Digests, Integrity Control and Key Derivation

SIO

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

João Paulo Barraca

Digest Functions

Overview

- Produce a digital **summary** of data called a **message digest**
  - Data is a text or any binary information

- The message digest **length is fixed**
  - independently of the text length
    - Both a 200 bytes and a 200 TB data items will result in a digest with the same length

- The message digest value **strongly depends** on the data

- Two digests are typically **very different**
  - Even if the original data is extremely similar

$\cancel{f^{-1}}$: preimage resistance

$f(x) \cancel{\to} f(x')$: 2nd-preimage resistance

$x \neq x' \to \cancel{f(x) = f(x')}$: collision resistance

Digest Functions

Properties

$$x \xrightarrow{f} y$$
$$\cancel{y \xrightarrow{f^{-1}} x}$$

- Preimage resistance
  - Given a digest, it is unfeasible to find an original text producing it

    Compute f^{-1} = X
  - That is: we cannot go back from a digest to the data (we cannot "decrypt" it)

we cannot generate a new valid one

$f(x) \cancel{\to} \underset{=\text{valid}}{f(x')}$

- 2nd-preimage resistance
  - Given a text, it is unfeasible to find another one with the same digest
  - That is: if we **have a text**, we cannot find another one with the same digest
- Collision resistance $\underline{f(x) \neq f(x')}$, $\underline{x \neq x'}$
  - It is unfeasible to find any two texts with the same digest
  - That is: given two unique texts, they will result **in a different digest**
    - Relates to the Birthday paradox: Collision probability P = $2^{n/2}$ where the typical *n* is >=256

Digest Functions

Lets check: Size independence

- Considering the similar, yet different texts:
  - T1: “`Hello User_A!`”
  - T2: “`Hello User_XPTO! Welcome to this lecture`”

- Different algorithms will create digests with different lengths, but **independent** from the dimension of the text
  - MD5 (128 bits):
    - T1: `70df836fdaf02e0dfc990f9139762541`
    - T2: `18f12f09c45d880ce738afe4780c2f3e`
  - SHA-1 (160 bits):
    - T1: `f591aa1eabcc97fb39c5f422b370ddf8cb880fde`
    - T2: `622f7832e204f2d70161cf42480c4bf0f13e7324`
  - SHA-256 (256 bits):
    - T1: `9649d8c0d25515a239ec8ec94b293c8868e931ad318df4ccd0dffd67aff89905`
    - T2: `6453be3f643d0a7e9b5890eed76bb63df8b6b071b30d5f97269a530c289b9839`

Digest Functions

Lets check: Content dependency

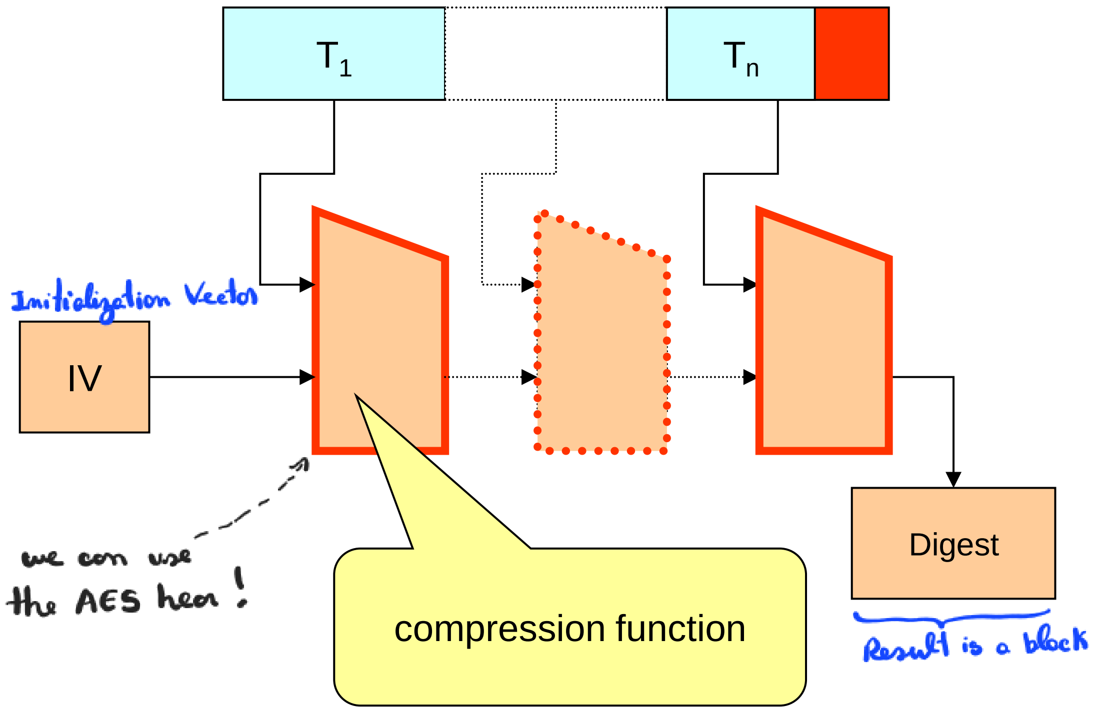
- Considering the similar, yet different texts (1 bit difference 'B' -> 'C'):
  - T1: "`Hello User_B!`", `[0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20, 0x55, 0x73, 0x65, 0x72, 0x5f,` **`0x42`**`, 0x21]`
  - T2: "`Hello User_C!`", `[0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20, 0x55, 0x73, 0x65, 0x72, 0x5f,` **`0x43`**`, 0x21]`

- A small difference in the text (1 bit) results in a **completely different digest**
  - MD5:
    - T1: `c32e0f62a7c9c815063d373acac80c37`
    - T2: `324a1bfc3041259480c6ad164cf0529f`
  - SHA-1:
    - T1: `bab31eb62f961266758524071a7ad8221bc8700b`
    - T2: `bd758d82899d132cd2af66dc3402b948d98de62d`
  - SHA-256:
    - T1: `e663a01d3bec4f35a470aba4baccece79bf484b5d0bffa88b59a9bb08707758a`
    - T2: `69f78345da90c6b8d4785b769cd6ae09e0531716fe5f5a392fde1bdc70a2bb7d`

Digest Functions

Approaches

- Merkle-Damgård construction
  - Collision-resistant, one-way compression functions
    - Can be a block cipher!
  - Iterative compression
  - Length padding
  - Digest size is the last block
  - Can be resumed!
    - Digest is the state at T_n
  - Algorithms: MD5, SHA1, SHA2

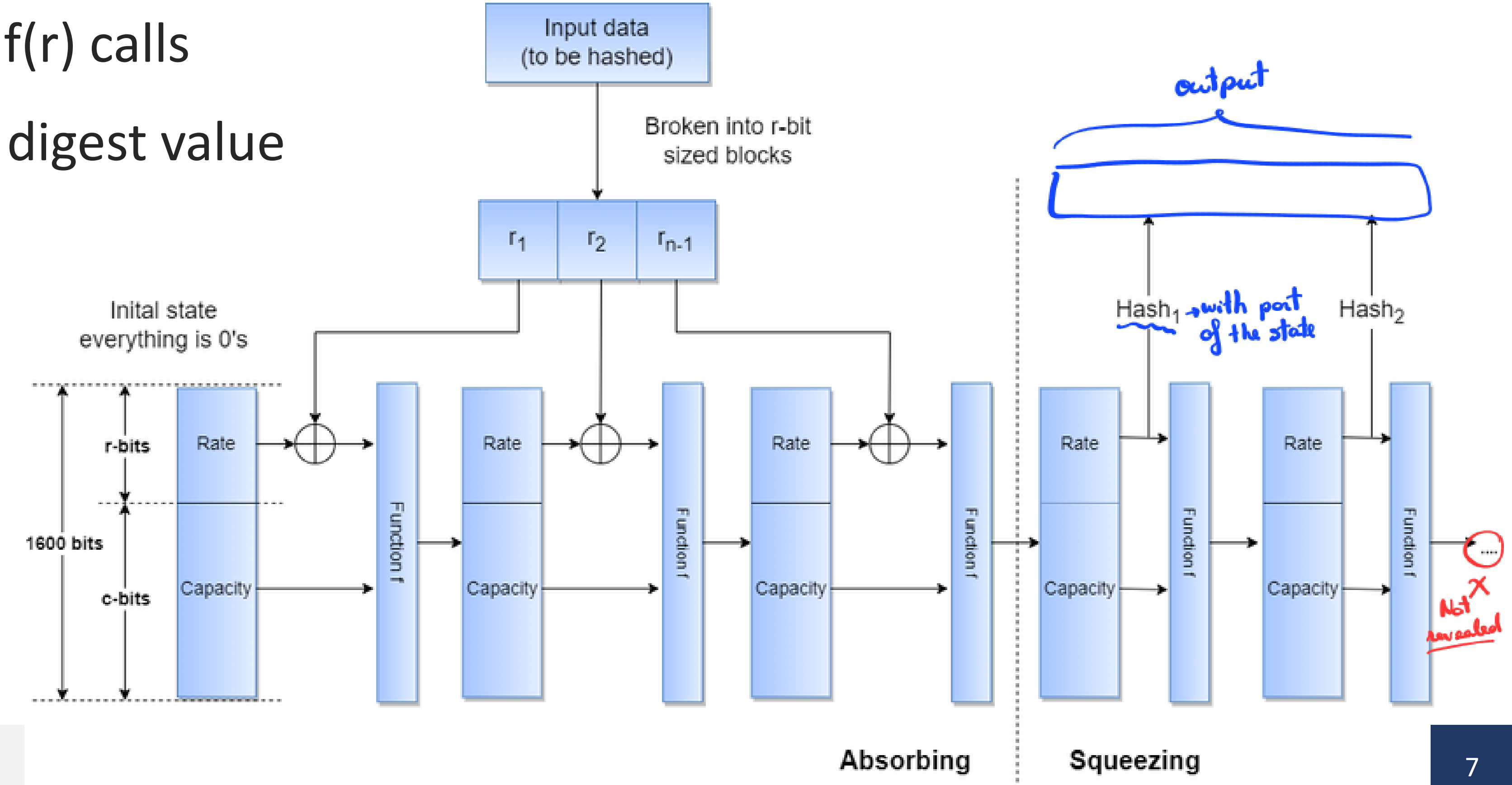


Digest Functions

Approaches

- Sponge functions
  - Data split in *r* sized blocks
  - Absorbing phase: chained f(r) calls
  - Squeezing: extract bits for digest value
  - Algorithms: SHA3

This is a best one ↓

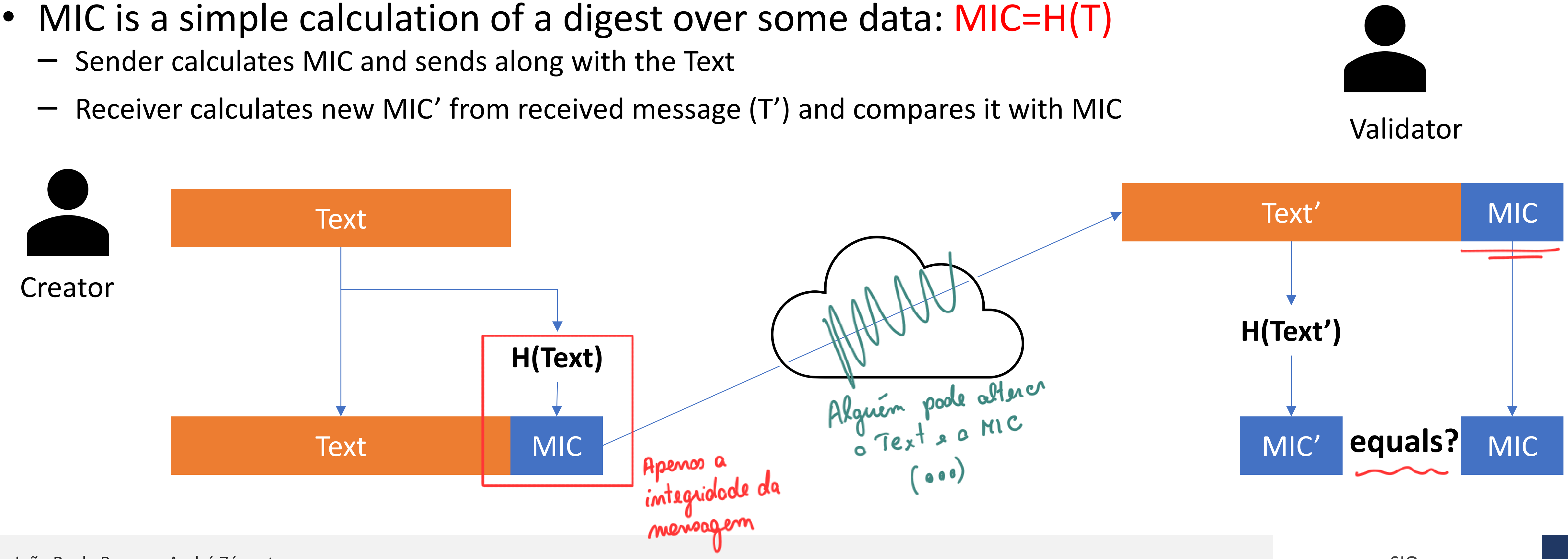


Message Integrity Code (MIC)

Integrity!

- Provide the capability to detect **arbitrary** changes to data
  - Communication/storage errors from a random process or without integrity control
  - Humans/Attackers can change the Text and calculate a new MIC! PROBLEM

- MIC is a simple calculation of a digest over some data: MIC=H(T)
  - Sender calculates MIC and sends along with the Text
  - Receiver calculates new MIC’ from received message (T’) and compares it with MIC



Validator

Creator

Text

H(Text)

Text | MIC

Apenas a integridade da mensagem

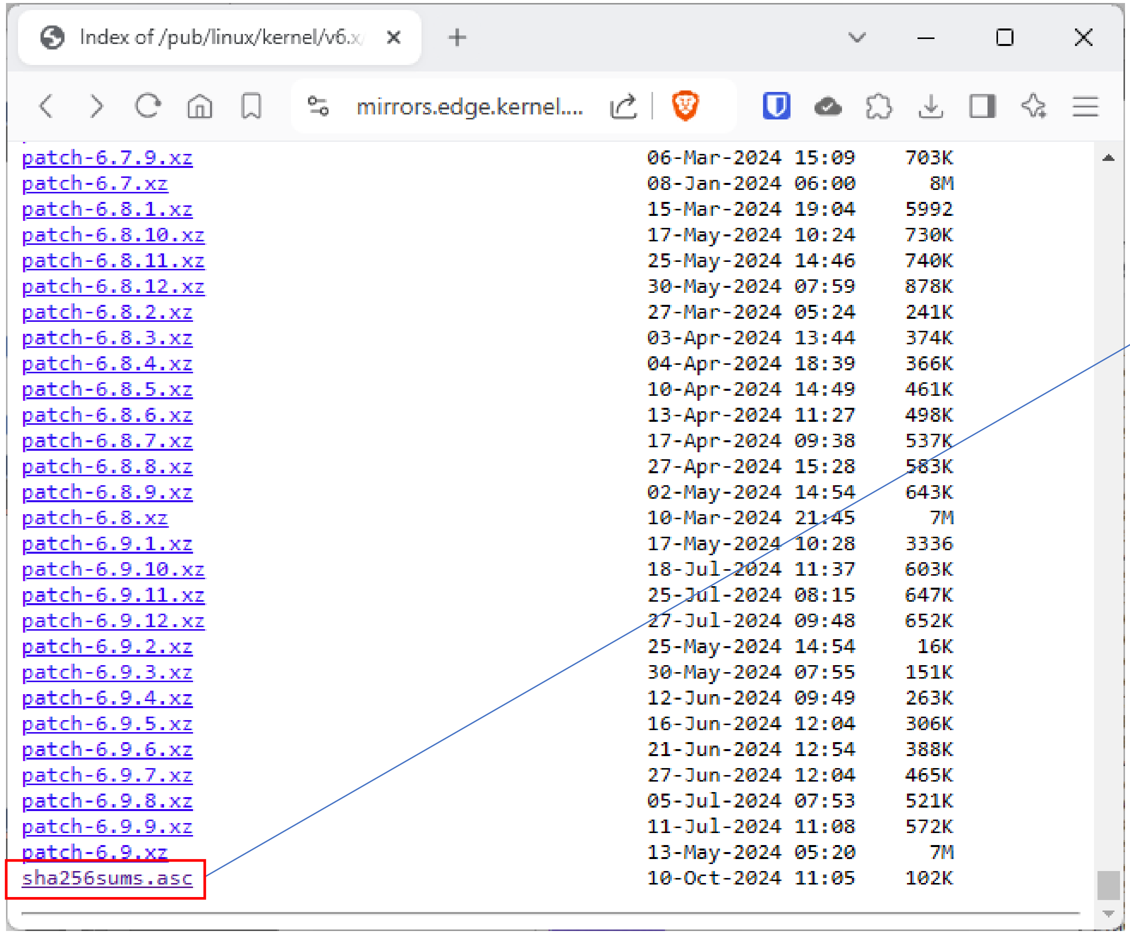
Alguém pode alterar o Text e o MIC (...)

Text’ | MIC

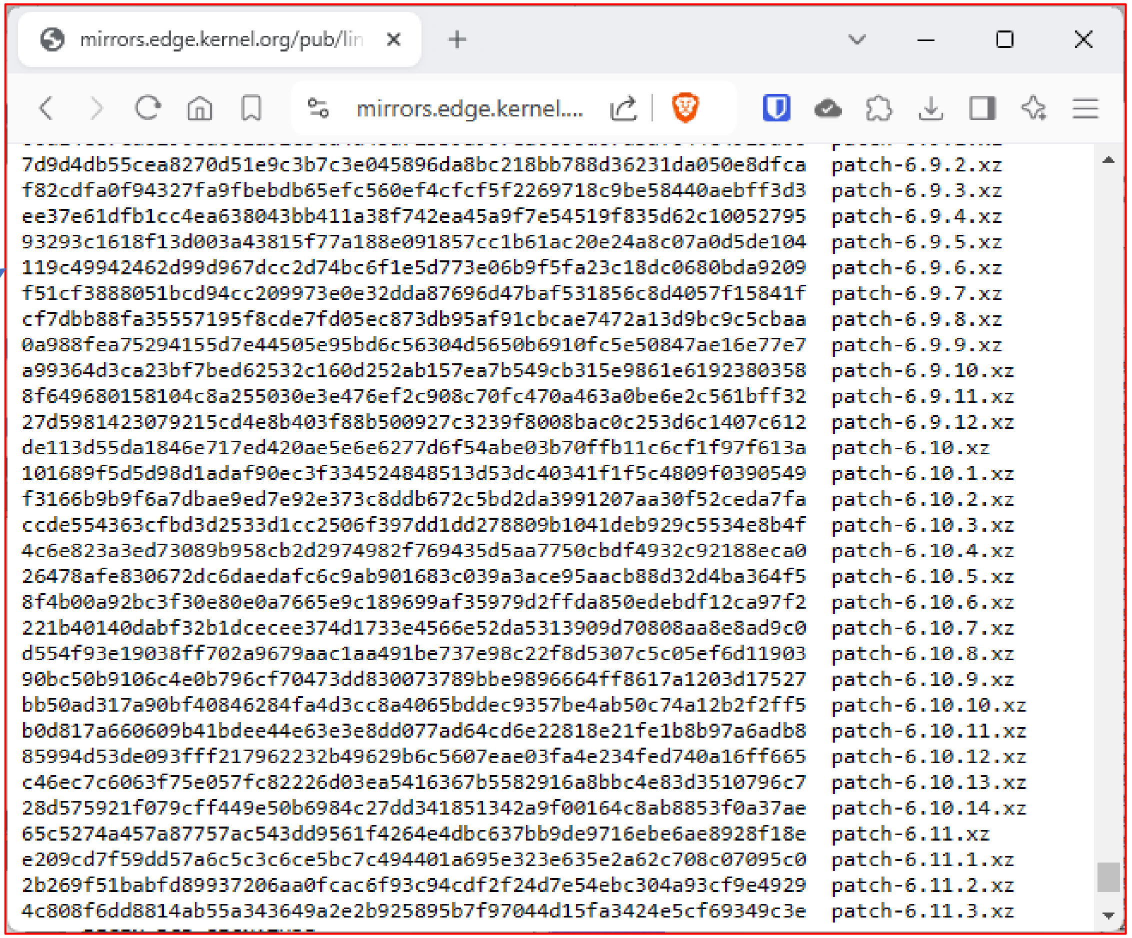
H(Text’)

MIC’ **equals?** MIC

Example usage at kernel.org to validate file integrity



```
patch-6.7.9.xz                                     06-Mar-2024 15:09    703K
patch-6.7.xz                                       08-Jan-2024 06:00      8M
patch-6.8.1.xz                                     15-Mar-2024 19:04    5992
patch-6.8.10.xz                                    17-May-2024 10:24    730K
patch-6.8.11.xz                                    25-May-2024 14:46    740K
patch-6.8.12.xz                                    30-May-2024 07:59    878K
patch-6.8.2.xz                                     27-Mar-2024 05:24    241K
patch-6.8.3.xz                                     03-Apr-2024 13:44    374K
patch-6.8.4.xz                                     04-Apr-2024 18:39    366K
patch-6.8.5.xz                                     10-Apr-2024 14:49    461K
patch-6.8.6.xz                                     13-Apr-2024 11:27    498K
patch-6.8.7.xz                                     17-Apr-2024 09:38    537K
patch-6.8.8.xz                                     27-Apr-2024 15:28    583K
patch-6.8.9.xz                                     02-May-2024 14:54    643K
patch-6.8.xz                                       10-Mar-2024 21:45      7M
patch-6.9.1.xz                                     17-May-2024 10:28    3336
patch-6.9.10.xz                                    18-Jul-2024 11:37    603K
patch-6.9.11.xz                                    25-Jul-2024 08:15    647K
patch-6.9.12.xz                                    27-Jul-2024 09:48    652K
patch-6.9.2.xz                                     25-May-2024 14:54     16K
patch-6.9.3.xz                                     30-May-2024 07:55    151K
patch-6.9.4.xz                                     12-Jun-2024 09:49    263K
patch-6.9.5.xz                                     16-Jun-2024 12:04    306K
patch-6.9.6.xz                                     21-Jun-2024 12:54    388K
patch-6.9.7.xz                                     27-Jun-2024 12:04    465K
patch-6.9.8.xz                                     05-Jul-2024 07:53    521K
patch-6.9.9.xz                                     11-Jul-2024 11:08    572K
patch-6.9.xz                                       13-May-2024 05:20      7M
sha256sums.asc                                     10-Oct-2024 11:05    102K
```

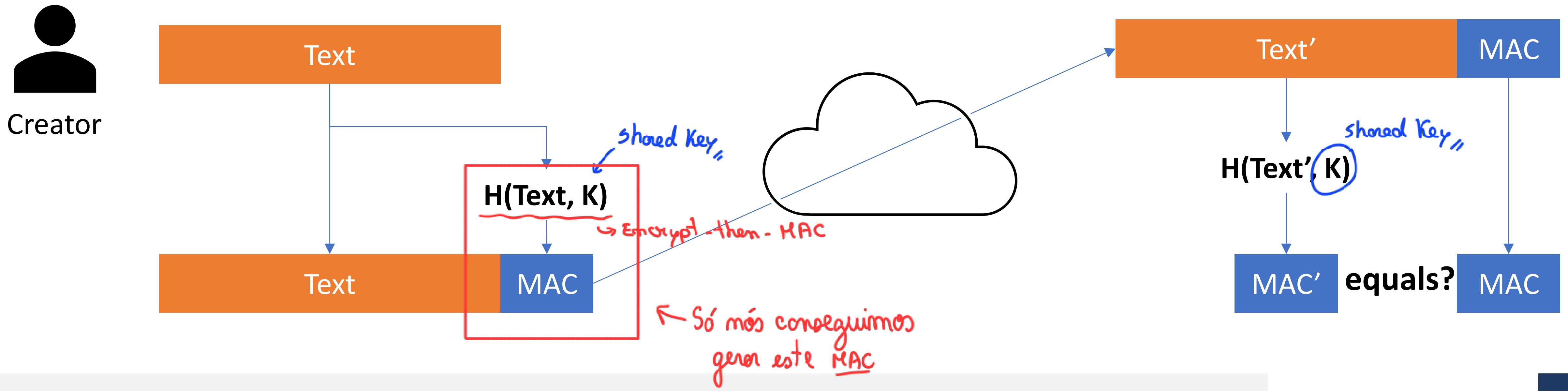


```
7d9d4db55cea8270d51e9c3b7c3e045896da8bc218bb788d36231da050e8dfca  patch-6.9.2.xz
f82cdfa0f94327fa9fbebdb65efc560ef4cfcf5f2269718c9be58440aebff3d3  patch-6.9.3.xz
ee37e61dfb1cc4ea638043bb411a38f742ea45a9f7e54519f835d62c10052795  patch-6.9.4.xz
93293c1618f13d003a43815f77a188e091857cc1b61ac20e24a8c07a0d5de104  patch-6.9.5.xz
119c49942462d99d967dcc2d74bc6f1e5d773e06b9f5fa23c18dc0680bda9209  patch-6.9.6.xz
f51cf3888051bcd94cc209973e0e32dda87696d47baf531856c8d4057f15841f  patch-6.9.7.xz
cf7dbb88fa35557195f8cde7fd05ec873db95af91cbcae7472a13d9bc9c5cbaa  patch-6.9.8.xz
0a988fea75294155d7e44505e95bd6c56304d5650b6910fc5e50847ae16e77e7  patch-6.9.9.xz
a99364d3ca23bf7bed62532c160d252ab157ea7b549cb315e9861e6192380358  patch-6.9.10.xz
8f649680158104c8a255030e3e476ef2c908c70fc470a463a0be6e2c561bff32  patch-6.9.11.xz
27d5981423079215cd4e8b403f88b500927c3239f8008bac0c253d6c1407c612  patch-6.9.12.xz
de113d55da1846e717ed420ae5e6e6277d6f54abe03b70ffb11c6cf1f97f613a  patch-6.10.xz
101689f5d5d98d1adaf90ec3f334524848513d53dc40341f1f5c4809f0390549  patch-6.10.1.xz
f3166b9b9f6a7dbae9ed7e92e373c8ddb672c5bd2da3991207aa30f52ceda7fa  patch-6.10.2.xz
ccde554363cfbd3d2533d1cc2506f397dd1dd278809b1041deb929c5534e8b4f  patch-6.10.3.xz
4c6e823a3ed73089b958cb2d2974982f769435d5aa7750cbdf4932c92188eca0  patch-6.10.4.xz
26478afe830672dc6daedafc6c9ab901683c039a3ace95aacb88d32d4ba364f5  patch-6.10.5.xz
8f4b00a92bc3f30e80e0a7665e9c189699af35979d2ffda850edebdf12ca97f2  patch-6.10.6.xz
221b40140dabf32b1dcecee374d1733e4566e52da5313909d70808aa8e8ad9c0  patch-6.10.7.xz
d554f93e19038ff702a9679aac1aa491be737e98c22f8d5307c5c05ef6d11903  patch-6.10.8.xz
90bc50b9106c4e0b796cf70473dd830073789bbe9896664ff8617a1203d17527  patch-6.10.9.xz
bb50ad317a90bf40846284fa4d3cc8a4065bddec9357be4ab50c74a12b2f2ff5  patch-6.10.10.xz
b0d817a660609b41bdee44e63e3e8dd077ad64cd6e22818e21fe1b8b97a6adb8  patch-6.10.11.xz
85994d53de093fff217962232b49629b6c5607eae03fa4e234fed740a16ff665  patch-6.10.12.xz
c46ec7c6063f75e057fc82226d03ea5416367b5582916a8bbc4e83d3510796c7  patch-6.10.13.xz
28d575921f079cff449e50b6984c27dd341851342a9f00164c8ab8853f0a37ae  patch-6.10.14.xz
65c5274a457a87757ac543dd9561f4264e4dbc637bb9de9716ebe6ae8928f18e  patch-6.11.xz
e209cd7f59dd57a6c5c3c6ce5bc7c494401a695e323e635e2a62c708c07095c0  patch-6.11.1.xz
2b269f51babfd89937206aa0fcac6f93c94cdf2f24d7e54ebc304a93cf9e4929  patch-6.11.2.xz
4c808f6dd8814ab55a343649a2e2b925895b7f97044d15fa3424e5cf69349c3e  patch-6.11.3.xz
```

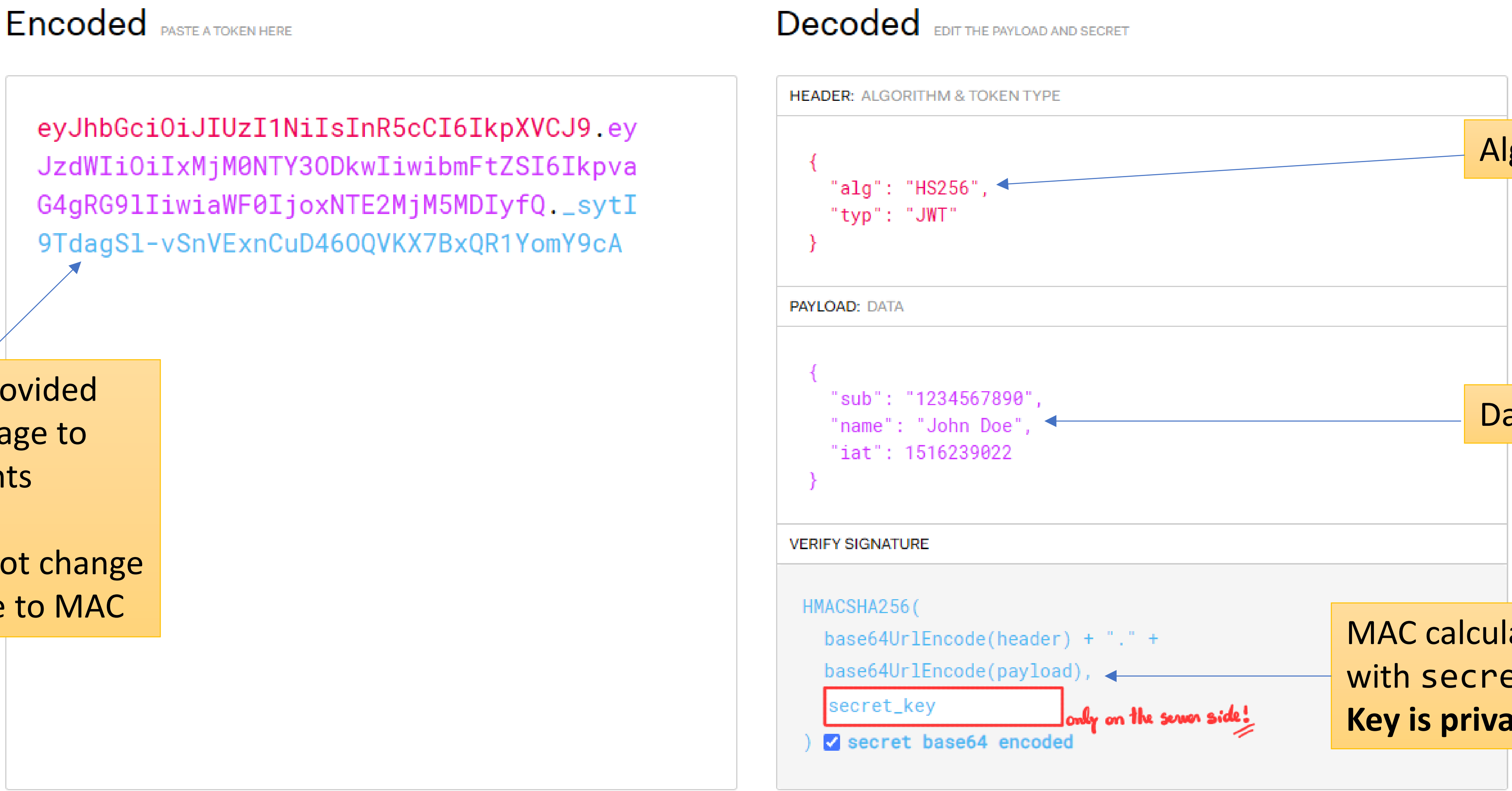
Message Authentication Code (MAC)

- Provide the capability to detect **deliberate** changes to data
  - Any change to data, even if from attackers!

- MAC is a keyed calculation of a digest over some data: M~~I~~AC=H(T, **K**)
  - Parties agree with Key K, which is kept private to participants
  - Sender calculates **MAC** using **K** and sends along with the **Text**
  - Receiver calculates new MAC from received message (T') and **K** and compares it with MAC



Example usage in JWT



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ._sytI
9TdagSl-vSnVExnCuD46OQVKX7BxQR1YomY9cA
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret_key
) ☑ secret base64 encoded
```

only on the server side!

Cookie provided in webpage to Clients

Clients cannot change Cookie due to MAC

Algorithm

Data in cookie

MAC calculated with `secret_key`.
Key is private to server

https://jwt.io/#debugger-io?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ._sytI9TdagSl-vSnVExnCuD46OQVKX7BxQR1YomY9cA

Message Authentication Code (MAC)

Approaches

Integrity

- Encryption of an ordinary digest (e.g. from SHA3)
  - Using, for instance, a symmetric block cipher

- Using encryption with feedback & error propagation
  - CBC-MAC or GCM

↖ !

- Adding a key to the hashed data
  - Keyed-MD5 (128 bits)
    - MD5(K, keyfill, text, K, MD5fill)

    → in the end is the same
  - HMAC (output length depends on the function H used)
    - H(K, opad, H(K, ipad, text))
    - ipad = 0x36 B times opad = 0x5C B times B = size of H input block
      - HMAC-MD5, HMAC-SHA-1, etc.

Message Authentication Code (MAC)

IMPORTANTE !

When used with encryption

- **Encrypt-then-MAC:** MAC is computed from cryptogram: $M = C \mid MAC(C, K_2), C=E(T, K_1)$
  - Allows verifying integrity before decryption
  - MAC calculation is frequently faster than decryption

↳ $M = E(T) \mid MAC(E(T))$

BAD

Semanticly Insecure

- **Encrypt-and-MAC: MAC is computed from plaintext: $M = E(T, K_1) \mid MAC(T, K_2)$**
  - May give information regarding original text (if similar to other text)
  - Receiver will find that text was manipulated **only after decryption plus MAC calculation (slower)**
  - Manipulated ciphertext may attack the decryption algorithm without detection

→ The MAC will be equal if $T_0 = T_1$ //

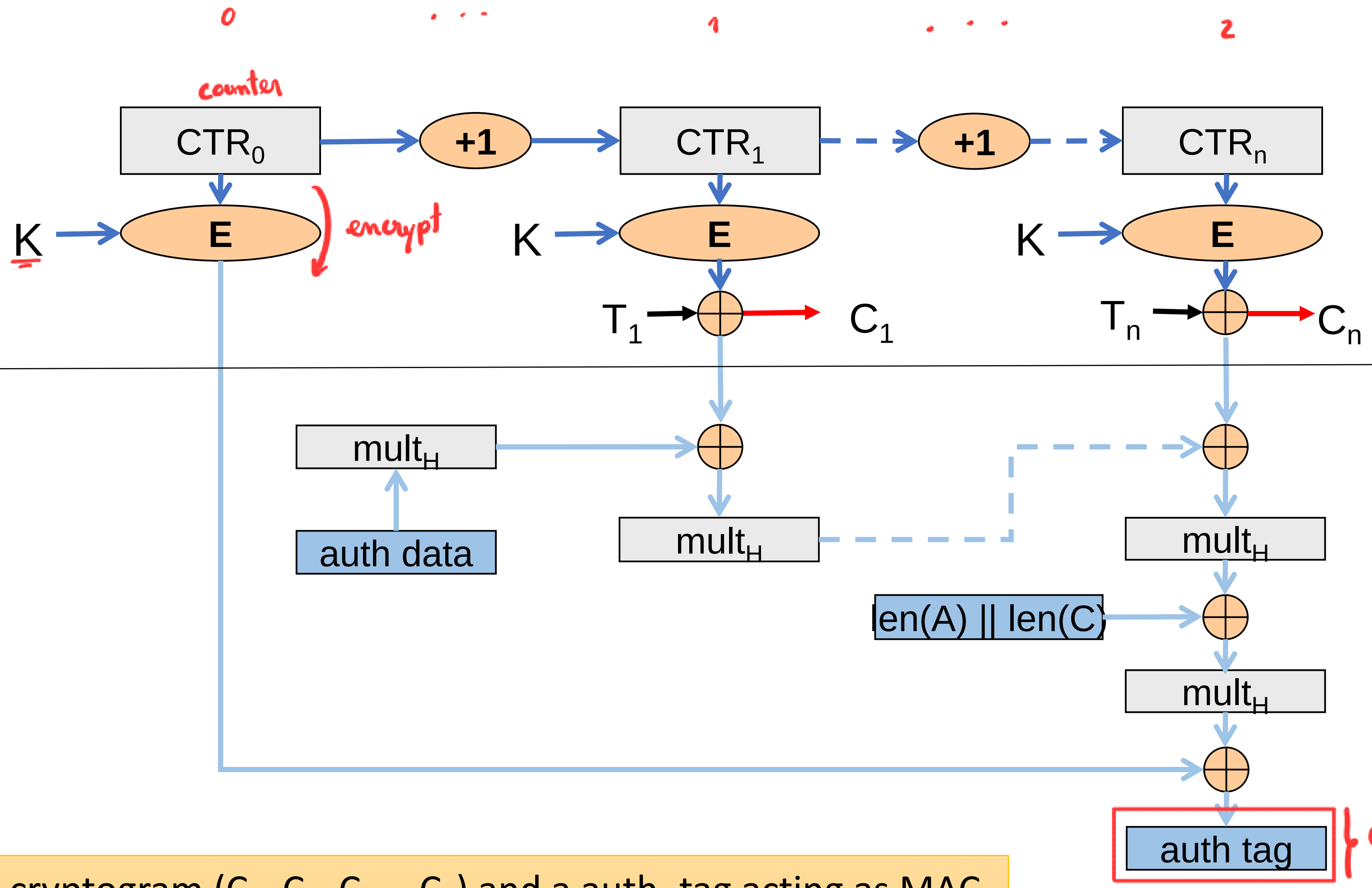
- **MAC-then-Encrypt: MAC is computed from plaintext: $M = E(T \mid MAC(T, K_2), K_1)$**
  - MAC is encrypted (which is not bad)
  - Receiver will find that text was manipulated **only after decryption plus MAC calculation (slower)**
  - Manipulated ciphertext may attack the decryption algorithm without detection

↗ some corruption is the encryption .... and the decryption didn't works

Example: GCM (Galois Counter Mode)

Authentication Encrypt Mode

→ Encrypt-then-mac built-in



Ciphertext with authentication

Standard CTR encryption process

Digest construction

Results in a cryptogram (C_1, C_2, C_3 … C_n) and a auth_tag acting as MAC
Requires an additional auth_data

Key derivation

Motivation

- Cipher algorithms require fixed dimension keys
  - 56, 128, 256… bits

- We may need to derive keys from multiple sources
  - Shared secrets
  - Passwords generated by humans
  - PIN codes and small length secrets

- Original source may have low entropy
  - Reduces the difficulty of a brute force attack
  - Although we must have some strong relation into a useful key

we need to increase the entropy!

- Sometimes we need multiple keys from the same material
  - While not allowing to find the material (a password, another key) from the new key

Key derivation

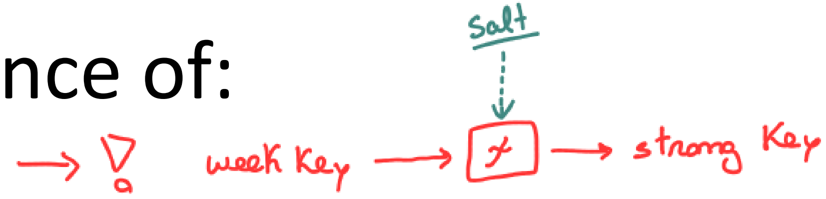
Purposes

- **Key reinforcement**: increase the security of a password
  - Usually defined by humans
  - To make dictionary attacks impractical

- **Key expansion**: increase/decrease the length of a key
  - Expansion to a size that suits an algorithm
  - Eventually derive other related keys for other algorithms (e.g. MAC)

Key derivation

- Key derivation requires the existence of:
  - A **Salt** which makes the derivation unique
    - The salt must be only Known by the server in a database
  - A difficult problem
  - A chosen level of complexity



- Computational difficulty
  - Transformation requires relevant computational resources

- Memory difficulty
  - Transformation requires relevant storage resources
  - Limits attacks using dedicated hardware accelerators

Key derivation

Simple Approach: A Digest function

- Arguments:
  - Salt = A random value
  - Password = a secret (provided by humans)
  - H = An adequate Digest Function

key = H(password, salt)

- Advantages:
  - Key has a large length, and can be truncated to the adequate length
  - Two passwords will result in diferent keys
  - Finding the key will not lead to the password
- Issues: simple, enabling brute force/diccionary attacks

Key derivation

Password Based Key Derivation Function (PBKDF2)

- Produces a key from a password, with a chosen difficulty

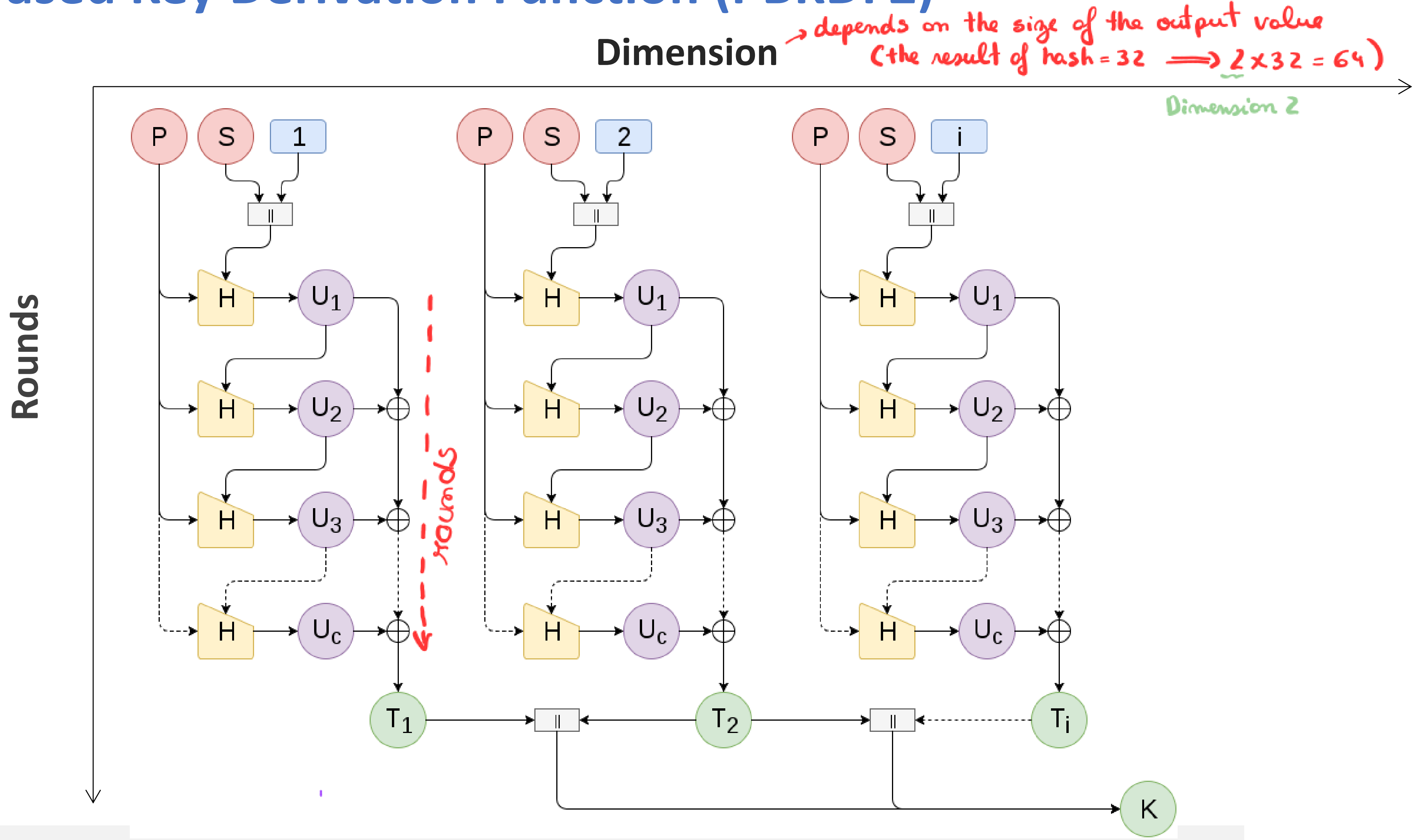
simply hash many times { 100K 600K≈ → To prevent GPU's attack

- **K = PBKDF2(PRF, Salt, rounds, dim, password)**
  - PRF: Pseudo-Random-Function: a digest function
  - Salt: a random value
  - Rounds: the computational cost (hundreds of thousands)
  - Dim: the size of the result required

- Operation: calculate ROUNDS x DIM operations of the PRF using the SALT and Password
  - Higher number of rounds will increase the cost of brute force/diccionary attacks

Key derivation

Password Based Key Derivation Function (PBKDF2)



→ depends on the size of the output value (the result of hash = 32 ⟹ 2 x 32 = 64)

Dimension 2

rounds

Key derivation

scrypt

- Produces a key with a chosen computation and storage cost

- **K = scrypt(password, salt, n, p, dim, r, hLen, Mflen)**
  - Password: a secret
  - Salt: a random value
  - N: the cost parameter
  - P: the parallelization parameter. p ≤ (232− 1) * hLen / MFLen
  - Dim: the size of the result
  - R: the size of the blocks to use (default is 8)
  - hLen: the size of the digest function (32 for SHA256)
  - Mflen: bytes in the internal mix (default is 8 x R)

Key Derivation: scrypt

- Produces a key with a chosen storage cost

- K = scrypt(password, salt, n, p, dim, r, hLen, Mflen)
  - Password: a secret
  - Salt: a random value
  - N: the cost parameter
  - P: the parallelization parameter. $p \leq (2^{32} - 1) * hLen / MFLen$
  - Dim: the size of the result
  - R: the size of the blocks to use (default is 8)
  - hLen: the size of the digest function (32 for SHA256)
  - Mflen: bytes in the internal mix (default is 8 x R)

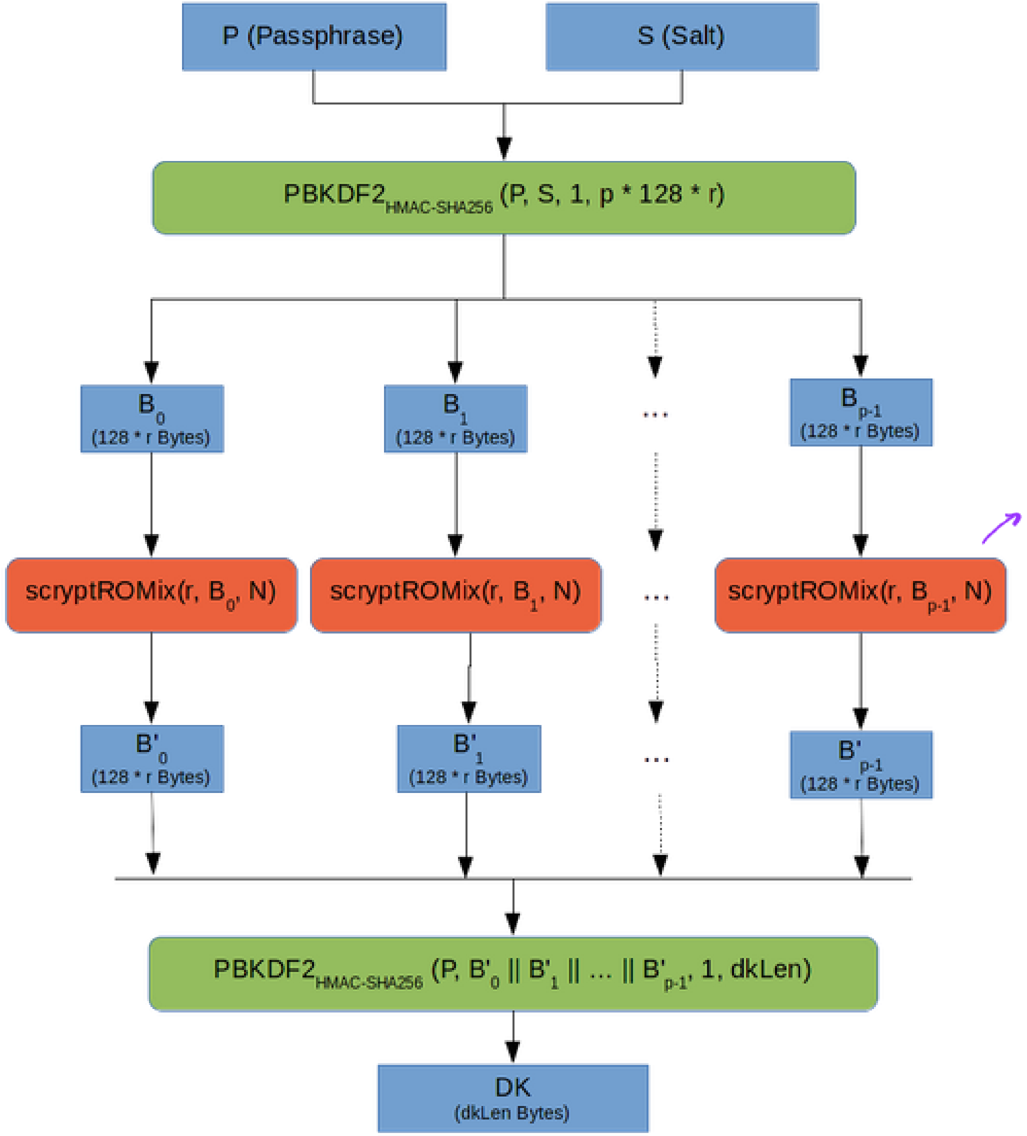
Key derivation

scrypt

scrypt (P, S, N, r, p, dkLen)

Parameters:

N (CPU/Memory Cost Parameter)
r (Block Size)
p (Parallelization Parameter)
dkLen (Output Length)



P (Passphrase)

S (Salt)

$\text{PBKDF2}_{\text{HMAC-SHA256}}$ (P, S, 1, p * 128 * r)

B_0 (128 * r Bytes)

B_1 (128 * r Bytes)

…

B_{p-1} (128 * r Bytes)

scryptROMix(r, B_0, N)

scryptROMix(r, B_1, N)

…

scryptROMix(r, B_{p-1}, N)

B'_0 (128 * r Bytes)

B'_1 (128 * r Bytes)

…

B'_{p-1} (128 * r Bytes)

$\text{PBKDF2}_{\text{HMAC-SHA256}}$ (P, B'_0 || B'_1 || … || B'_{p-1}, 1, dkLen)

DK (dkLen Bytes)

→ not used for nothing...

} Making the process slower ...