message integrity

Message Authentication:

≠ user autentication

Secure hash-based constructions

Confidentiality ≠ Integrity

➔**Confidentiality**

➔Message hiding

⇨Only the **legitimate destination** should be able **to see** the content of a message

➔**Integrity**

➔Message authenticity

⇨**Nobody** should be able to **modify** a message while in transmission

⇨Only the **legitimate source** should be able to **forge** a message

➔**Integrity-only: often is the only thing you want**

⇨Example: your CV

➔Everybody should be able to read it! (no confidentiality)

➔But only ➔**YOU**← should be able to modify it

Encryption does NOT guarantee integrity!

➔**Example: let's take the best possible cipher**
one time pad – unconditionally secure…

PT	1000 1000 1000 1111	⊕
KEY:	0100 1101 0011 0101	
CT	1100 0101 1011 1010	
⊕		
Δ	0000 1001 0000 0000	
CT'	1100 1100 1011 1010	⊕
KEY:	0100 1101 0011 0101	
PT'	1000 0001 1000 1111	

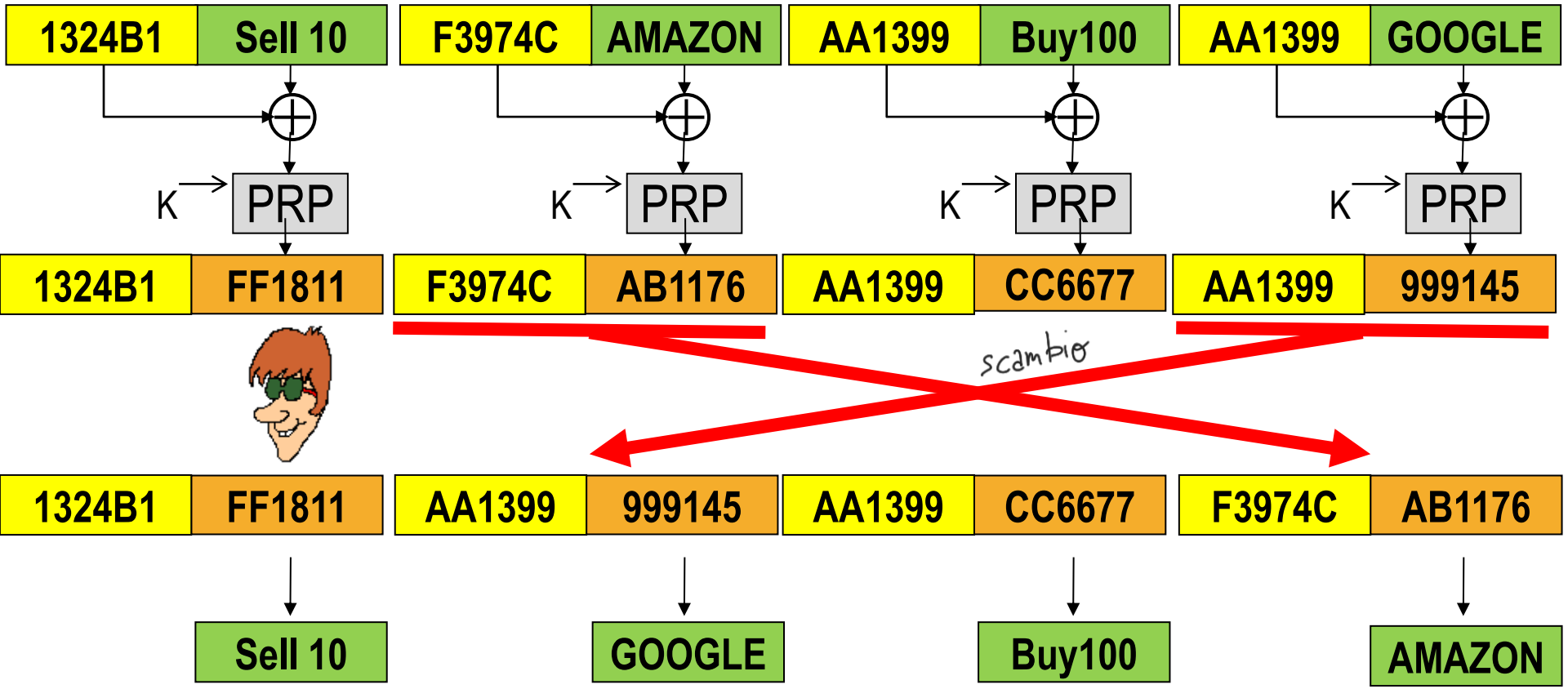
Let's turn 1000 in 0001

Conclusion: Best possible Cipher has zero integrity!!

Encryption does NOT guarantee integrity!

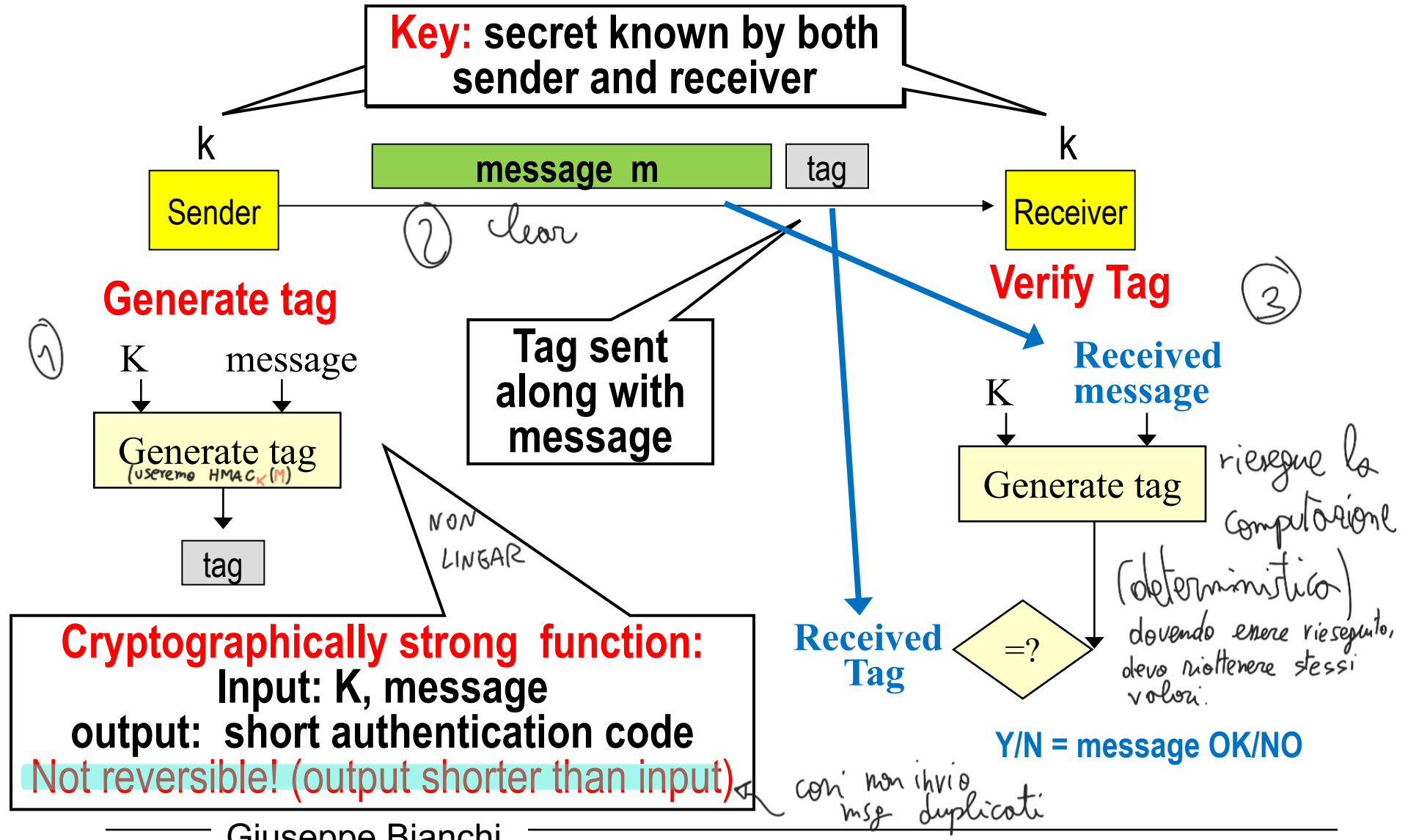
➔ **Example with block cipher** (posso cambiare la semantica)

(ECB with explicit IV per block to make it semantically secure, more later)



VERY different semantics than what meant by transmitter!

Message authentication
(with symmetric key)



Message authentication: what it does NOT protect?

Protects from MITM attacker

message m	TAG

TAG = F(K, m)

Modify
m → m*

But (assumptions on F = cryptographically strong):

1) Sees TAG & m: must NOT be able to «invert» F and get K
2) Cannot change TAG→TAG*=F(K, m*) without knowing K
3) Cannot change m→m* such that F(K,m)==F(K,m*)

message m*	TAG

F(K, m*) = TAG* ≠ TAG
Someone modified m!!

Protects from message spoofing

(metto NUOVO msg nello scambio)

Spoofed message X | TAG = F(K,X)

?????

(assumptions on F = cryptographically strong):
Cannot compute F(K, X) without knowing K

tiro a caso!

Spoofed message X | Invalid TAG

F(K, X) ≠ TAG
Spoofed message!

Does NOT protect from replay attacks

I'm sending the same message twice as I really want to pay 2000$

Pay 1000 $	TAG

Pay 1000 $	TAG

Let's stop. I just want to pay $1000 once.

Pay 1000 $	TAG

NON DISTINGUIBILI!

Ops! This is a valid message!

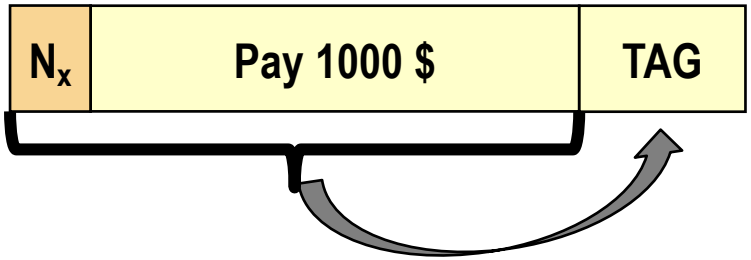
Pay 1000 $	TAG

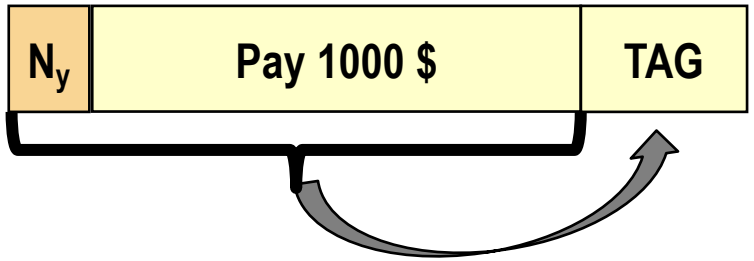
Nah! I'll make you pay twice!!

(numero singolo utilizzo)

Preventing replay attacks: Nonces

- **Replay attacks prevented if ALL messages are different each other!**
- **How to?**
  - ⇨ Application MAY legitimately send two identical messages!





- **Add «nonce» to each packet**
  - ⇨ Nonce=«fresh» information, guaranteed to differ in each message!
  - ⇨ Does not have to remain secret → can be added to the msgin clear
    - →Sequence number
    - →Random number
    - →Time stamp

$F(N_x, MSG, K)$

- **and include it in the TAG computation!**

Test your understanding

➔**Which pros and cons of each type of nonce?**

➔**Sequence number?**

⇨How to manage «reboots» ☺ ?

➔**Random numbers?** (come capisco l'ordine?, dovrei usare DB e fare check)

⇨How RX can control that packet is a fresh one?

➔**Time stamps** (es: cambio orario al telefono)

⇨Who guarantees time? «now» is really «now»?

MAC = Digital Signature?

➔**Digital signature (DS):**

⇨Nobody can modify a digitally signed message (except creator)

➔**Message Authentication Code (MAC):**

⇨Nobody except TX and RX (which have the key) can modify a MAC-authenticated message

➔**Both have an identical purpose!**

⇨Message/data Integrity

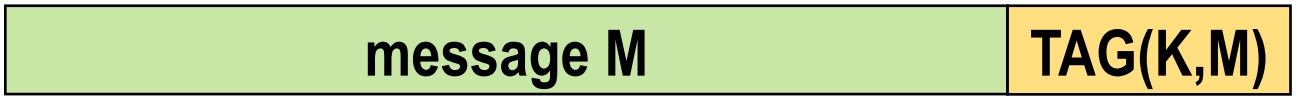
➔**But unlike MAC, DS further guarantees non repudiation (source authentication)**

⇨DS = stronger form of authentication than MAC

➔But requires different crypto ➔ asymmetric, more later

⇨Why this is useful? Legal scenarios, Multicast scenarios

Defining Security for Message Authentication Codes



➔ **Security = Unforgeability**

⇨ Attacker should not be able to create or modify a message

⇨ Implies that attacker should not be able to extract key from pair {M, TAG(K,M)}

➔ **Formally: a game against an attacker model**

⇨ From weaker to stronger: Known Message Attack →
→ Chosen Message Attack → Adaptively Chosen Message Attack

➔ **Attacker is given (may choose) a number of past message/tag pairs**

⇨ (m_1,t_1), (m_2,t_2), (m_3,t_3), …

➔ **Now sees (choses) message m: must be unable to forge tag**

➔ **More specifically:**
probability to forge valid pair must be NEGLIGIBLE

size of tag ?

Test your understanding

definisco 8 bit

len [F (msg grande, K 128 bit] = 8 bit

- 1 byte tags
- No way for attacker to guess tag from msg, beyond pure random choice

da agg al msg

- Is this secure?

Se modifico msg, creo random tag: $\frac{1}{256} \rightarrow 2^8$

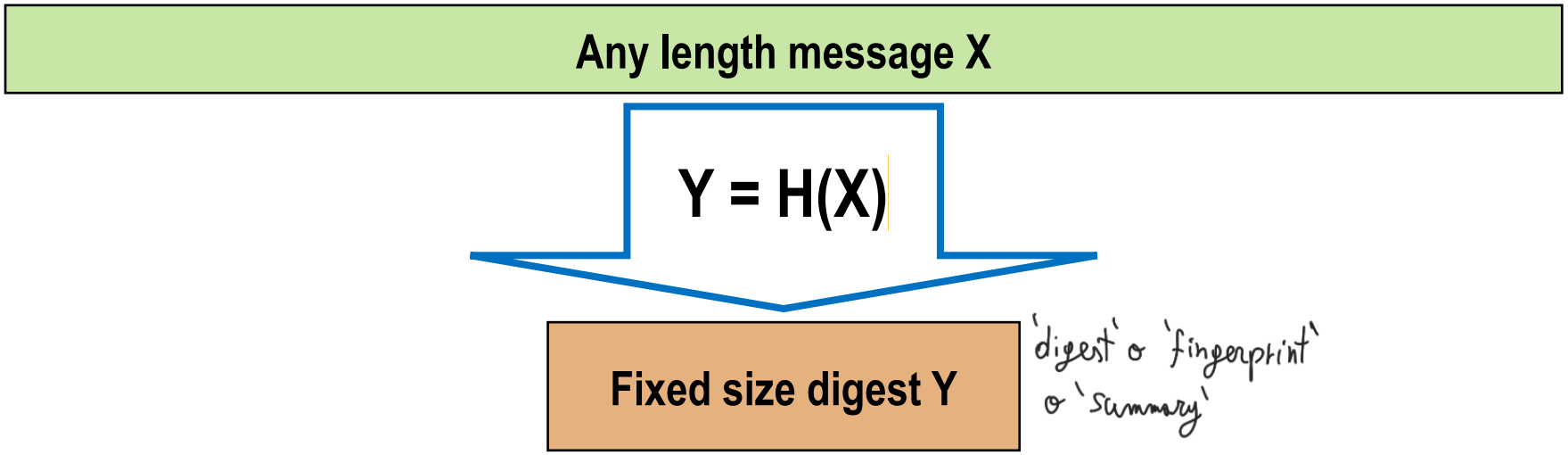
- NO! Probability of guessing = 1/256
  - Not nearly negligible!!
- Note the crucial difference with encryption security definitions!

Which (highly non linear) function can we use?

Hash function? What's this?

What is a Hash function?

(non crittografica)



Arbitrary size X → fixed size Y=H(X)

(from as short as 1 single bit to… any size) → *e.g. exactly 256 bits for SHA256*

Y=H(X): "fingerprint" (short summary) of X

H(*X*) *should be relatively easy to compute for any given X, making both hardware and software implementations practical*

Se modifico N volte il msg, non potro' avere msg diversi e stesso hash dell'originale

(COLLISIONE)

Cryptographic hash functions

Hic ego: laudare igitur eloquentiam et quanta vis sit eius expromere quantamque eis, qui sint eam consecuti, dignitatem afferat, neque propositum nobis est hoc loco neque necessarium. hoc vero sine ulla dubitatione confirmaverim, sive illa arte pariatur aliqua sive exercitatione quadam sive natura, rem unam esse omnium difficillumam. quibus enim ex quinque rebus constare dicitur, earum una quaeque est ars ipsa magna per sese. quare quinque artium concursus maxumarum quantam vim quantamque difficultatem habeat existimari potest.

H[ ]

c6c8258947bffe06ea4a0c8132af337a3c74ec
81d754a96d5a29e3ca7d8ce49d

Hic ego: laudare igitur eloquentiam et quanta vis sit eius expromere quantamque eis, qui sint eam consecuti, dignitatem afferat, neque propositum nobis est hoc loco neque necessarius. hoc vero sine ulla dubitatione confirmaverim, sive illa arte pariatur aliqua sive exercitatione quadam sive natura, rem unam esse omnium difficillumam. quibus enim ex quinque rebus constare dicitur, earum una quaeque est ars ipsa magna per sese. quare quinque artium concursus maxumarum quantam vim quantamque difficultatem habeat existimari potest.

H[ ]

3238ead7fb611463703c47adc4215aa245a1f
1a4a0cea4c11296b466a76bbac4

Fixed size digest
(e.g. SHA-256: 64 hex)

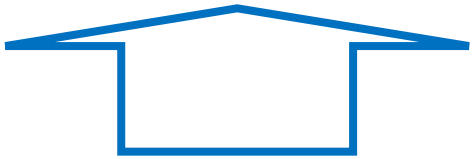
No way for an attacker to either create or purposedly modify/extend/replace initial text so as to obtain original digest!!

Property #1 of a cryptographic hash function

➔ **Preimage resistance (one way)** (reverse difficile)

⇨ Given Y = result of a hash, it is hard to find ANY X such as H(X)=Y

? ? ? ?



2^{256}

Digest: c6c8258947bffe06ea4a0c8132af337a3c74ec
81d754a96d5a29e3ca7d8ce49d

One way: much more than

«non-invertible»

Infinite (!!) messages can generate this digest

but you should NOT be able

(in a reasonable time)

to find ANYONE of such messages!!

La funzione HASH deve resistere a 'forza bruta', il digest deve avere dimensioni elevate.
Se 16 bit → 2^{16} output = 65536 valori. Un PC attuale computa 66 Milioni di Hash/s. NON BASTANO GRANDI DIMENSIONI, conta anche l'implementazione. Se ad esempio 2 bit sono dipendenti, mi serve ancora meno tempo, la metà: $2^{16}/2 = 2^{15}$
Esempio: SHA1 usa 160 bit, ma per 'romperlo' bastano 2^{69} tentativi. Ogni 10 bit, ho un fattore di '1024'. Da 40 bit a 60 bit, in caso forza bruta, diff è 20 bit, quindi se ogni 10 bit = 1024 → 1024·1024 ≈ 1mln. $\frac{2^{160}}{2^{69}}$ ho diff 100 bit, quindi $(1024)^{10}$.

Property #2 of a cryptographic hash function

➔ **Second preimage resistance (weak collision resistance)**

⇨ Given X, it is hard to find another X' such that H(X) = H(X')

Good Message M:

Cari ragazzi non potete modificare questo messaggio se, come prova di quanto sto scrivendo, conservo la sua hash in una mia penna usb

? ? ? ?
Attacker cannot find ANY other message which has the same hash

Digest: 920d7e8cf22725391c2415ade26358f8c099e d37837e1a8911786b0e3684f23c

Stronger property than one-wayness!

there are functions which satisfy property #1 but fail property #2 (raro)

MODULAR EXPONENTIATION (LIMITATO COME SERVE / INPUT ∞, OUT FINITO)

$Y = F(x) = g^x \bmod p$, $1 < \text{digest} < p-1$, p large prime number, g not 0

Graficamente  , se $2^x \bmod (101) = 60$ è possibile trovare 'x', ma complessità esponenziale. RISPETTA priorità #1 ($Y \xrightarrow{\text{hard}} X$)

NON priorità #2 (periodo: p-1, quindi: $g^{x \bmod (p-1)} \bmod p = g^x \bmod p$)

(collision resistant)

Property #3 of a cryptographic hash function

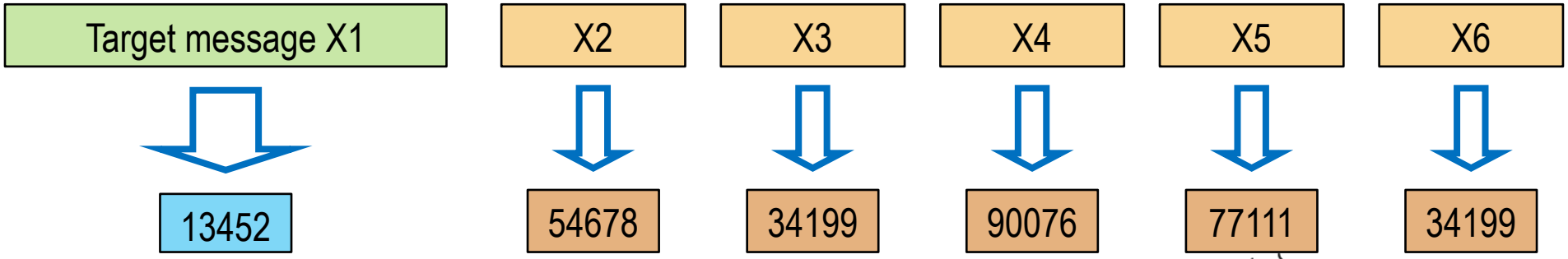
più hard da garantire

➔ **Collision resistance (strong collision resistance)**

⇨ Ii is hard to find two generic X1 and X2 such that H(X1) = H(X2)

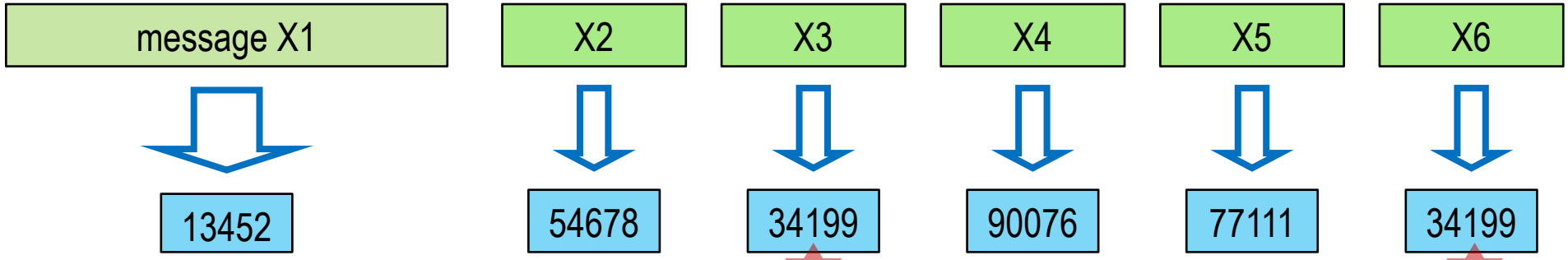
Weak collision resistance: target message

(provo finché ho collisione con X_1)



Strong collision resistance: ANY message

(tra ogni msg !!)



Birthday paradox

23 people are in a class
(22 + myself)

Human being X	→	birthday

(arbitrario) (365 possibilità)

1. **What is the probability that none of you N=22 is born in my same day?**

$$\left(1-\frac{1}{365}\right)^{22}=\left(\frac{364}{365}\right)^{22}=\cdots \quad \textbf{94.1\%}$$

(NON NATO GIORNO "B")

= you have my same birthday in less than 6% cases

giorno X — giorno $Y\neq X$ — giorno $\lambda\neq X\neq\lambda\neq\ldots$

$$1\cdot\left(1-\frac{1}{365}\right)\cdot\left(1-\frac{2}{365}\right)\cdots\left(1-\frac{22}{365}\right)=$$

2. **What is the probability that no two+ of us N=23 are born the same day?**

$$=\frac{365\cdot(365-1)\cdots(365-22)}{365^{23}}=$$

$$=\frac{365!\ /\ (365-23)!}{365^{23}}=\cdots \quad \textbf{49.3\%}$$

In more than 50% cases two of us have the same birthday!!! Not so obious…

IV: 24 bits

Birthday paradox math

(64 bits)

n bit digest → N=2^n **K = n. messages** **p = collision probability**

in funzione di k

$$P(\text{no collision}) = 1 - p = \frac{N!/(N-K)!}{N^K} = \quad \text{come prima!}$$

$$= \frac{N}{N} \cdot \frac{N-1}{N} \cdot \frac{N-2}{N} \cdot \dots \cdot \frac{N-(K-1)}{N} =$$

$$= 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdot \dots \cdot \left(1 - \frac{K-1}{N}\right) =$$

Remember: for x small, 1-x approx e^(-x)

$$= \prod_{i=1}^{K-1} \left(1 - \frac{i}{N}\right) \approx \prod_{i=1}^{K-1} e^{-i/N} = e^{-\sum_{i=1}^{K-1} i \,/\, N}$$

$$= e^{-\frac{K(K-1)/2}{N}} \approx e^{-\frac{K^2}{2N}} = 1 - p$$

$e^{-1/n} \cdot e^{-2/n} = e^{(-\frac{1}{n} - \frac{2}{n})}$

$1 + 2 + 3 + 4 = \frac{5 \cdot 4}{2}$

Birthday paradox math

So, we concluded that: $1 - p \approx e^{-\frac{K^2}{2N}}$

quanti msg servono per avere una probabilità p prefissata.

Let's solve in K: $\log(1-p) \approx -\frac{K^2}{2N} \Rightarrow K = \sqrt{2N} \cdot \sqrt{\log \frac{1}{1-p}}$

50% (p=1/2) collision probability: $K \approx \sqrt{2} \cdot \sqrt{\log 2} \sqrt{N} \approx 1.177\sqrt{N}$

And since $N=2^n$, using n bits: $K \approx 1.177\sqrt{2^n} \approx 2^{n/2}$

Conclusion: n-bit digest → security level is NOT 2^n but $2^{n/2}$

Hash digest size

(truly Random)

➔Must be assessed against birthday paradox!

⇨32 bits (RAND) ➔ 4.3 billion outputs

➔But 50% collision after $2^{16} \sim 60.000$ msg (very little!)

(da "collezionare" per avere stesso output Hash)

⇨128 bits (MD5) ➔ 50% collision after $2^{64} = 1.8 \times 10^{19}$ (weakish today)

» but MUCH worse than this: MD5 broken via cryptoanalysis since 2005

⇨256 bit (SHA256) ➔ 50% collision after $2^{128} = 3.4 \times 10^{38}$, OK today

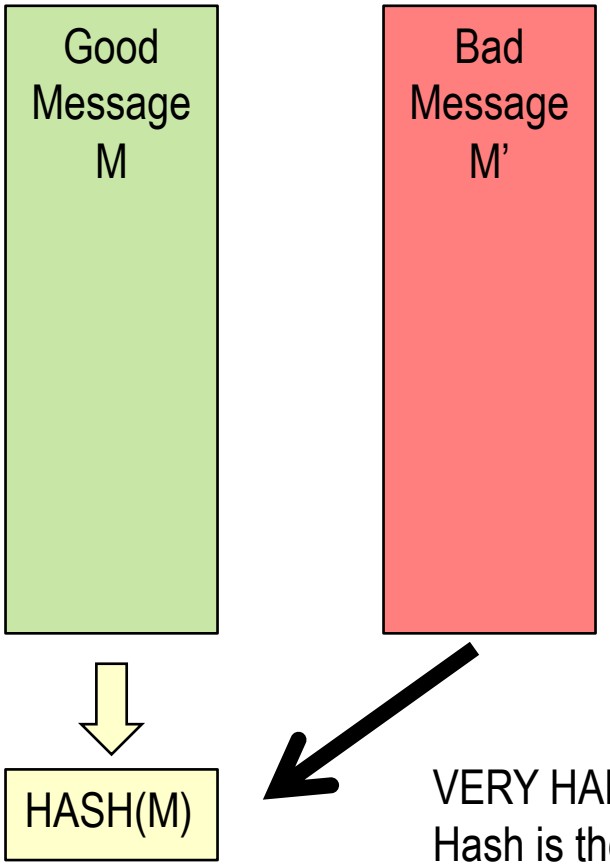
» 3.4×10^{38} ➔ ~ 2000x 4 superenalotto in a row

~ 1 billion centuries of work by whole worldwide bitcoin network (march 2020 ~ 10^{20} H/s)

Back to Message authentication!

Message Authentication Code using hash functions

ingredient 1: good hash



Good crypto HASH:

Today: SHA-2 family (**SHA256**, SHA224, SHA384, SHA512)

Past: SHA1, MD5 (both broken)

next: SHA-3 family (still 224, 256, 384, 512, but new construction)

Most common (used in TLS & Ipsec)

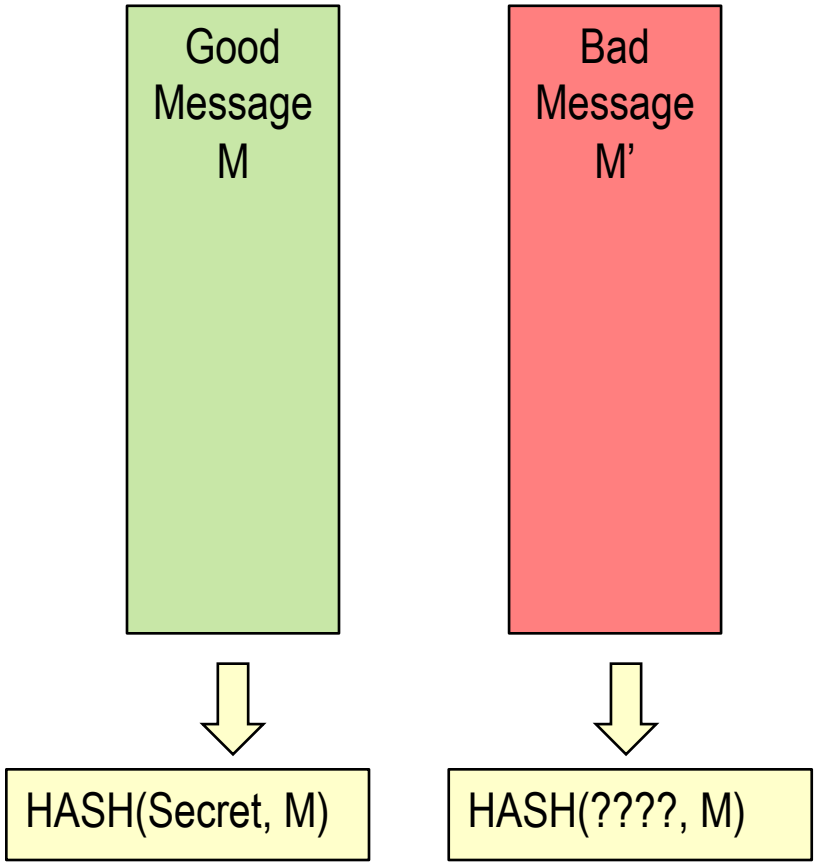
trivial to use (e.g. linux sha256sum command)

(well, apparently ☺ - devil is in details, more later!)

VERY HARD to find message M' whose Hash is the same of M (collision resistant hash)

Message Authentication Code using hash functions

ingredient 2: include secret in the hash



secret = authentication key

Since it is not know by attacker, computationally hard to construct a valid message authentication code

Where to put authentication key in hash?

H(M || S)?

Message M: Dear all, I'd wish to authenticate this message. Let me hash this along with a secret.

Auth key K: 0c8132af337a3c74 ec81d754a96d5a29

Tag: 3596c18d7c6f7b5df983b1edb5f7a0d3 ddb8b1d041d202a1fc9b5fb4916d20ed

H(S || M)?

Auth key K: 0c8132af337a3c74 ec81d754a96d5a29

Message M: Dear all, I'd wish to authenticate this message. Let me hash this along with a secret.

Tag: 12d6aeadec831f55eb5b0556a0f7e52c e241451ad2556aa70c04e7f32ee4b19

DIGEST

Some other fancy way? E.g. «envelope»: H(K||M||K)?

But why you ask? Silly question, why should we care?

We would NOT care IF hash function were a «perfect» one: Random Oracle

➔**«oracle» = black box model**

➔**Random oracle:** = funzione hash perfetta

⇨Given any distinct input X, H(X) = **truly** random value

➔But with repeatibility property:
same input X, same output H(X)

➔**Fact: NO practical hash function can be a random oracle**

⇨Digest cannot be a «truly» random value

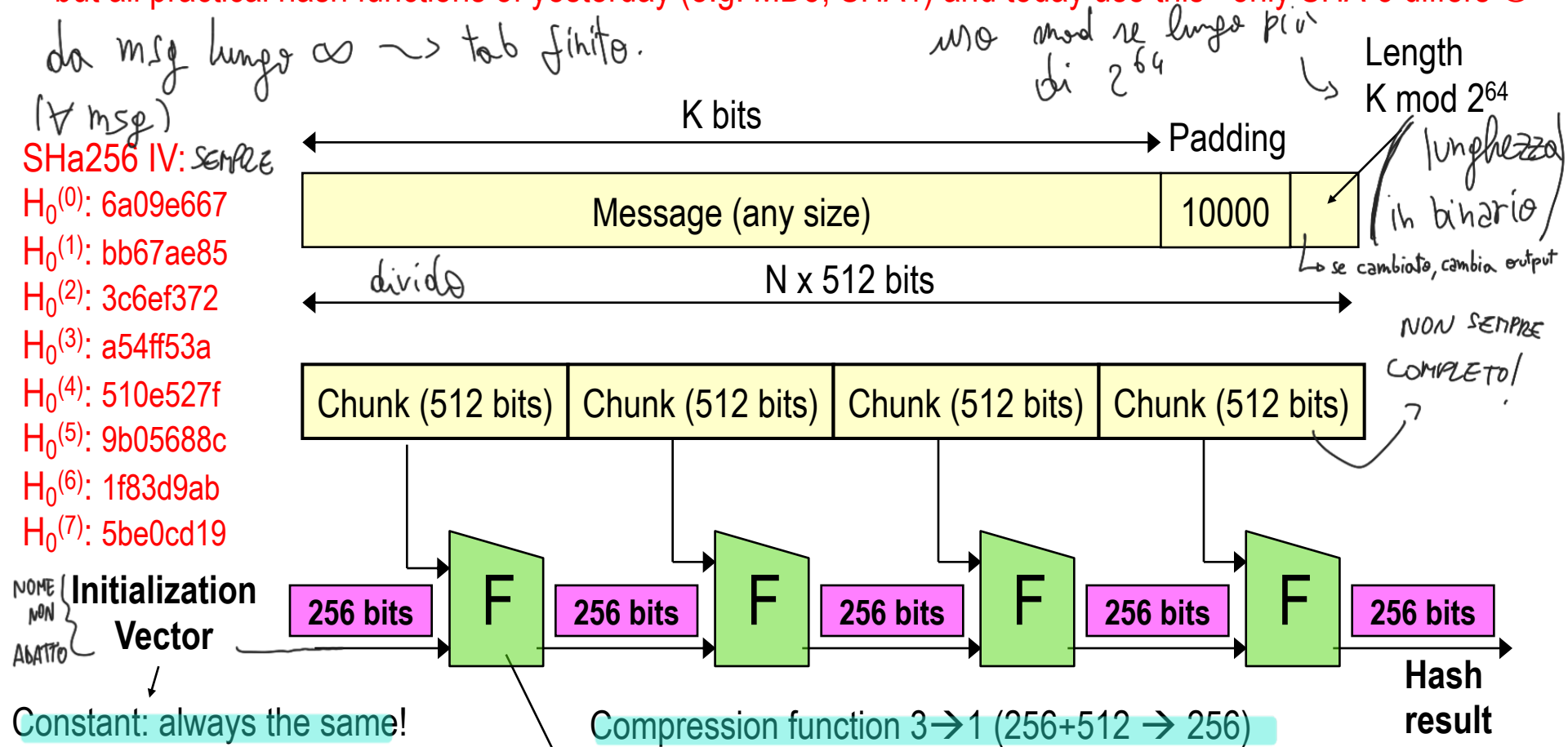
But WE HAVE TO BE CAREFUL! devil is in details…

➔**Matter is: Hash functions are not black boxes!**

Hash constructions: iterative

Merkle-Damgard construction – example: SHA-256

but all practical hash functions of yesterday (e.g. MD5, SHA1) and today use this –only SHA-3 differs ☺

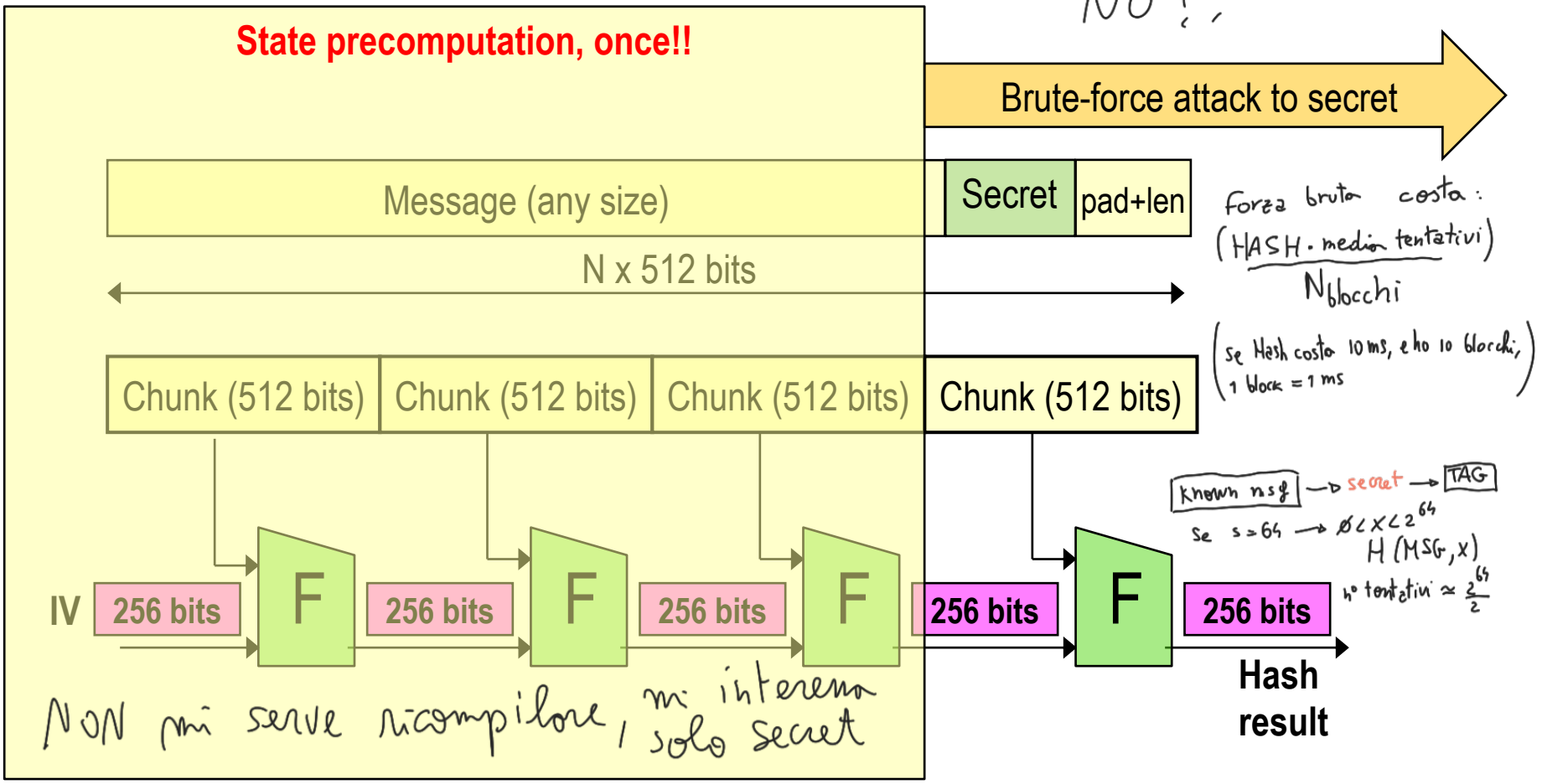


DOVE LO METTO?

Secret Suffix: H(M,S)?

alla fine?

NO!!



forza bruta costa: (HASH · medio tentativi) / $N_{blocchi}$

(se Hash costa 10 ms, e ho 10 blocchi, 1 block = 1 ms)

known msg → secret → TAG

se s = 64 → $0 < x < 2^{64}$ H(MSG, x)

n° tentativi $\approx \frac{2^{64}}{2}$

Weakened security: from N compression blocks per trial, to just 1!

all'inizio, con' devo computare ∀ secret tutto quanto! Ho dei contro? Si

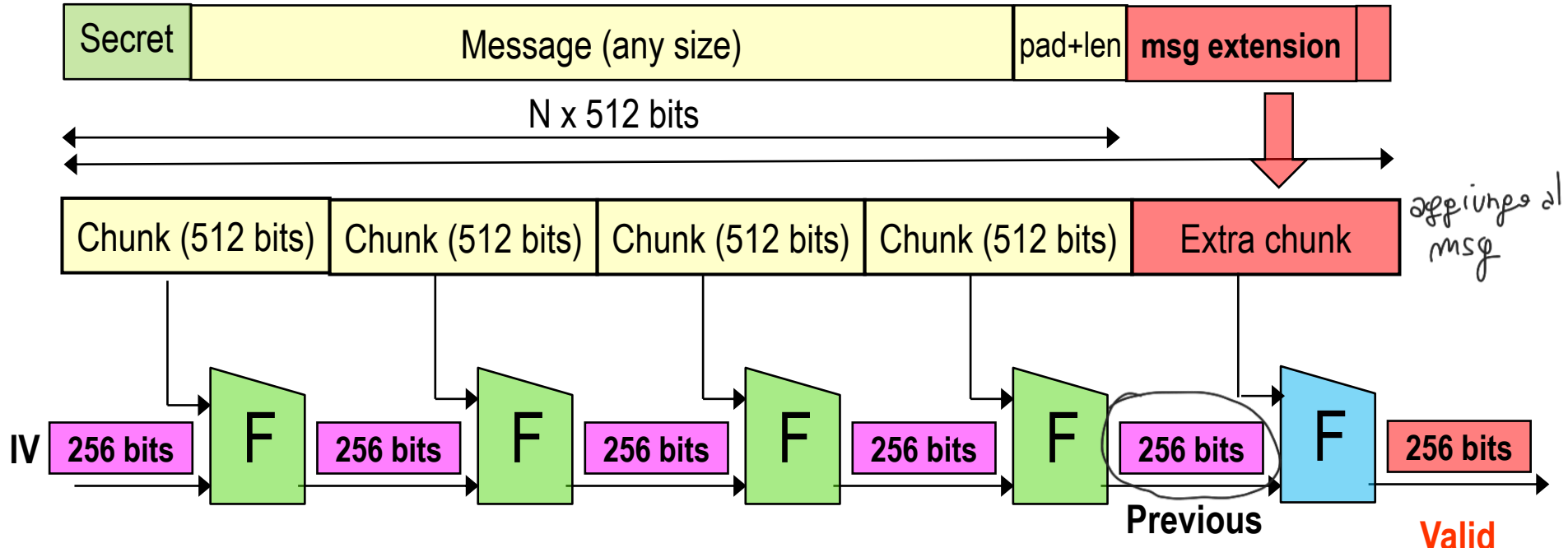
Secret Prefix: H(S,M)? Peggio!

➔ **Expansion attack!**

⇨ Message extension does not require knowledge of secret

➔ Secret already "used" in the (now known) hash state

es: è di 256 bit



Practical attacker might need to still overcome pad/len-strengthening….

But in terms of security, we have forged a new message without knowing the secret key ➔ security broken!

Expansion attack!

- **Trivial to "extend" the message!**
- **Especially critical if secret at the beginning**
  - Example: start from SHA256(k | x), k unknown secret
  - Append y
  - To compute SHA256(k | x | y) use iterative Merkle construction!
    - No need to know k!!!
- **Length (Damgard) strengthening: helps but does not solve the problem**
  - A strong reason to use different constructs (HMAC)
    - http://csrc.nist.gov/pki/HashWorkshop/2005/Nov1_Presentations/Puniya_hashDesign.pdf

basato su Hash

HMAC to the rescue!

- **Lesson learned so far: A secure hash is not enough!**
  - Actually, **a secure hash may yield a completely insecure MAC** (e.g. expansion attack). Serve anche 'segret' ed un modo per combinarli
- **Question: is there a «secure» way to include a secret in a hash?**
  - ⇨ Irrespective of the specific hash construction
  - ⇨ hash should be considered as a black box!
- **Yes: problem solved in 1996: HMAC**
  - Original paper: Bellare, Canetti, Krawczyk, Keying hash functions for message authentication, crypto 96
  - standard: IETF RFC 2104
  - ⇨ Provably secure construction
  - ⇨ «pluggable» hash (you choose/replace it)
  - ⇨ (more or less) same performance of used hash

HMAC construction

$$HMAC_K(M) = H(K^+ \oplus opad \,||\, H(K^+ \oplus ipad \,||\, M))$$

→ K^+ = shared key, but "extended" to block size

» E.g. SHA256 → padded with zeros up to 512 bits

» *Makes sure secret uses the entire first hash chunk*

128 bit secret	00000000000000000000000000000000
512 bit Chunk (SHA256 example)	

⊕

00110110.00110110. 00110110.00110110. 00110110.....

→ opad = 0x36 = 00110110 repeated as needed

→ ipad = 0x5C = 01011100 repeated as needed

» *We would need TWO different (inner and outer) secrets, but since we have only one, let's "pseudo-randomly" derive inner and outer keys from the single key*

(roughly) equivalent to NMAC construction: $H(K_o \,||\, H(K_i \,||\, M))$

HMAC - diagram

• Se NON cambio la chiave l'output non cambio (deterministico)

magic value
(flips half of key bits)

$K^+ \oplus$ ipad

aggiunto

message (arbitrary size)

H(S,M) con N blocchi

Inner key | Msg chunk | Msg chunk | Msg chunk

(stesso INPUT)
deterministic

IV

F | F | F | F

256 bit | 256 bit | 256 bit

modo per randomizzare IV in funzione della key.

Sort of "encrypted" IV for the message!

another magic value
(flips different bits)

$K^+ \oplus$ opad

Output of inner hash
(with secret as prefix)

256 bit

NON PIU' ESPANDIBILE

outer key | Inner hash | pad

Outer message:
still secret prefix but no expansion possible as message is hashed!
(IL MSG è solo 1 blocco!)

IV

F | F

256 bit

HMAC

IMPORTANTE:
IL COSTO è N+2
(N° blocchi)

Just two extra compression blocks
Complexity: from N to N+2

"locks" HMAC: Attacker cannot know such "encrypted" IV

• AUTHENTICATION CODE ⇏ NO Hash
• Devo usare Hash Funct? uso HMAC!

HMAC security

IPOTESI: • pseudo-random output (scorrelati tra loro) ⇒ HASH secure
• collision resistant

➔**quantitatively proven robustness: as secure as its underlying hash is**

⇨see Bellare, Canetti, Krawczyk paper for math proof

⇨we will not dig into this, further

➔**Actually more secure (surprise!)** than HASH

⇨Bellare 2006: collision resistence NOT necessary

➔Pseudorandomness is the only requirement

⇨You might even still use HMAC with MD5 or SHA-1 even if they are both broken - there are algorithms to compute collision

dettagli skippabili

Collisions without HMAC

- **→Collision in hash = collision in MAC**
- **→H(M,S) construction:**
  - ⇨Obvious
    - →find collision on first part of the message, then expand
- **→H(S,M) construction:**
  - ⇨ less obvious, but same problem
    - →Start from H(S,x) → IV
    - →Find collision H*(IV,X1)=H*(IV,X2)
    - →Mi = x | pad(x) | Xi | pad (Xi)
    - →Hence H(S,M1)=H(S,M2)

Take-home

- **In crypto, NEVER use a function which is «more or less» close to your needs**. HASH NON VA BENE per tutto, ∃ funct ∀ scopo!

computare Message Authentication Code ≠ computare fingerprint (HASH)

- **Rather, ONLY use functions SPECIFICALLY designed for your PRECISE need!**

- **Hash: seemed OK, but was NOT!**
  - ⇨We needed to create a new function, HMAC

Summary

- **Confidentiality ≠ integrity**
- **Message authentication (symmetric key)**
  - And formal definition of a secure MAC scheme
- **Replay attacks → MAC not enough → nonces**
- **Ideal vs real world hash functions:**
  - Perfect hash: Random Oracle
  - Real world hash: Iterative (merkle-damgard) constructions
- **How to include key in hash? Non trivial!**
  - Solution: HMAC construction