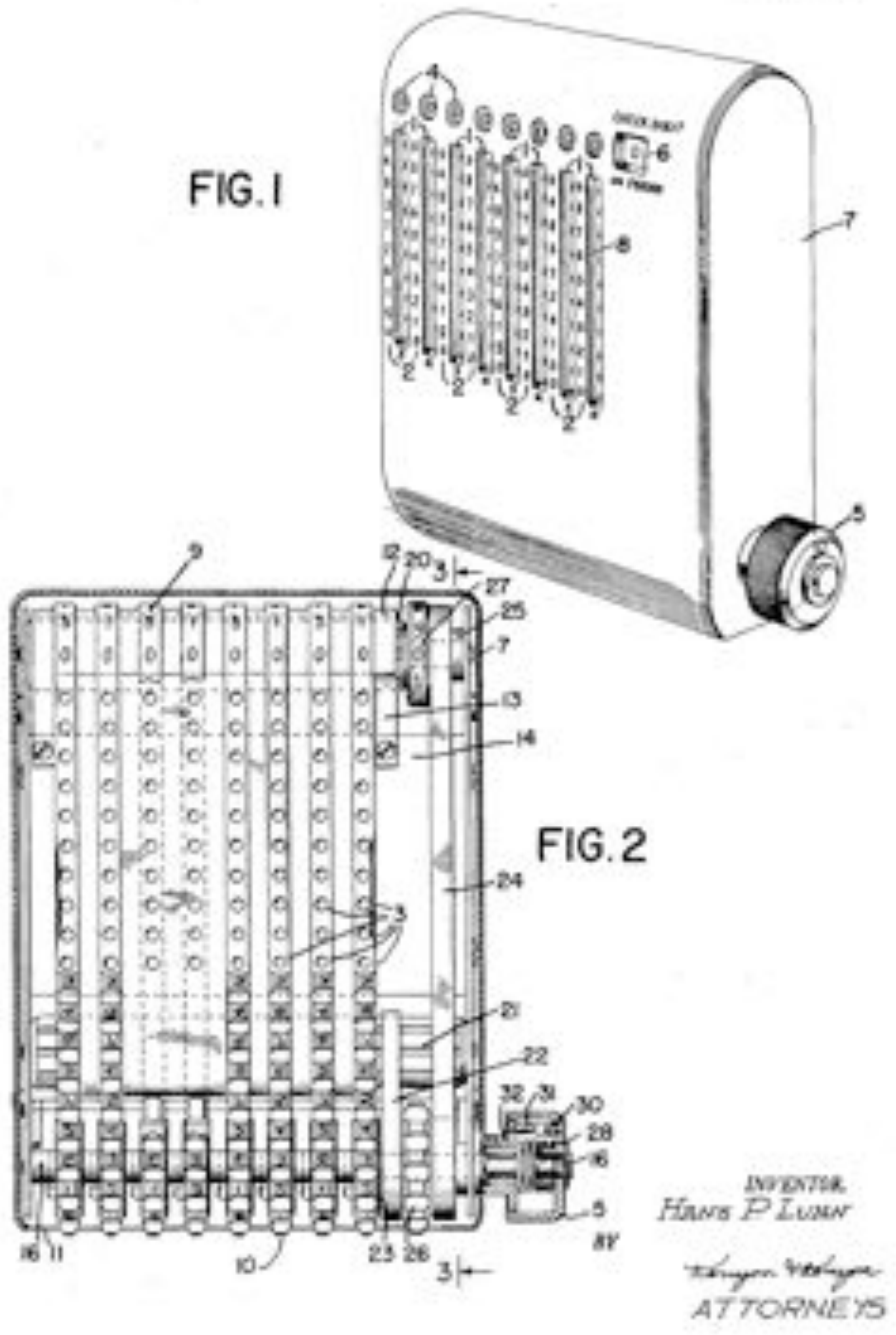
- $n = 16$ for VISA MasterCard and 19 for UnionPay 银联
- The digit $H(m)=m_1$ is defined by $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} f(m_i) \equiv 0 \pmod{10}$ where $f(x) = 2x+\lfloor x/5 \rfloor \pmod{10}$.
- This hash function was patented in 1954 by Luhn, an IBM engineer.
- Write a python program which checks if credit card number given as a string is correct:

LUHN'S PATENT (1954, 1960)

Aug. 23, 1960 H. P. LUHN 2,950,048

COMPUTER FOR VERIFYING NUMBERS

Filed Jan. 6, 1954 3 Sheets-Sheet 1



United States Patent Office

2,950,048

Patented Aug. 23, 1960

1

2,950,048

COMPUTER FOR VERIFYING NUMBERS

Hans P. Luhn, Armonk, N.Y., assignor to International Business Machines Corporation, New York, N.Y., a corporation of New York

Filed Jan. 6, 1954, Ser. No. 402,491

5 Claims. (Cl. 235—61)

This invention relates to a hand computer for computing a check digit for numbers or for verifying numbers which already have a check digit appended.

The principal object of the invention is to provide a simple, inexpensive and portable computer for computing check digits and to provide a simple device for verifying numbers which have a check digit appended.

A further object of the invention is to provide apparatus for computing, in a fast and simple manner, check digits to append to the numbers or to verify numbers with check digits attached.

Pursuant to the invention, a visual check is provided for use at the time of verification. Stamping means is also preferably provided for recording the verified number and for preserving the visual check, which may be appended to the number.

The apparatus of my invention is used in a checking system for multi-digit numbers to indicate whether, in transmitting a number, an error has been made, such as a transposition of the digits. It may be used, for example, where a great many parts are ordered, manufactured, invoiced, shipped, and billed by multi-digit numbers. When a number is first assigned to a new part a check digit is computed, as will be explained hereinafter, and this check digit is appended to the righthand end of the part number. Thereafter whenever the correctness of that part number is in question the number can always be easily and quickly verified by my invention.

The particular mathematical system of number checking preferably embodied in my invention is one in which a single digit, called the check digit, is appended to the righthand end of the original or true number. The value of this check digit is so computed that in verifying the number by cross addition of the multiple digits of the number and the check digit, in accordance with a rule of substitution, the result will be a zero. This zero will appear as such on the computer. If the stamping or printing means of my device is utilized, a check mark may be used to indicate that the number is correct.

Specific illustrations of my invention are shown in the accompanying drawings illustrating two embodiments of the invention, and in which:

Figure 1 is a front perspective view of one of the said embodiments of my device;

Figure 2 is a front elevation of one of the said embodiments partially in cross section;

Figure 3 is a cross-section of one of the said embodiments on the line 3—3 in Figure 2;

Figure 4 is a perspective view of a portion of the same, partly in cross section;

Figure 5 is a front elevation of another embodiment of my invention; and

Figure 6 is a sectional view taken on the line 6—6 in Figure 5.

For convenience of description, the operation of the apparatus of my invention, first in computing a check digit and secondly in verifying a number with a check digit appended, will be set forth to facilitate a complete

2

understanding of the function and purpose of the apparatus. This will be followed by a description of the apparatus and its operation.

It is commonly known that in copying a number comprised of a plurality of digits it often happens that an error occurs by transposing two of the digits. This common error is detected by the invention herein described by the cross addition of digits, the alternate digits being replaced by "substitute" digits, prior to the cross addition. It should be understood that other systems of cross addition checking could be utilized but the system used herein is described as a practical example. In such a method of cross addition for checking a number, it is readily seen that the straight cross addition of the original digits of a number would fail to give any information concerning erroneous transposition because the sum would be the same regardless of the relative placement of the digits. However, if every other digit is a substitute digit in accordance with the system herein set forth, such an error will be detected.

The substitute digit equals twice the original digit plus an end around carry (an end around carry in this system means the addition of any digit standing in the tens position to the digit standing in the units position in the doubled number, as shown below). Thus the substitute digit for an original –3– is –6–(–2–)×(–3–=–6–). The substitute digit for an original –6–, illustrating the end around carry, is –3–(–2–)×(–6–=–12–=–1–+–2–=–3–).

The following table gives the substitute for each digit according to this system.

Original	–0–	–1–	–2–	–3–	–4–	–5–	–6–	–7–	–8–	–9–
Substitute	–0–	–2–	–4–	–6–	–8–	–1–	–3–	–5–	–7–	–9–

Applying this system of substitute digits to determine the check digit for a number of seven digits (which is the number of digits provided for in the particular embodiment of the invention hereinafter described), such as –4872148–, first a check digit will be determined, and secondly the number with the check digit appended will be verified. In accordance with the substitution system utilized in my invention, the first digit of the number reading from left to right is a substitute digit, the second digit is an original digit and then this order is repeated until all of the digits have been accounted for. The first digit of the example number, the original –4–, is replaced by its substitute digit, an –8–. This –8– is added to the next digit –8–, an original digit, resulting in the sum of –16– which becomes a –6– by casting out tens in the usual manner. The next digit is an original –7– which is replaced by its substitute digit, a –5–. This –5– is added to the –6– resulting in a –1–. This cross addition, if continued in accordance with the above, across the remaining four digits of the sample number would result in a sum of –6–. This can be determined from the following table giving the original and alternate substitute digits for the number in question.

Original .... –4 8 7 2 1 4 8–
Alternate Substitute ... –8+8+5+2+2+4+7– = –6–

Once this sum of –6– has been computed the check digit to be appended is derived by adding to this sum its tens complement or in this case the digit –4–, this being the amount to be added to –6– to produce ten. If this –4– is added to the sum –6– as an original number, the total in the last column will be a –0–. The significance of this particular end result will become apparent in the explanation of the verification of a number having a check digit appended.

It should be realized that the check digit should be added in as an original number. This is accomplished by starting out by using either the original or the substitute digit for the first digit of the alternate substitute

LUHN'S FUNCTION

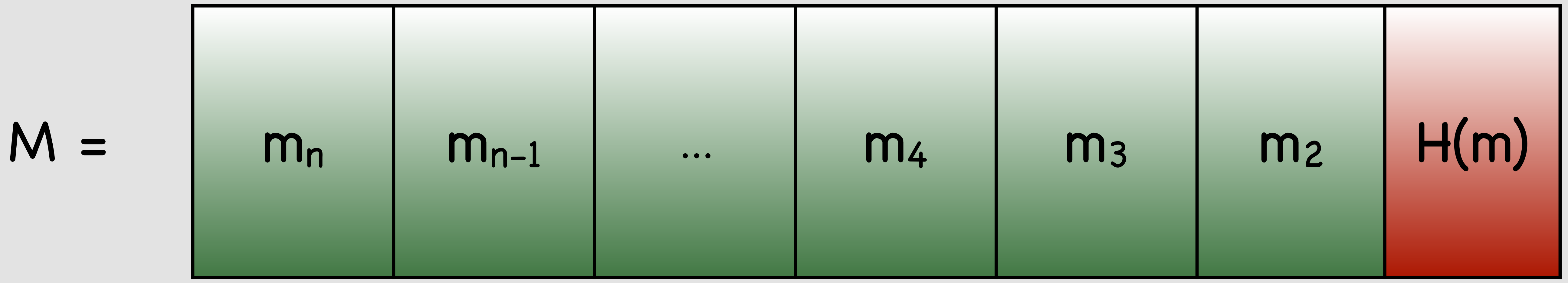
- $f(x) = 2x + \lfloor x/5 \rfloor \pmod{10}$.

x	0	1	2	3	4	5	6	7	8	9
f(x)	0	2	4	6	8	1	3	5	7	9
f(x)-x	0	1	2	3	4	6	7	8	9	0

- Is f a permutation?
- How many fixed points does f have?

PROPERTIES OF LUHN'S HASHING

- The check equation $\sum_{i \text{ odd}} m_i + \sum_{i \text{ even}} f(m_i) \equiv 0 \pmod{10}$ fails under either of the following modifications of



- A single digit is modified.
- Two distinct consecutive digits are swapped, except if it's 09 or 90 because f(0)=0 and f(9)=9. That's 98% of all transposition errors.

QUESTION

- To detect all single-digit errors and all transposition errors, we need a permutation f of $\{0,1,\ldots,9\}$ such that $y \longmapsto f(y)-y \pmod{10}$ is also a permutation.

- Does such an f exist?

A PHYSICIST GUESS

- There are 10! permutations f of {0,1,...,9}.
- The probability that a random function of {0,...,9} is a permutation is $10!/10^{10}$.
- Since $10! \times 10!/10^{10} \approx 1317$, maybe there exists a permutation f such that $y \longmapsto f(y)-y \pmod{10}$ is also a permutation...

YET...

- **Theorem**: There is no permutation f of {0,1,...,9} such that $y \mapsto f(y)-y \pmod{10}$ is also a permutation.
- Proof: Let f be a permutation. For any permutation g of {0,1,...,9}, $\sum_i g(i) = 9 \times 10/2 = 45 \equiv 5 \pmod{10}$.
- If $h(y)=f(y)-y$ then $\sum_i h(i) = \sum_i f(i) - \sum_i i \equiv 0 \pmod{10}$
  so h cannot be a permutation.

BETTER THAN LUHN?

m_n	m_{n-1}	…	m_4	m_3	m_2	$H(m)$

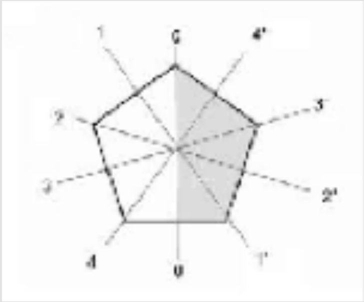
- Luhn's function is of the form $H(m)=\sum_{i\geq 2} f_i(m_i) \pmod{10}$ for some permutations $f_2, f_3 \ldots f_n$.
- **Theorem**: Such a hash can detect all single-digit errors, but cannot detect at least two transposition errors.
- Luhn's function is optimal among them.

BETTER THAN LUHN?

m_n	m_{n-1}	…	m_4	m_3	m_2	$H(m)$

- Luhn's function is of the form $H(m)=\sum_{i\geq 2} f_i(m_i) \pmod{10}$ for some permutations $f_2, f_3 \ldots f_n$.
- **Theorem**: Such a hash can detect all single-digit errors, but cannot detect at least two transposition errors.
- Proof: $f_i(m_i)+f_j(m_j)\neq f_i(m_j)+f_j(m_i)$ if $m_i\neq m_j$ is equivalent to (f_j-f_i) is a permutation, which is impossible: there must be $m_i\neq m_j$ s.t. $(f_j-f_i)(m_i)=(f_j-f_i)(m_j)$.

BETTER THAN LUHN?



m_n	m_{n-1}	…	m_4	m_3	m_2	H(m)

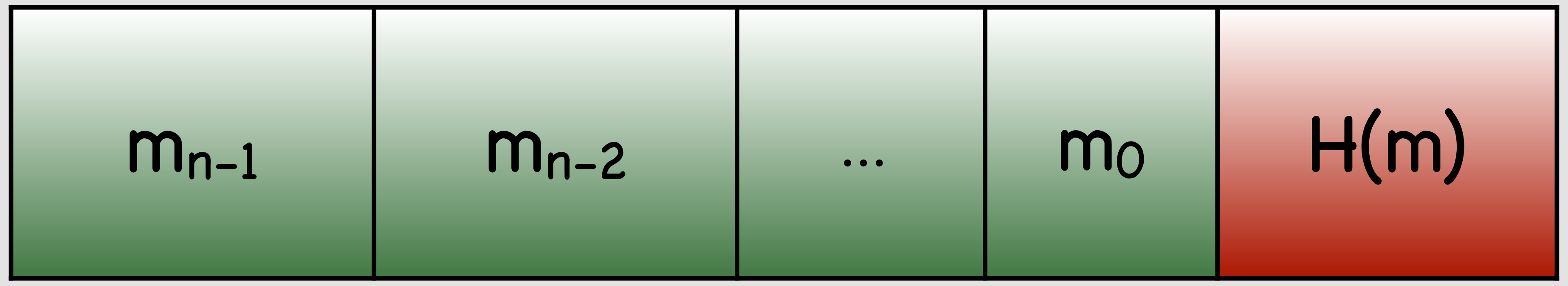
- [Verhoeff1969] detects all single-digit and transposition errors (and more) using the 10-order dihedral group: the check is $\sum_{i\geq1} f_i(m_i) = 0$ where + is the group operation, and the f_i's are iterates of some fixed permutation.
- [Damm2004] detects simply using a special quasigroup: the check is just $\sum_{i\geq1} m_i = 0$. Used in Singapore patent numbers.

RECAP

Check equation	Set of order 10	All transpositions?
$\Sigma_{i\geq 1}\ a_i m_i = 0$	$\mathbf{Z}/10\mathbf{Z}$	No
$\Sigma_{i\geq 1}\ f_i(m_i) = 0$	$\mathbf{Z}/10\mathbf{Z}$	No
$\Sigma_{i\geq 1}\ f_i(m_i) = 0$	Dihedral group	Yes
$\Sigma_{i\geq 1}\ m_i = 0$	Quasigroup	Yes

THE CASE OF BITS: PARITY CHECK

- The message is formed of bits $\in \{0,1\}$.



- Parity Check: H(m) is a bit $\in \{0,1\}$
  - 0 if the number of 1 in $(m_0,m_1,\ldots,m_{n-1})$ is even.
  - 1 otherwise.
- If we sum all bits modulo 2, we obtain 0.
- Detects all single-bit modifications.

THE CASE OF BITS: CRC

- CRC = cyclic redundancy check.
- Let G(X) be a binary polynomial of degree k.

m_{n-1}	m_{n-2}	…	m_0	H(m)	=	M_{n+k-1}	M_{n+k-2}	…	M_0

- $H(m_0,m_1,\dots,m_{n-1}) = (\sum_{0\leq i<n} m_i X^i)\, X^k \bmod G(X)$ thus $(\sum_{0\leq i<n+k} M_i X^i) \bmod G(X) = 0$
- Generalizes parity check: k=1 and G(X)=X+1
- Ex: Ethernet (1975) introduced k=32 with

$$G(X)=X^{32}+X^{26}+X^{23}+X^{22}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$$

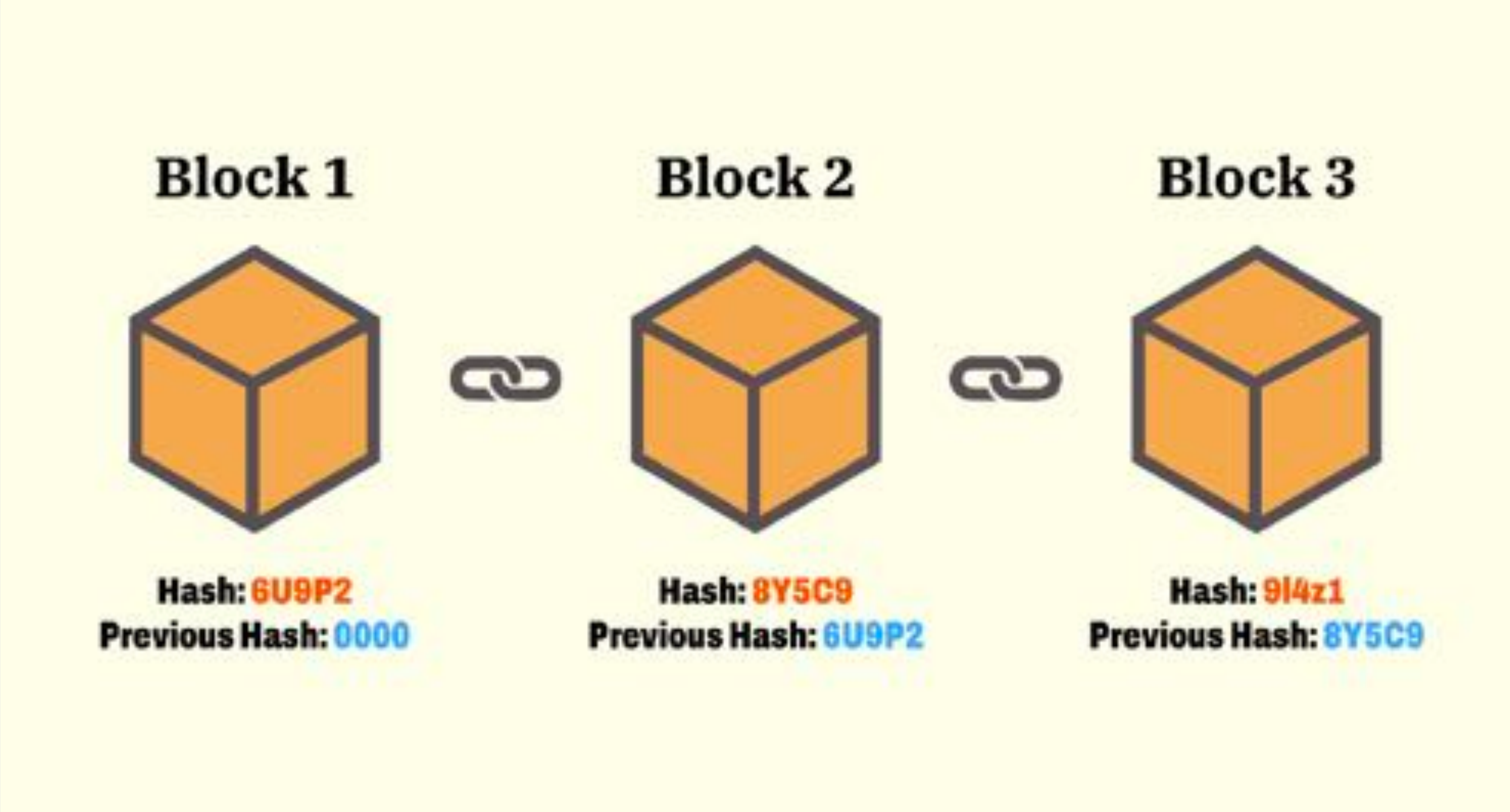
CRYPTOGRAPHIC HASH FUNCTIONS

CRYPTOGRAPHIC HASH FUNCTION

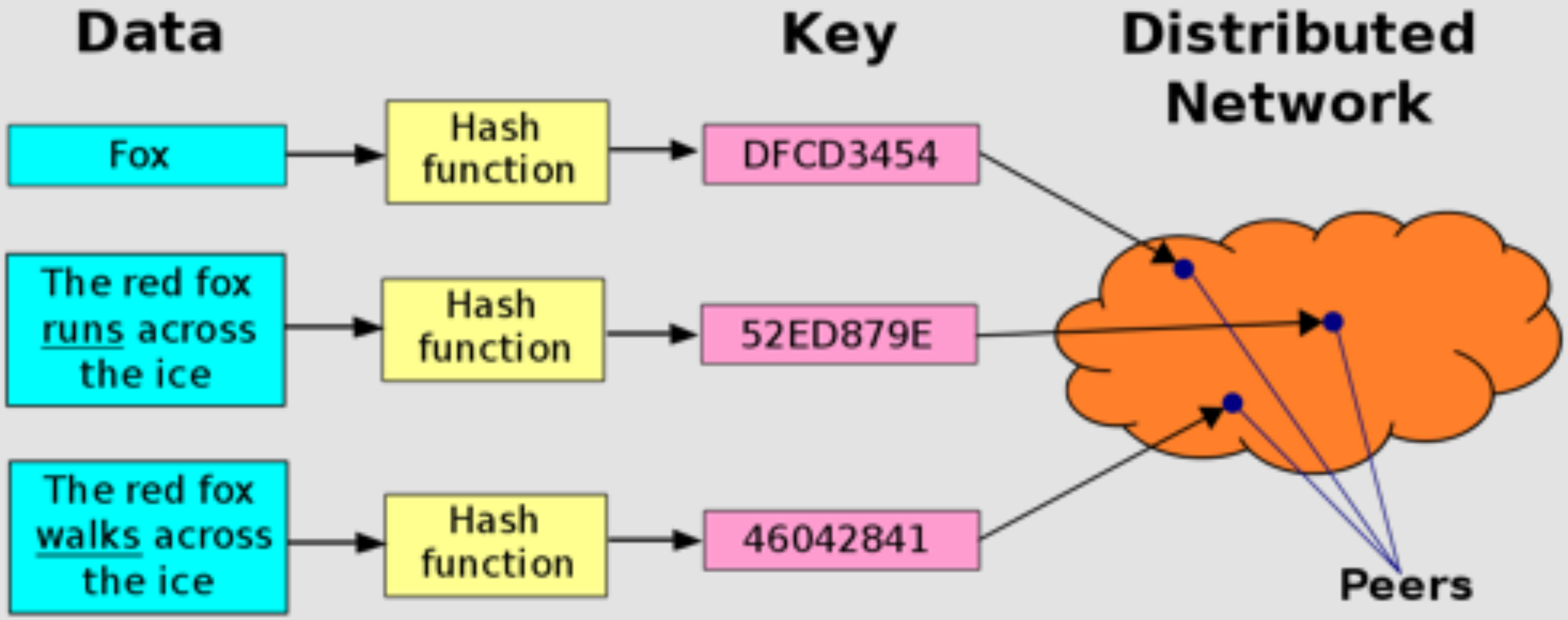
- Different from normal hash functions = checksums.
- A hash function outputs a small digest from any input:
  - Post-Typical sizes: 128 (MD5), 160 (SHA-1), 224-256-384-512 (SHA-2 and SHA-3).
- A hash function is deterministic.
  - On two identical inputs, the output is the same.
  - The output is called a digest.

EXAMPLE: BLOCKCHAINS

- Blocks are chained together by hashing.
- Each block contains the hash of the previous block.
- Each hashed block is "small".



IN PRACTICE


Data
Key
Distributed Network
Fox
Hash function
DFCD3454
The red fox runs across the ice
Hash function
52ED879E
The red fox walks across the ice
Hash function
46042841
Peers

IN PRACTICE

- Passwords are hashed, and only their digests are stored.
- To check passwords, we check hashes.
- It should be infeasible to retrieve a password from its digest: true if the hash function is a one-way function.

SECURITY REQUIREMENTS

- A cryptographic hash function must satisfy very strong security requirements, compared to traditional hash fuctions. They are slower, but still very fast, compared to other cryptographic functions.
- Intuitively, digest should “look like” random numbers. This is the random-oracle model: the hash function is modeled as a random function. But it cannot hold in practice, because a hash function is deterministic.

SECURITY REQUIREMENTS

- One-wayness
- Preimage resistance
- Collision resistance

ONE-WAYNESS

- Let H have n-bit digests.
- H is one-way if given a random n-bit y, it should be infeasible find x s.t. y=H(x).
- It is always possible to find x in time roughly 2^n by exhaustive search, so one asks that there is no better attack.
- If y is not random, it can be very easy.

TABLE LOOK-UP

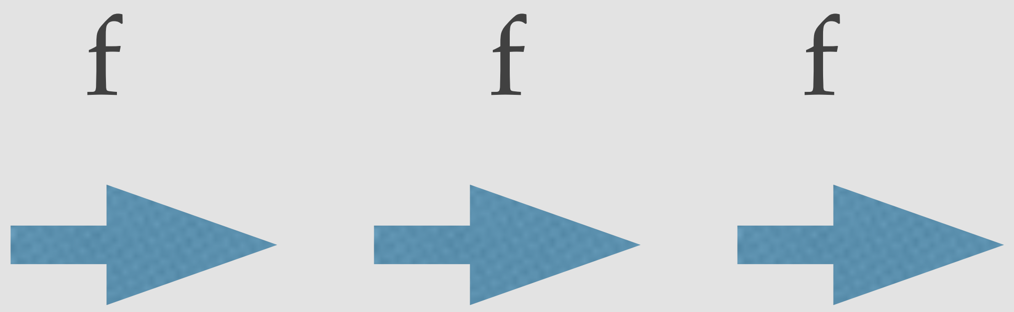
- Compute $f(x)$ for all x, and store them in a table $(x,f(x))$.
- Sort the table over $f(x)$.
- For any y, find the collision in the table.
- How much does it cost?

TIME MEMORY TRADE/OFF

- Assume that the input set has n elements.
- Exhaustive search: Time = n evaluations ; Space = Negligible
- Table Look up: Time = log(n) ; Space = n pairs ; Precomputation = n*log(n) operations.
- Hellman (1980) introduced time/memory trade-offs, which are intermediate between exhaustive search and table look up. Last improvement: rainbow tables (2003).
    - Application: Password cracking.

OVERVIEW

- Assume that $f:E \rightarrow E$ where E has n elements.
- Imagine that E can be rewritten as an $m \times t$ array $(x_{i,j})_{1 \leq i \leq m,\ 1 \leq j \leq t}$ where $x_{i,j}=f(x_{i,j-1})$.
- We **only store the first and the last column**, and we sort the last column: this costs $O(m \log m)$.



$x_{1,1}$	$x_{1,2}$		$x_{1,t}$
$x_{2,1}$	$x_{2,2}$		$x_{2,t}$
…	…		…
…	…		…
$x_{m,1}$	$x_{m,2}$		$x_{m,t}$

IDEA

- Given $y \in E$, we check if y belongs to the last column: this costs $O(\log m)$.
  - If yes, we know i s.t. $y = x_{i,t} = f(x_{i,t-1})$, so we compute $x_{i,t-1} = f(f(\ldots f(x_{i,1})))$ from the first column: this costs $O(t)$ evaluations.
  - Otherwise, we compute $y_1 = f(y)$ and check if y_1 belongs to the last column.
    - If yes, we know i s.t. $y_1 = x_{i,t} = f(f(x_{i,t-2}))$, so we compute $x_{i,t-2} = f(f(\ldots f(x_{i,1})))$ from the first column, and hope that $y = f(x_{i,t-2})$.

$\xrightarrow{f}$ $\xrightarrow{f}$ $\xrightarrow{f}$

$x_{1,1}$	$x_{1,2}$		$x_{1,t}$
$x_{2,1}$	$x_{2,2}$		$x_{2,t}$
…	…		…
…	…		…
$x_{m,1}$	$x_{m,2}$		$x_{m,t}$

ANALYSIS

- If « everything goes well », the online cost is at most:
  - $O(t)$ evaluations of f
  - $O(t)$ table look-ups of the last column: total cost = $O(t \log m)$.
  - Space cost: $O(m)$ elements.
- So the total online cost is: Time=$O((t+m) \log m)$, Space = $O(m)$.
- But it is not clear that we can take $mt=n$: [Hellman80] suggests that we can take $mt^2=n$.

PREIMAGE RESISTANCE

- Given a random x, it should be infeasible to output $x' \neq x$ s.t. $H(x')=H(x)$.
- What's the difference with one-wayness?
- It is always possible to find x′ in time roughly 2^n by exhaustive search, so one asks that there is no better attack.

COLLISION RESISTANCE

- Nobody knows a collision, that is, a pair (x,y) s.t. $x \neq y$ and $H(x)=H(y)$.
- In theory, there are infinitely many collisions, but we may not know any of them.
- For MD5, many collisions have been found, and generating new ones only costs a few seconds.
- It is always possible to find collisions in time roughly $2^{n/2}$ by "exhaustive search", thanks to the birthday paradox.

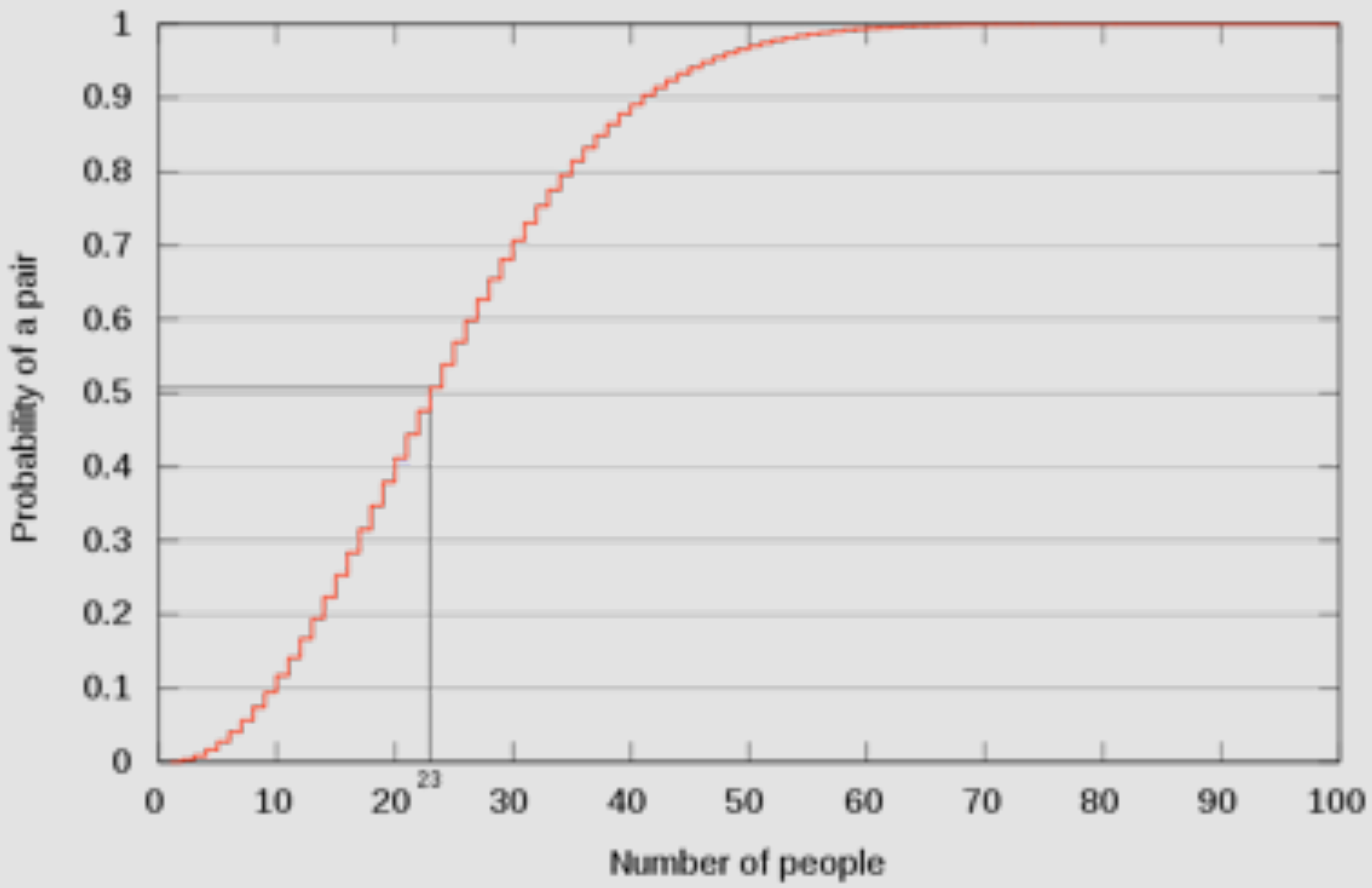
THE BIRTHDAY PARADOX

THE BIRTHDAY PARADOX

- Consider m people in this room.
- From which value of m can we expect to have two people with the same birthday (same day, same month)?

THE BIRTHDAY PARADOX

- If m>365, there must be a collision.
- But if we assume that birthdays are uniformly distributed and independent, then much smaller m suffice:
  - Proba = 0.507 for m=23
  - Proba = 0.706 for m=30

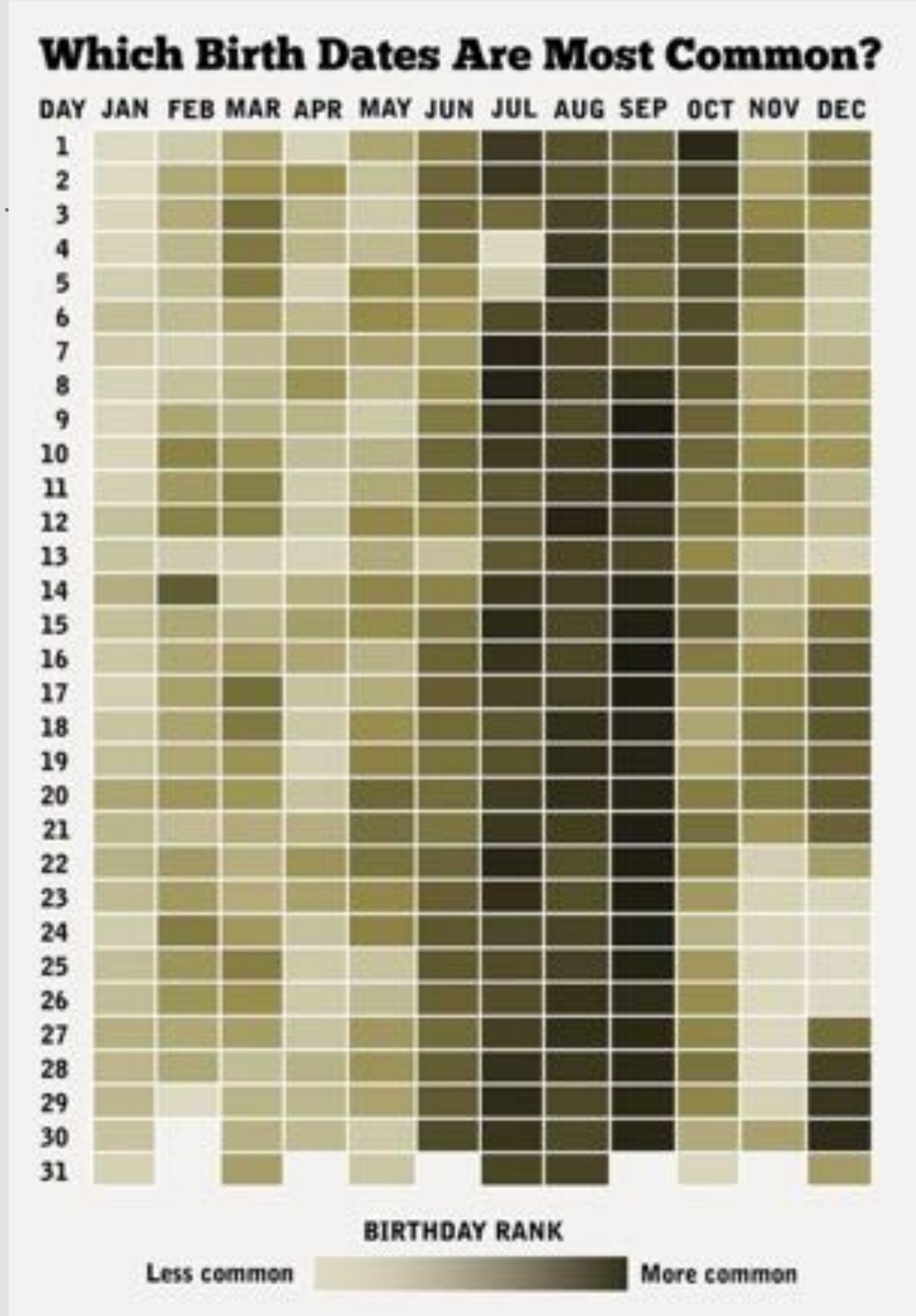


REAL-WORLD EXAMPLE

- Football’s tournament teams have 23 players.
  - Half of the teams should have birthday collisions.
  - Recent world cups have 32 teams.
- Teams with birhday collisions:
  - 2018 World Cup: 16/32 teams.
  - 2014 World Cup: 19/32 teams.
  - 2010 World Cup: 17/32 teams.

ACTUAL DISTRIBUTION OF BIRTHDAYS

- Not exactly uniform
- But this is increases the probability of collisions



GENERALIZED BIRTHDAY PARADOX

- Choose m values uniformly at random from a set of N elements.
- If $m \geq \sqrt{\frac{N\pi}{2}}$, there is a collision with probability $\geq 1/2$.
- The probability increases significantly with slightly higher values of m.
- Essentially, $\sqrt{N}$ values are enough to guarantee the existence of collisions.

GENERALIZED BIRTHDAY PARADOX

- Choose m values uniformly at random from a set of N elements.
- Pr(no collision) =

$$1 \times \frac{N-1}{N} \times \frac{N-2}{N} \times \cdots \times \frac{N-m+1}{N} = \frac{N!}{N^m (N-m)!}$$

$$= 1 \times (1 - 1/N) \times (1 - 2/N) \times \cdots \times (1 - (m-1)/N)$$

$$\approx e^{-1/N} e^{-2/N} \ldots e^{-(m-1)/N} = e^{-m(m-1)/(2N)}$$

THE BIRTHDAY ATTACK

- Let H have n-bit digests.
- The birthday attack is a low-memory attack to find collisions for H in time roughly $\sqrt{(2^n)}=2^{n/2}$.
- What is the cost of a naive attack based on the birthday paradox?
- The security level for collision resistance is at most n/2 bits, rather than n for preimage resistance and one-wayness.

NAIVE BIRTHDAY ATTACK

- Generate random messages $m_1,\ldots,m_t$.
- Compute their hashes $h_1=H(m_1),\ldots, h_t=H(m_t)$.
- If $t \geq 2^{n/2}$, we should have a collision: $h_i=h_j$.
- But how much does it cost to find it?

LESS NAIVE BIRTHDAY ATTACK

- Generate random messages $m_1,\ldots,m_t$.
- Compute their hashes $h_1=H(m_1),\ldots, h_t=H(m_t)$.
- If $t \geq 2^{n/2}$, we should have a collision: $h_i=h_j$.
- Sort $h_1=H(m_1),\ldots, h_t=H(m_t)$: this costs $O(t \ln t)$ time and $O(t)$ space.
- Now, collision can be found in time $O(t)$.
- Overall, the cost is now $O(2^{n/2})$ in time and space, ignoring polynomial factors.

THE BIRTHDAY ATTACK

- Generate a random message m_1.
- Consider the recursive sequence $m_2=H(m_1),\ldots, m_t=H(m_{t-1})$.
- If $t \geq 2^{n/2}$, we should still have a collision: $h_i=h_j$.
- Any such collision creates many collisions: $m_i=m_j$ implies that for any $k\geq 0$, $m_{i+k}=m_{j+k}$
- Cycle-detection algorithms find a collision in time $O(2^{n/2})$ and negligible space (faster with more space).

FLOYD'S ALGORITHM (1967?)

- $a := H(m_1)$; $b := H(a)$
- while $a \neq b$ do $a := H(a)$; $b := H(H(b))$
- $t := 0$; $b := a$; $a := m_1$
- while $a \neq b$ do $a := H(a)$; $b := H(b)$; $t := t+1$
- $u = 1$; $b := H(a)$
- while $a \neq b$ do $b := H(b)$; $u := u+1$
- Return the pair (m_t, m_{t+u})
- There are many variants.

GENERIC ATTACKS

- Let H have n-bit digests.

Problem	Cost of generic attack
One-wayness	2^n
Preimage	2^n
Collision	$2^{n/2}$

IDEALIZED HASHING

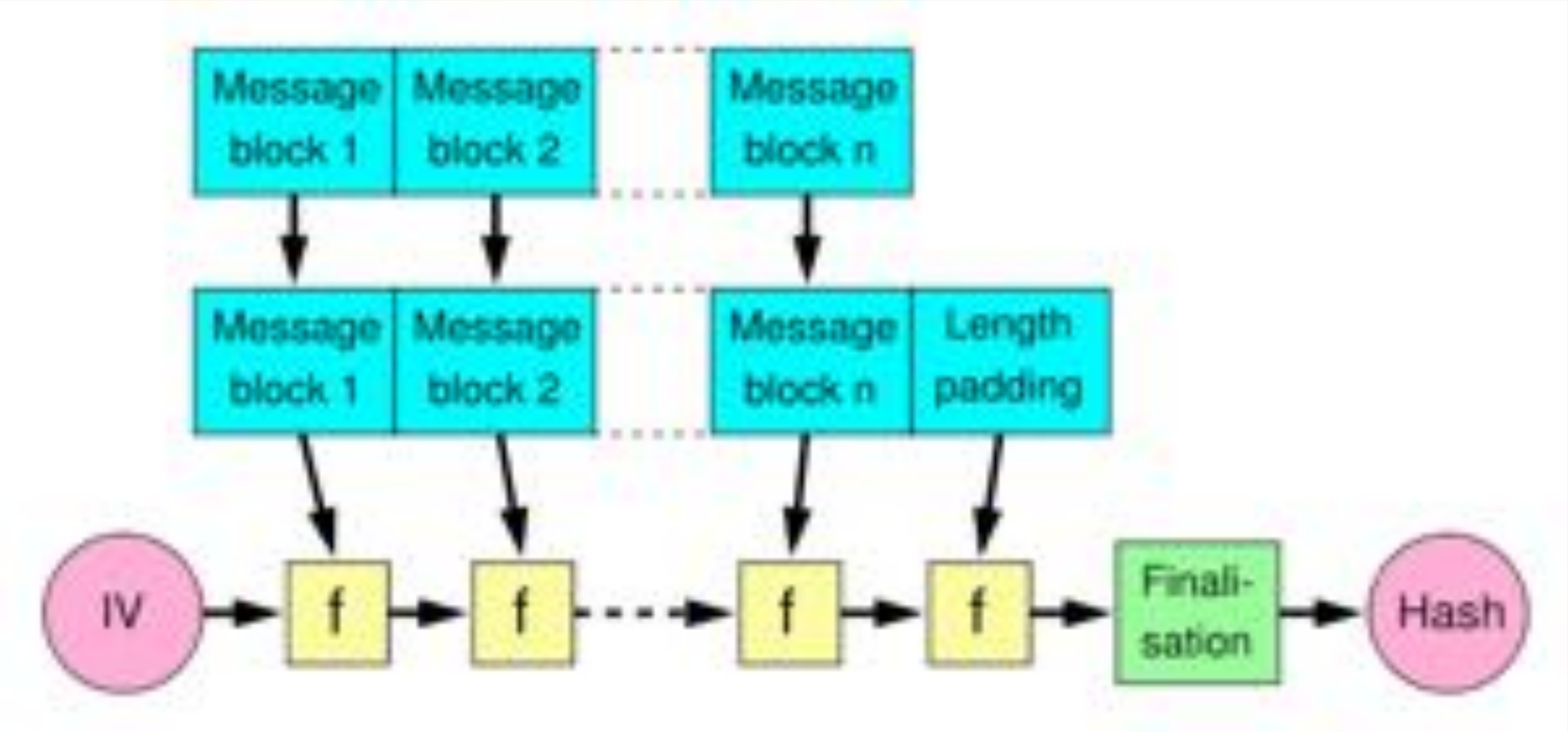
- Description of a random oracle
  - Store a list of (msg,digest), initially empty.
  - When a msg is queried:
    - if in the list, output the same digest.
    - otherwise, generate a new perfectly random digest, and add (msg,digest) to the list.
- But random oracles do not exist in the real world. Why?

ITERATIVE HASHING

- Most cryptographic hash functions rely on iterative hashing: intuitively, we split the message into many blocks, and hash blocks.
- MD5, SHA-1 and SHA-2 use the so-called Merkle-Damgard construction.
- Recent hash functions like SHA-3 use stronger constructions.

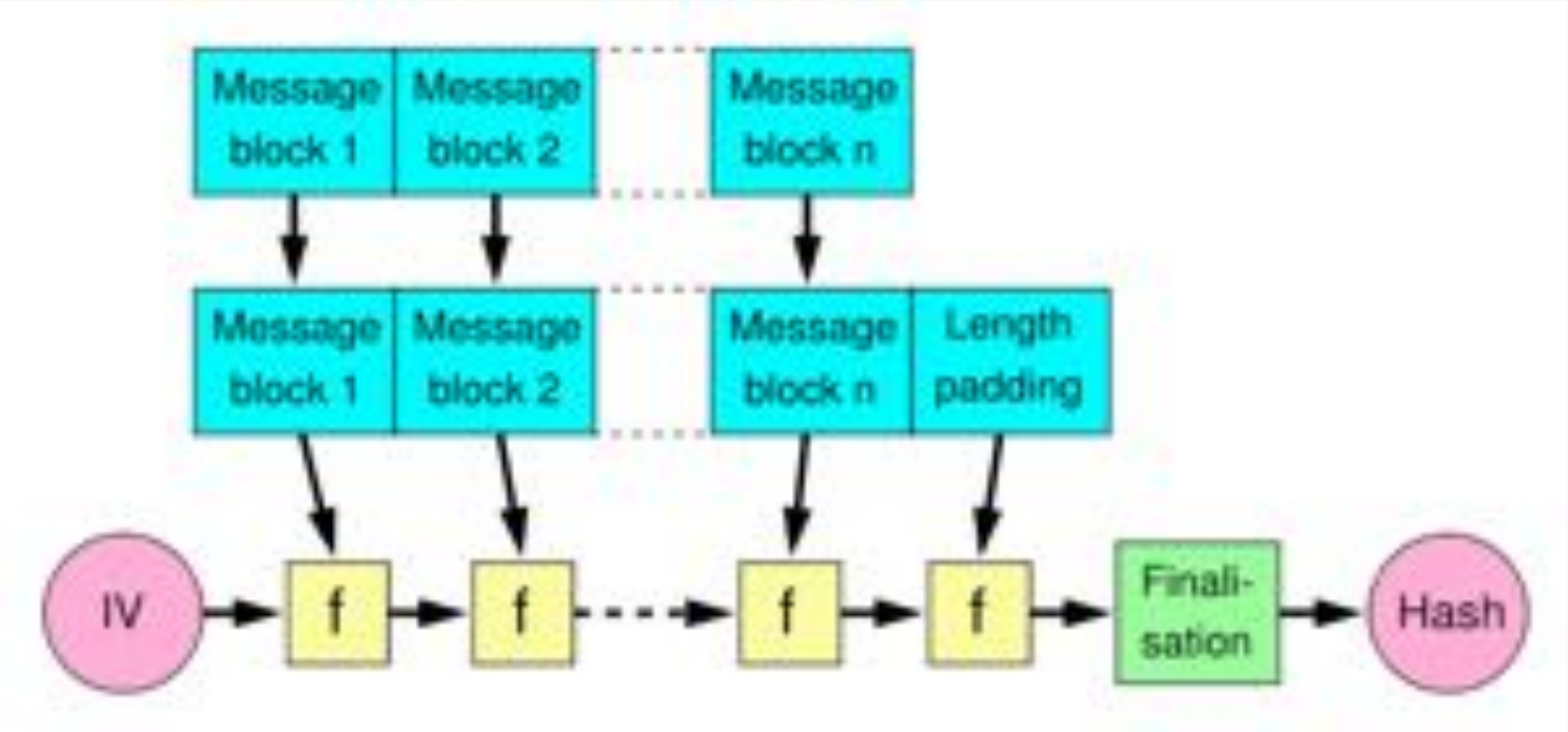
MERKLE-DAMGARD

- Use a compression function f, which is a mini-hash-function for fixed input size.



LENGTH PADDING

- Show that if one removes the length padding, then one can easily find preimages.



MERKLE-DAMGARD SECURITY

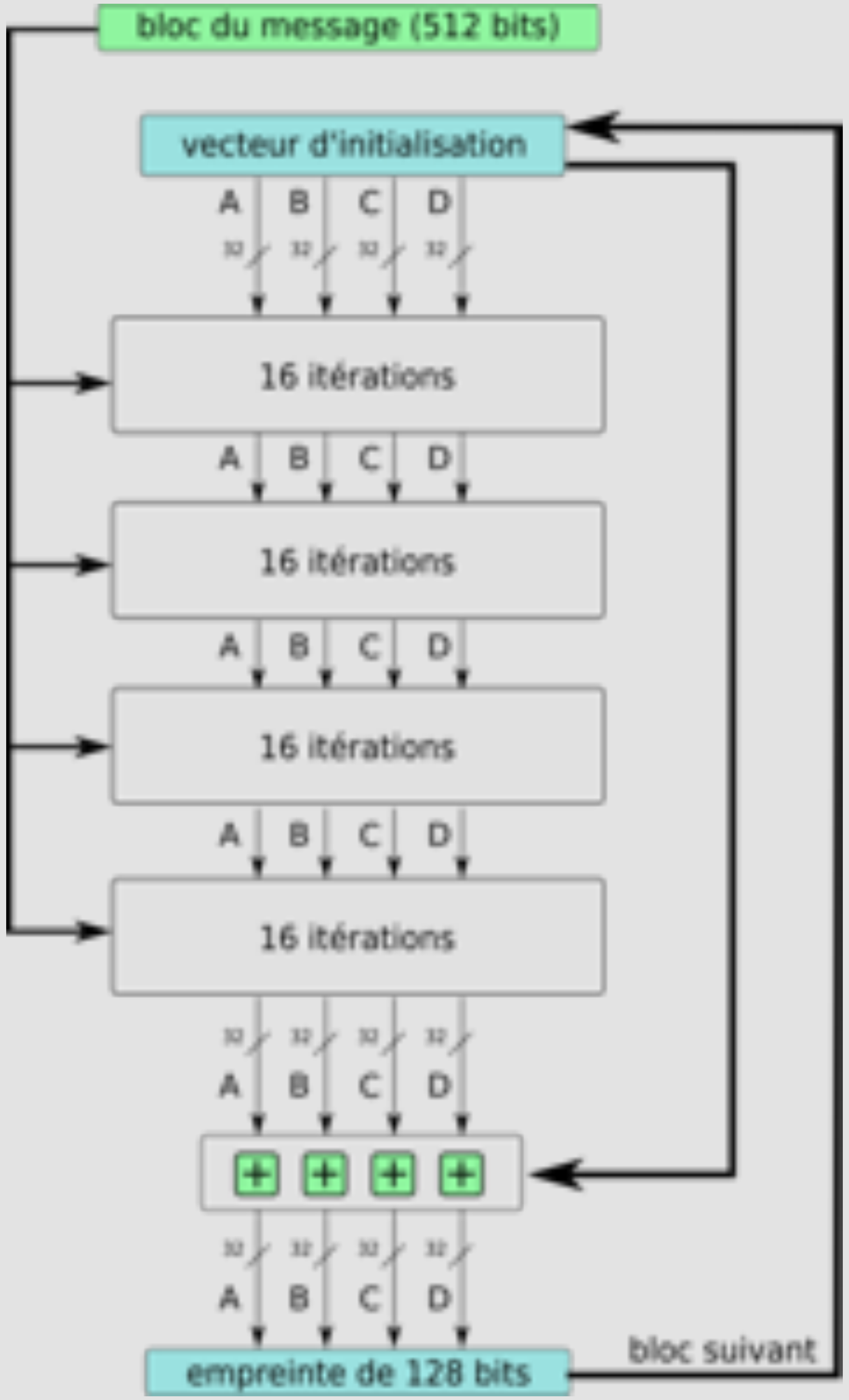
- "Any MD-hash function is not weaker than its compression function."
- Theorem: If one knows a collision $H(x)=H(x')$, then one can deduce $(h,m)\neq(h',m')$ such that $f(h,m)=f(h',m')$.
- Any collision in the hash function gives rise to a collision in the compression function.

MD5

THE CASE OF MD5

- Designed by Rivest in 1991, to fix MD4.
  - Trivial finalization
  - Block = 512 bits
  - IV and output = 128 bits
  - Compression has two inputs:
    - 128 bits decomposed as 4*32 bits.
    - 512 bits = message block.
  - Very efficient in software.

MD5 COMPRESSION

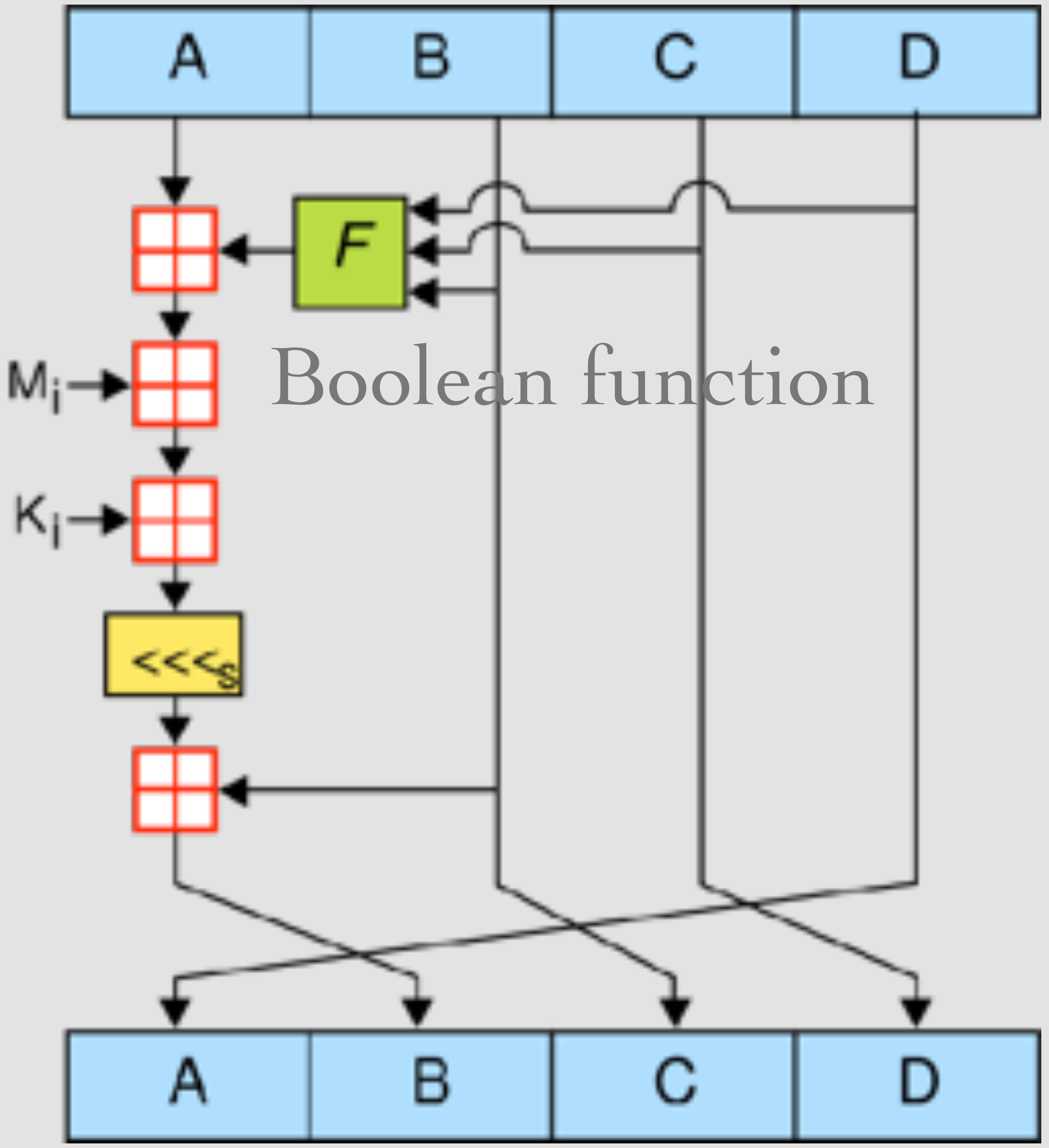


64 rounds

Msg word

Cst word

Boolean function



BEHIND MD5 COMPRESSION

- Most compression functions rely on a dedicated block cipher.
- MD5's compression uses the Davies-Meyer mode of operation: $f(IV,m) = E_m(IV) \oplus IV$
  - $E_k()$ is a block cipher
  - $\oplus$ is addition modulo 32 bits, word by word.

PSEUDO-COLLISIONS ON MD5 (1996)

- 1996: Dobbertin found pseudo-collisions, which are collisions for the compression function.
  - He found IV′ and m≠m′ s.t. f(IV′,m)=f(IV′,m′).
  - Because IV ≠ IV′, this did not give collisions for MD5 itself.

COLLISIONS ON MD5 (2004)

COLLISIONS ON MD5 (2004)

- 4072-byte files hello and erase.
- md5 erase
    - MD5 (erase) = da5c61e1edc0f18337e46418e48c1290
- md5 hello
    - MD5 (hello) = da5c61e1edc0f18337e46418e48c1290

COLLISIONS ON MD5 (2004)

- 2004: Wang et al. found the first MD5 collisions, for any IV, including the wrong IV given in Schneier's book.
  - Today, the best variants of Wang's attack only cost less than one second.
  - Wang's collision was for 1024-bit messages.
- 2010: Chinese researchers disclosed the first 512-bit collision: $f(IV,m)=f(IV,m')$ i.e. $E_m(IV)=E_{m'}(IV)$

FASTER MD5 COLLISIONS: ATTACKS ONLY GET BETTER

YEAR	LOG-COST
≤2004	64
2004 (Wang et al.)	40
2005	37
2006	32 i.e. ≤ 5 minutes
2007	25
2008	21
2009 (Stevens et al.)	16 i.e. less than one second!

BEHIND THE MD5 COLLISIONS

- The main reason is that the underlying block cipher is weak, particularly against differential cryptanalysis.
- In fact, Wang's attack is a differential attack:
  - For two blocks:
    $MD5(m_1 m_2) = f(f(IV,m_1),m_2) = f(E_{m1}(IV)\oplus IV,m_2) = E_{m2}(E_{m1}(IV)\oplus IV)\oplus E_{m1}(IV)\oplus IV.$
  - So Wang's collision implies that:
    $E_{m2}(E_{m1}(IV)\oplus IV)\oplus E_{m1}(IV) = E_{m'2}(E_{m'1}(IV)\oplus IV)\oplus E_{m'1}(IV)$

FLEXIBILITY ON MD5 COLLISIONS

- Random collisions are not very useful
- Finding structured collisions can be very dangerous

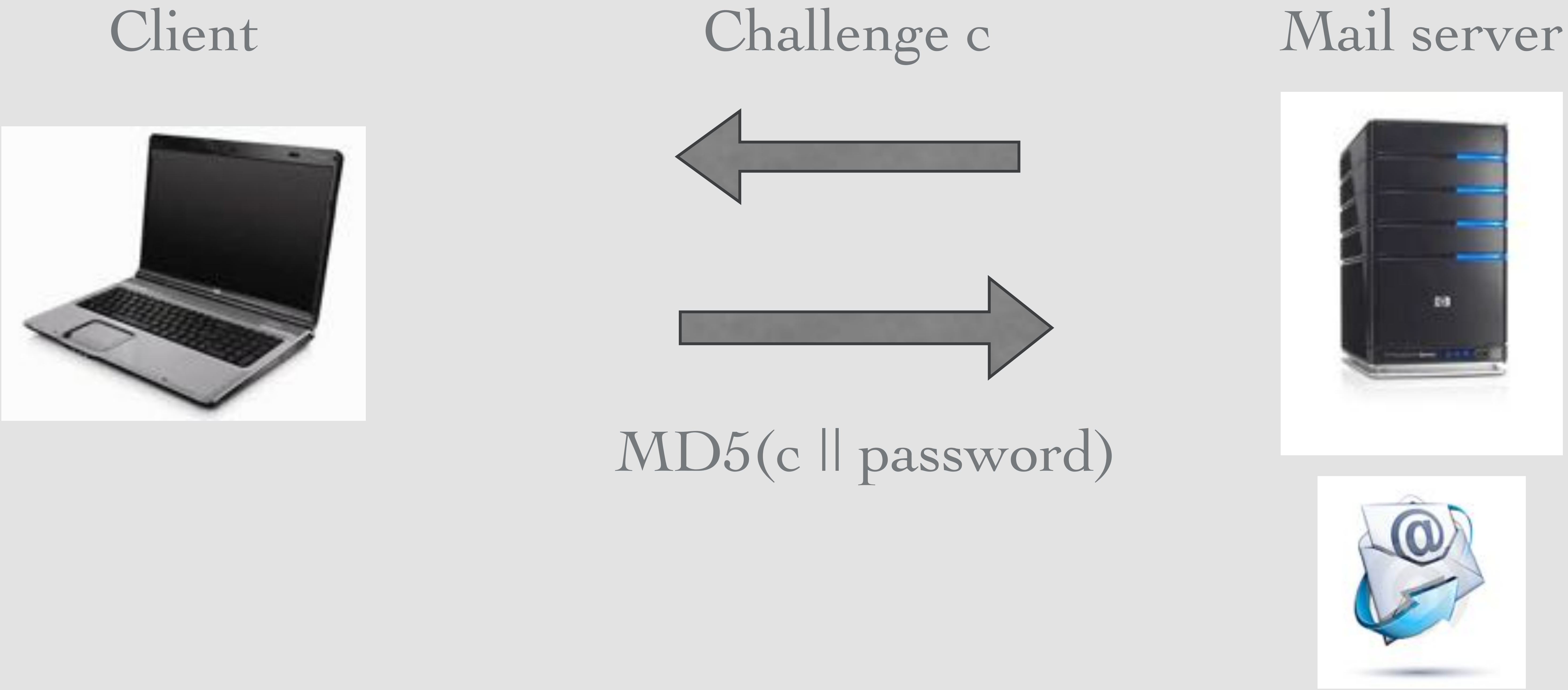
FLEXIBILITY ON MD5 COLLISIONS

- Given arbitrary files f_1 and f_2 of the same size, it is now possible to « quickly » generate m_1 and m_2 s.t. $MD5(f_1 || m_1) = MD5(f_2 || m_2)$
- In 2009, Stevens et al. did this in one day on a 200-PS3 cluster with 2048-bit m_1 and m_2. It allowed them to create a fake certification authority.
- Similar techniques were used in the Flame virus allegedly designed by a nation-state.
- It can be applied to many settings: programs, pictures with the same MD5.

IMPACT OF MD5 COLLISIONS

- Not all applications of MD5 are insecure.
- Yet, it is now strongly recommended to discard MD5, because several applications have been attacked.
  - X.509 certificates: [Stevens et al. 2009] were able to create a fake certification authority, accepted by all browsers.
  - Meet-in-the-middle attacks on APOP.

APOP



- There are efficient meet-in-the-middle attacks to recover passwords, based on Wang's MD5 collision techniques. [Leurent2007,Sasaki et al. 2008]
- A fake mail server sends a few challenges c, and recover the password from MD5(c | pwd) by observing if there are collisions or not.

REPLACING MD5

- The main alternatives to MD5 are the following standard algorithms:
  - SHA-1, used by BitTorrent
  - The SHA-2 Family, used by BitCoin
  - The SHA-3 standard
- All are slower and more secure than MD5.

SHA-1 AND SHA-2

SECURE HASH STANDARD: SHA-1

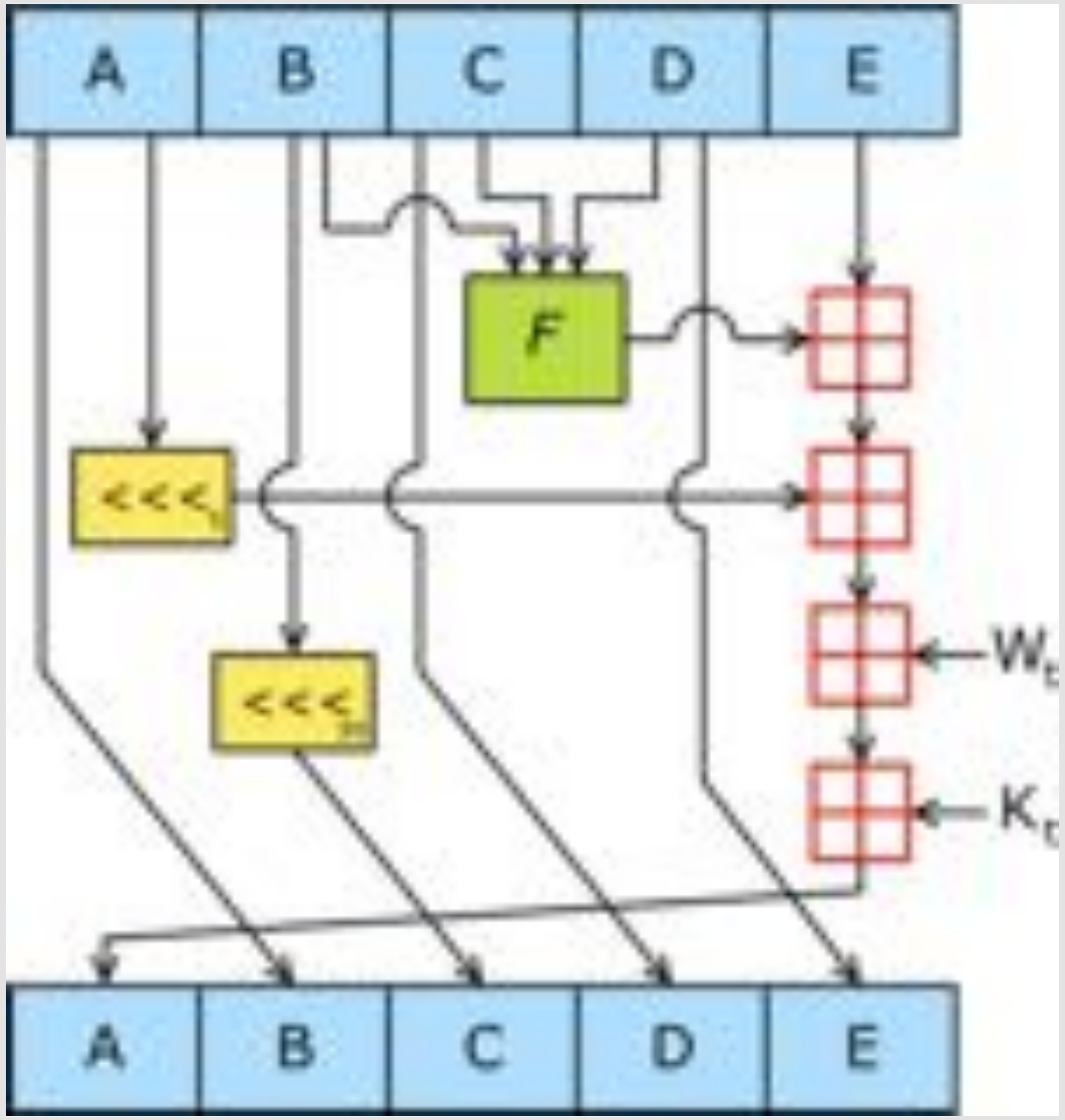
- US standard released in 1995, designed by the NSA
- It’s a very slightly modified version of SHA-0, released in 1993 and withdrawn.
- Digests are 160 bits: breakable by Bitcoin power in one day.
- Compression function: 512-bit x 160-bit.

DESIGN OF SHA-1

- Merkle-Damgard like MD5.
- Davies-Meyer like MD5.
- But the underlying block cipher is a bit different:
  - 160 bits rather than 128 bits.
  - 80 rounds rather than 64 bits.
  - Non-trivial key schedule.

SHA-1

- W is a 80-word array generated by the message block.
- In some sense, each W_t is a subkey, and the generation of W is the key schedule.



SHA-1 MESSAGE EXPANSION

- For t = 0 to 15 do
  - W[t] = Message[t]
- For t = 16 to 79 do
  - W[t] = LeftRotate(W[t-3] XOR W[t-8] XOR W[t-14] XOR W[t-16])
- In SHA-0, there was no LeftRotate.

SECURITY OF SHA-1

- 1998: 2^{61} collision attack on SHA-0 [Chabaud-Joux].
- 2004: Joux et al. find the first SHA-0 collision, in time 2^{51}.
- 2005: Wang et al. announced a collision attack on SHA-1 of cost $2^{69} \ll 2^{80}$.
- 2015: a SHA-1 compression-collision has been found.
- 2017: Stevens et al. find the first SHA-1 collision.
- No longer recommended by US federal agencies.
  - Browsers stopped using SHA-1 in certificates in 2017.
  - NIST wants SHA-1 to be retired by 2030.

SHA-1 FREE-START COLLISION (2015)

The following two input IV/message pairs give the same output value after applying the SHA-1 compression function:

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**)

SHA-1 COLLISION (HTTPS://SHATTERED.IO/)

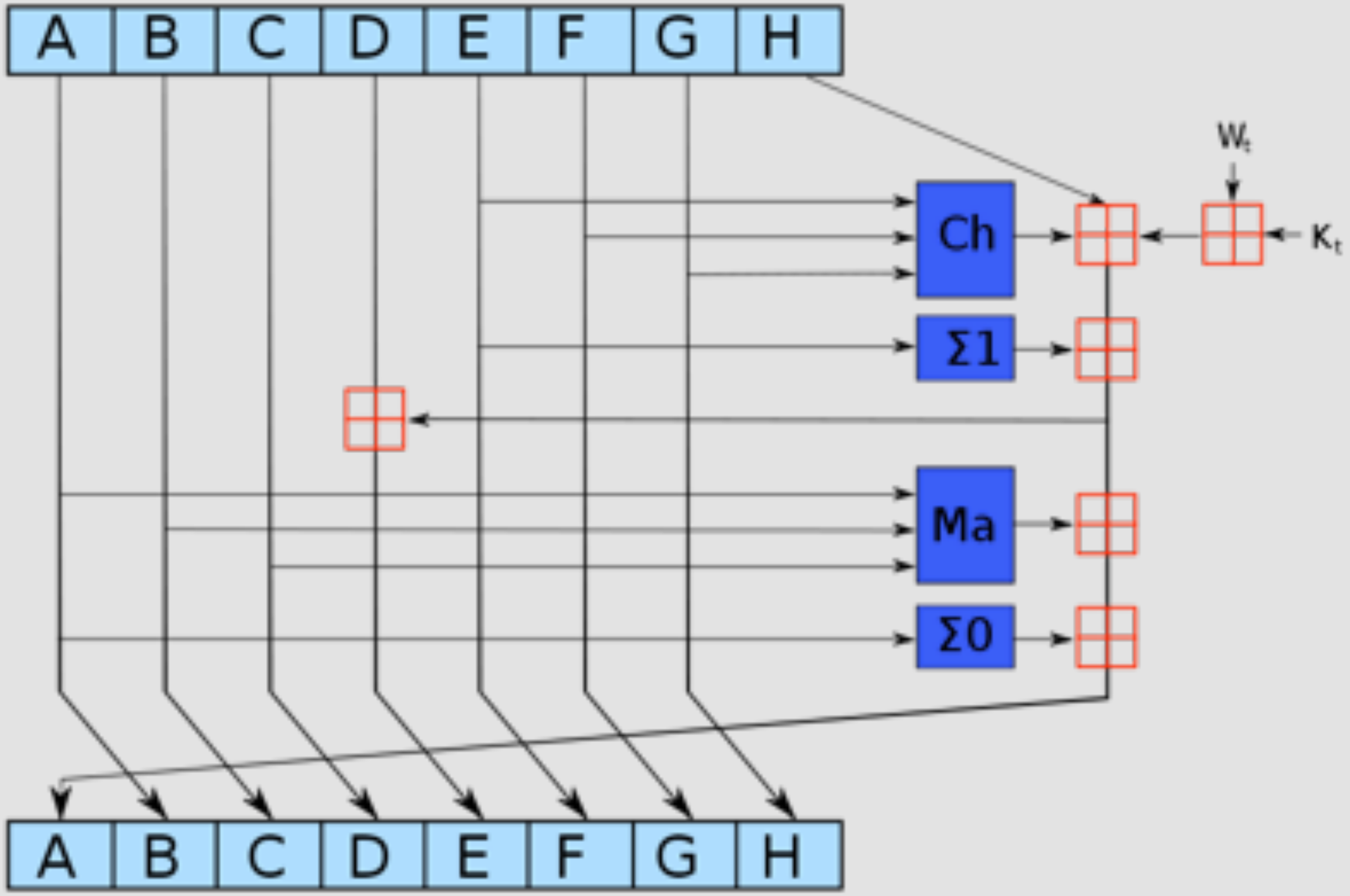
- In Feb. 2017, an international team (Stevens et al.) found the first SHA-1 collision using approximately 2^{63} hash computations. It took 6500 CPU years and 100 GPU years.

SHA-2 (2001)

- Digest size: 224, 256, 384, 512.
- Designed by the NSA to be more secure than SHA-1.
- Still Merkle-Damgard and Davies-Meyer.
- More unbalanced Feistel network for the block cipher.
- SHA-256/512 uses 32/64-bit words and 64/80 rounds.
- SHA-224/384 are simply truncations.

SHA-2: EXPANSION AND ROUND

- Let w[0-15] be the block message.
- **for** i **from** 16 to 63
  - s0 := (w[i-15] ⋙7) **xor** (w[i-15] ⋙18) **xor** (w[i-15] ⋙3)
  - s1 := (w[i-2] ⋙17) **xor** (w[i-2] ⋙19) **xor** (w[i-2] ⋙10)
  - w[i] := w[i-16] + s0 + w[i-7] + s1



SHA-2 DEPLOYMENT

- Double SHA-256 is used in Bitcoin.

- In Nov. 2023, the Bitcoin network performs 2^{68} hash/sec = 2^{93} hash/year where hash = SHA-256(SHA-256()). The hardware cost is ≤ 400 million.
- How is it possible? We estimated that the total number of PCs sold per year can compute 2^{86} clock cycles/year and SHA-256 requires much more than one clock cycle!

SHA-2 HARDWARE

- A 2-GHz core performs $2 \times 10^9 = 2^{31}$ clock cycles/sec.
- In 2023, you can buy an ASIC dedicated for SHA-256 with a speed of 120 TH/s = 120×10^{12} hash/sec for about 2,000 USD. Instead of one clock cycle, the ASIC performs 56,000 hash!
- Besides speed, what matters is energy efficiency.

PROOF-OF-WORK AND MINING

- In crypto-currencies, a cryptographic hash function is used as a proof-of-work.
- A block B is valid if Hash(B) starts with many zeroes.
- More precisely, Hash(B) $\leq$ target, where target is a 256-bit number with many zero MSBs, which changes over time.
- In Nov 2023, the target starts with 77 zero-bits.
    - A random hash is valid with probability 2^{-77}.
    - By changing approximately 77 bits in a block, one obtains a valid block after roughly 2^{77} hash computations = 2^{9} seconds = 9 mins on the Bitcoin network. This process is called mining. Mining gives you 6.25 bitcoins ≈ 200,000€.

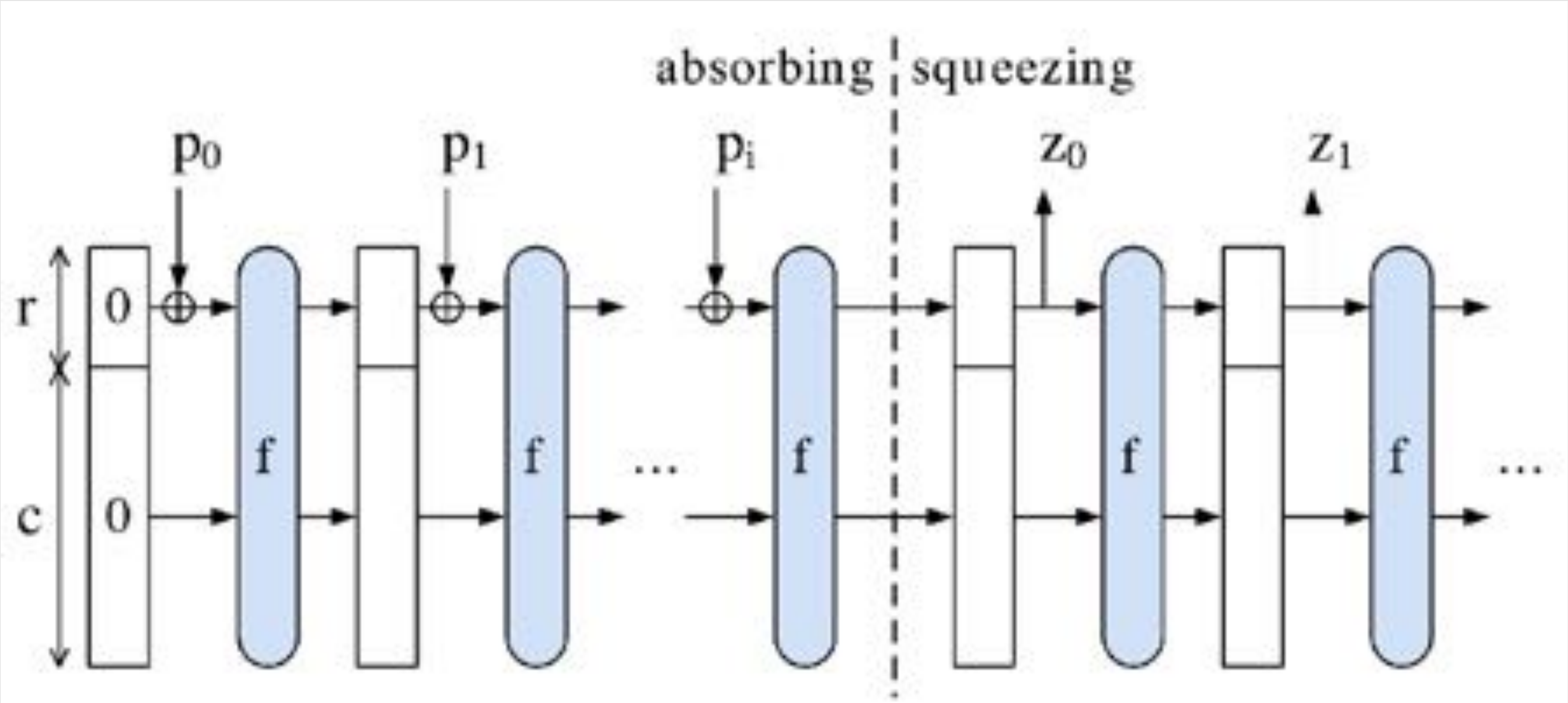
SHA-3

SHA-3 (2008–12)

- The attacks on MD5 and SHA-1 motivated a hash competition for a new US hash standard.
- There were 64 candidates in 2008.
- Then 5 finalists were selected: BLAKE, Grostl, JH, Keccak and Skein.
- In 2012, the NIST chose Keccak, designed by Belgian researchers, like the AES. The standard was released in Aug 2015.

DESIGN OF SHA-3

- Keccak does not use the MD design.
- It is based on the sponge construction.
  - The message is « absorbed » by a sponge
  - The digest is « squeezed out

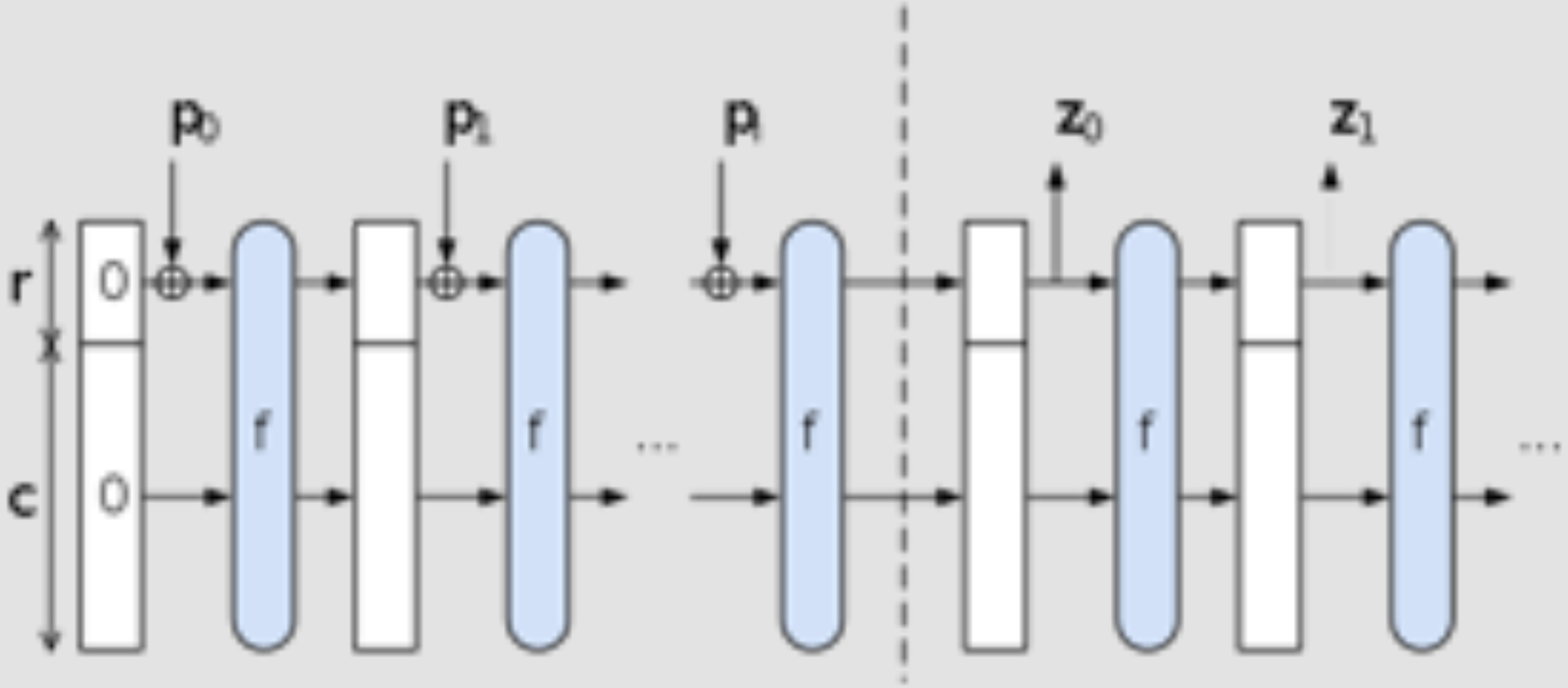


SPONGE 

- a state memory S, initially zero: for SHA-3, 5x5 array of 64-bit words = 1600 bits.
- a function f that permutes or transforms the state memory: for SHA-3, the permutation operates on a 5x5xw array of bits, using parity, bitwise rotation, XOR, non-linear bit operations.
- a padding function

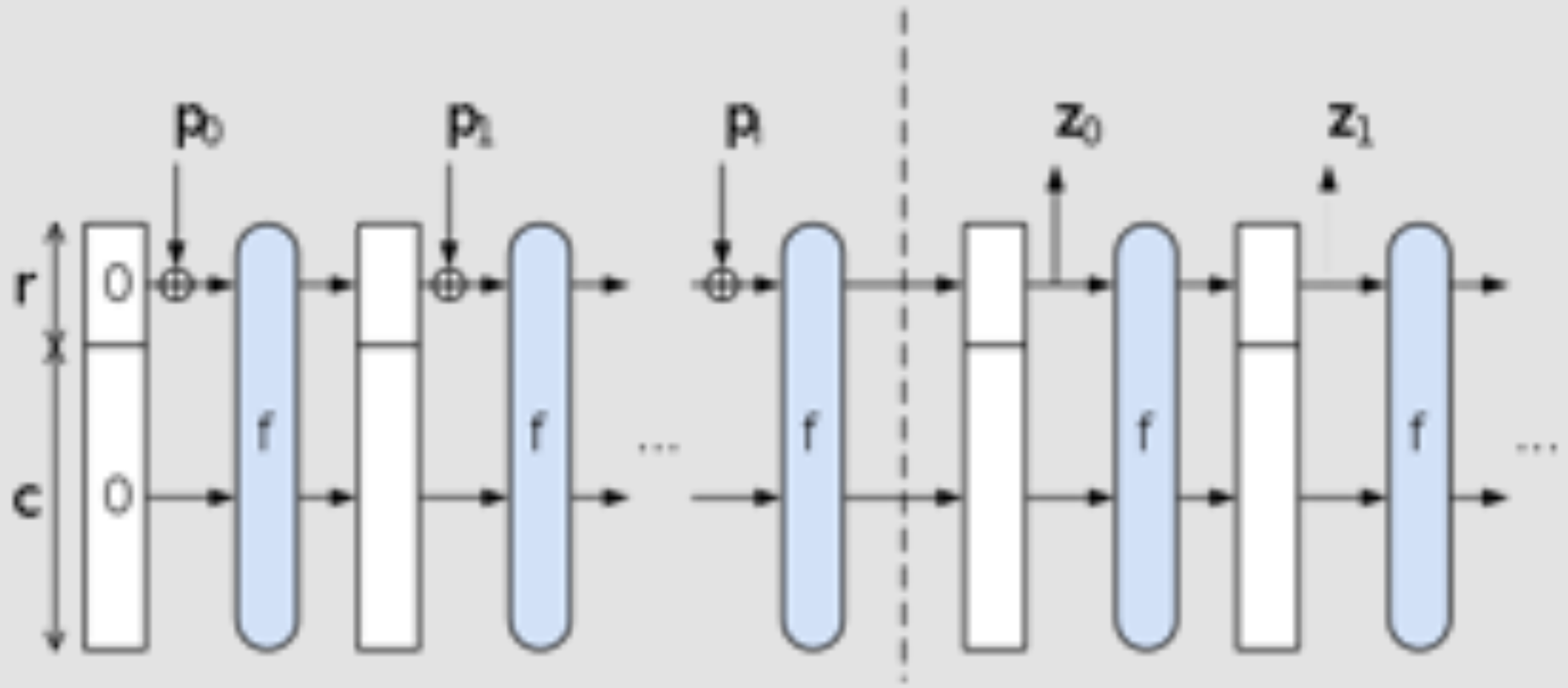
SPONGE ABSORBING

- A subset R of the state S is XORed with the first r-bit block of padded input
- S is replaced by f(S)
- The same subset R of the state S is XORed with the next r-bit block of padded input
- S is replaced by f(S)
- ...



SPONGE SQUEEZING

- The R portion of the state memory is the first r bits of output.
- If more output bits are needed, replace S by f(S), then squeeze R, until enough bits.



SHA-3 BENCHMARKS

- 12.5 cycles/byte on a Core 2.
- Much slower than MD5/SHA-1.
- Will SHA-3 be widely used?

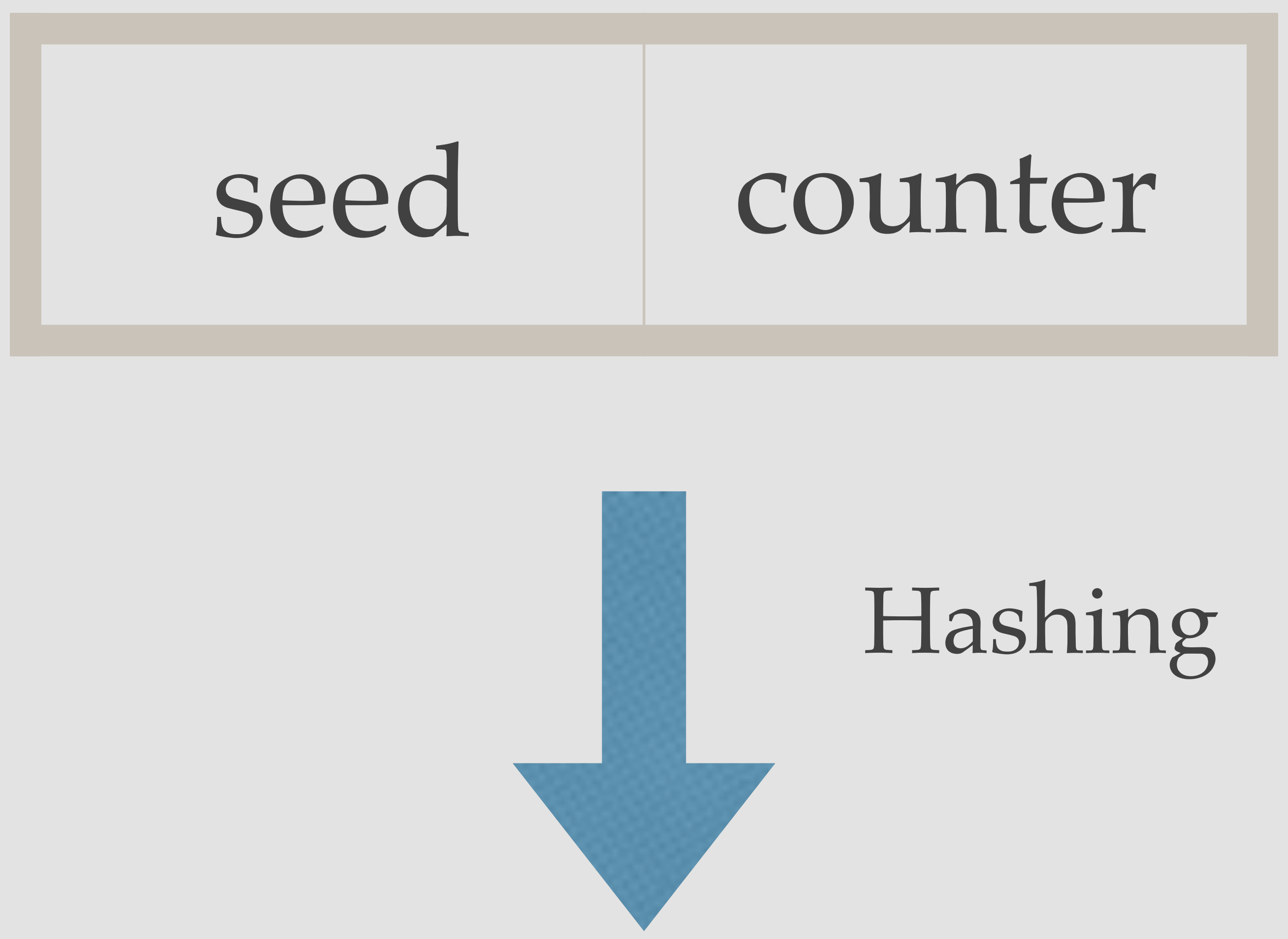
PSEUDO-RANDOM NUMBERS

PRNG

- Input: a short seed. It could be a password, or truly random bits (entropy).
- Output: an arbitrarily long sequence of bits
- Usual requirement: for a random seed, the output should have uniform distribution.
- Cryptographic security: the output should be indistinguishable from a sequence of truly random bits.

EX: PRNG STANDARD IN PKCS

- Let H be a cryptographic hash function, such as SHA-2.

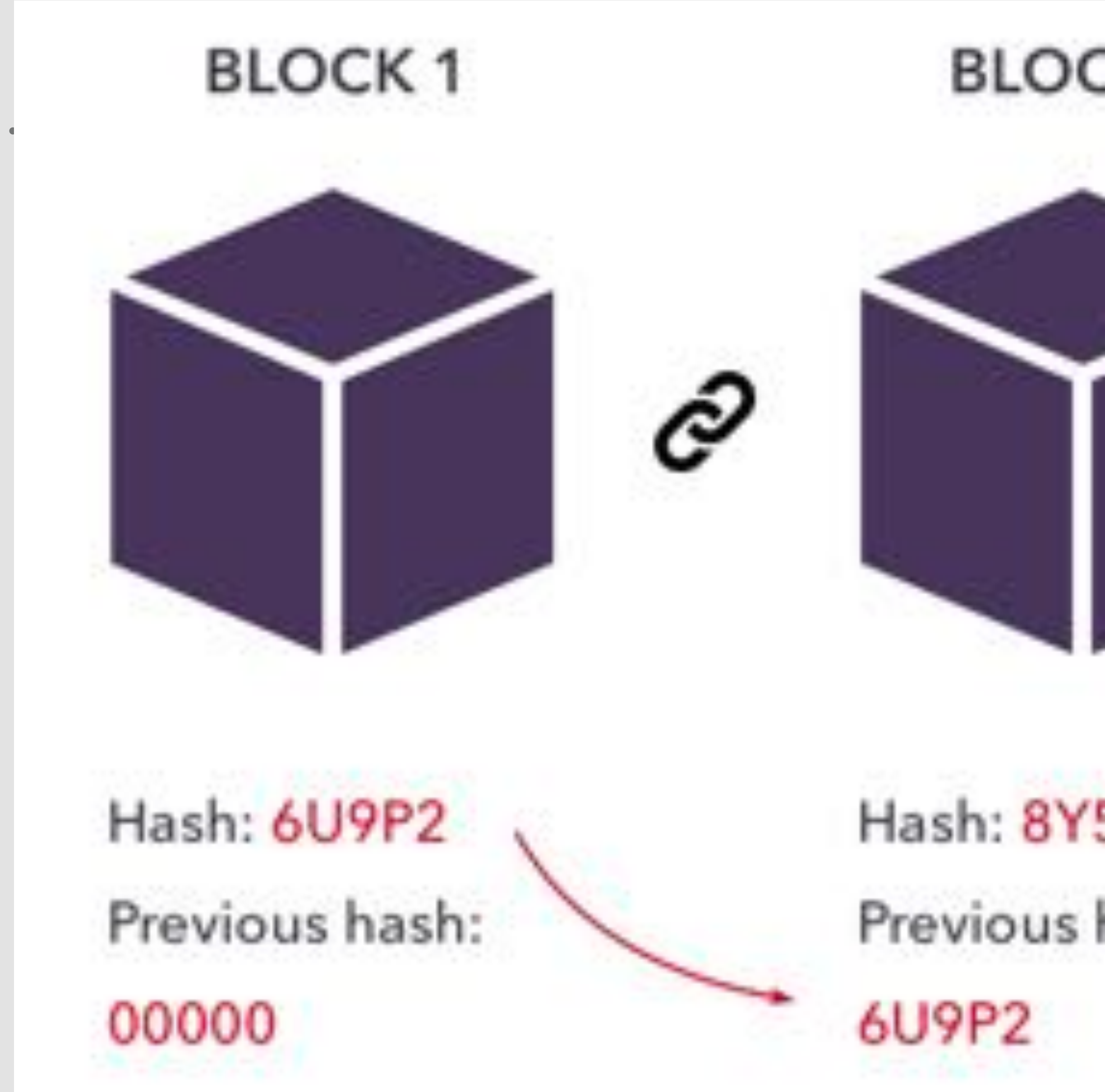


Sequence of pseudo-random bits (256 bits for SHA-2)

CONCLUSION

- Hash functions are the workhorse of cryptography:
  - They are used almost everywhere.
  - They are efficient.
  - But their security is still not very well-understood.

EXAMPLE: BITCOIN



- The blockchain:
  - Blocks are chained together by hashing.
  - Each block contains the hash of the previous block.
- A bitcoin block contains statements of the form:
  - « My name is XYZ, and I certify that I give xxx bitcoins to the person ABC » together with a digital signature.

TAKE AWAY

- Public-key cryptography has no unconditional security.
- It requires to make computational assumptions: it is impossible to recover the secret key (or an equivalent key) from the public key.
- But then what is impossible?

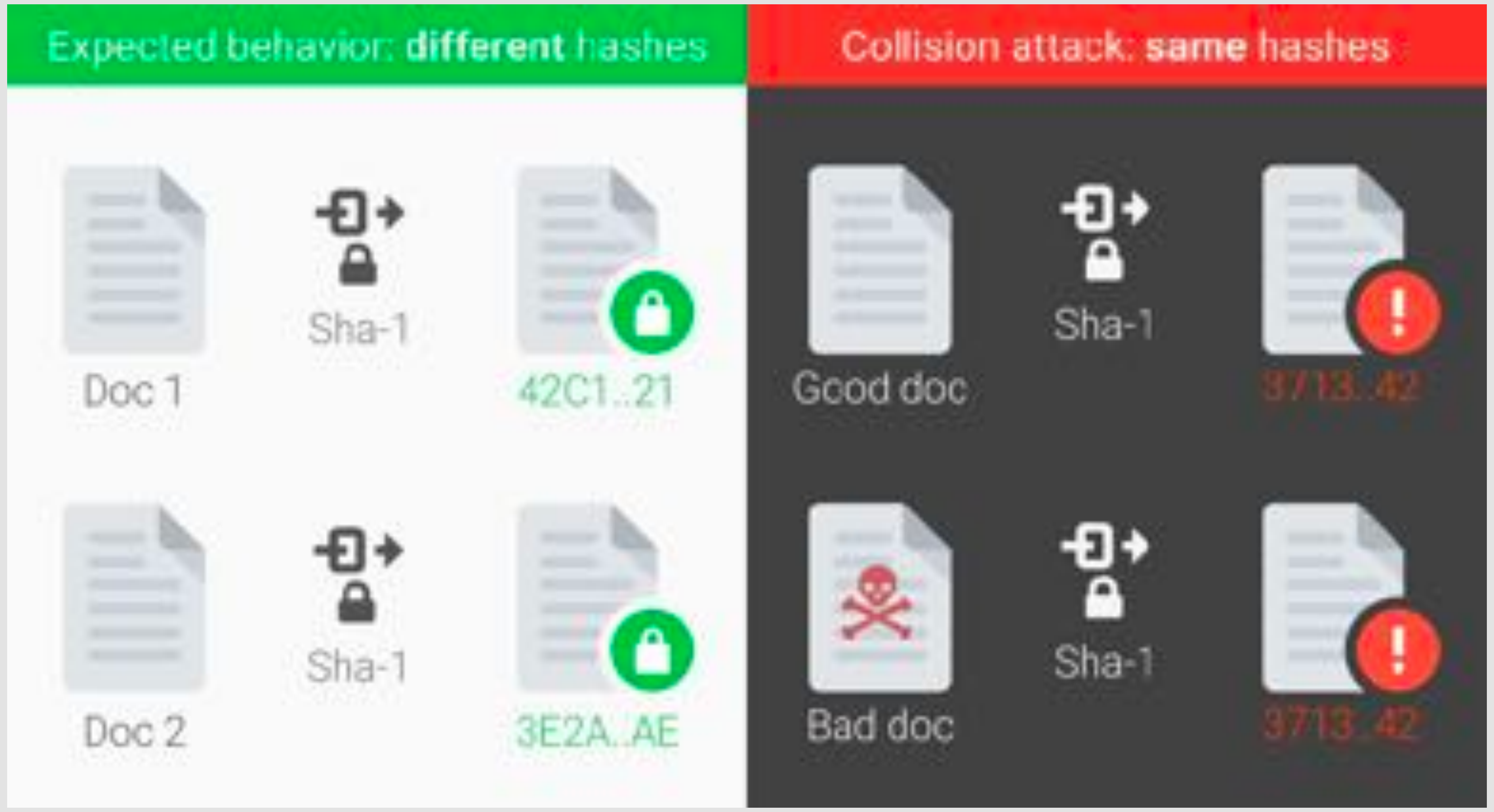
WHAT IS THE LARGEST COMPUTATION EVER?

THE LARGEST (PUBLIC) COMPUTATION EVER

- Duration: 4 years, from 1998 to 2002.
- 2^{64} RC5 encryptions ≈ 2^{74} clock cycles.
- Up to 300,000 PCs used on the Internet.
- But this is much less than Bitcoin computations.

MORE RECENTLY

- In Feb. 2017, an international team (Stevens et al.) found the first SHA-1 collision using approximately 2^{63} hash computations.
- It took 6500 CPU years and 100 GPU years.



BITCOIN POWER

- If D is the current Bitcoin difficulty, mining requires to calculate D x $2^{32}/600$ hashes per second.
    - ➤ In Nov 2023, D≈62,4*10^{12} so #hash/s≈2^{68} and #hash/year≈2^{93}
- D is updated so that mining takes about 10 minutes.