

# Digestões, controlo da integridade e derivação de chaves

SIO

João Paulo Barraca

# Funções de digestão

## Visão geral

- Produzir um **resumo** digital de dados denominado **message digest**
  - Os dados são um texto ou qualquer informação binária
- O **comprimento do** resumo da mensagem **é fixo**
  - independentemente do comprimento do texto
    - Tanto um item de dados de 200 bytes como um item de dados de 200 TB resultarão num resumo com o mesmo comprimento
- O valor do resumo da mensagem **depende fortemente** dos dados
- Duas digestões são normalmente **muito diferentes**
  - Mesmo que os dados originais sejam extremamente semelhantes

# Funções de digestão

## Propriedades

- Resistência à pré-imagem
  - Dado um resumo, é impossível encontrar um texto original que o produza
  - Ou seja: não podemos passar de um resumo para os dados (não os podemos "desencriptar")
- Resistência à 2ª pré-imagem
  - Dado um texto, é impossível encontrar outro com o mesmo resumo
  - Ou seja: se **tivermos um texto**, não podemos encontrar outro com o mesmo resumo
- Resistência à colisão
  - É impossível encontrar dois textos com a mesma digestão
  - Ou seja: dados dois textos únicos, estes resultarão **num resumo diferente**
    - Relacionado com o paradoxo do aniversário: probabilidade de colisão  $P = 2^{n/2}$  quando o  $n$  típico é  $\geq 256$

# Funções de digestão

## Vamos verificar: Independência do tamanho

- Considerando os textos semelhantes, mas diferentes:
  - T1: "Olá      Utilizador\_A!"
  - T2: "Olá      Utilizador\_XPTO!      Bem-vindo a esta aula"
- Algoritmos diferentes criarão resumos com comprimentos diferentes, mas **independentes** da dimensão do texto
  - MD5 (128 bits):
    - T1: 70df836fdaf02e0dfc990f9139762541
    - T2: 18f12f09c45d880ce738afe4780c2f3e
  - SHA-1 (160 bits):
    - T1: f591aa1eabcc97fb39c5f422b370ddf8cb880fde
    - T2: 622f7832e204f2d70161cf42480c4bf0f13e7324
  - SHA-256 (256 bits):
    - T1: 9649d8c0d25515a239ec8ec94b293c8868e931ad318df4ccd0dfffd67aff89905
    - T2: 6453be3f643d0a7e9b5890eed76bb63df8b6b071b30d5f97269a530c289b9839

# Funções de digestão

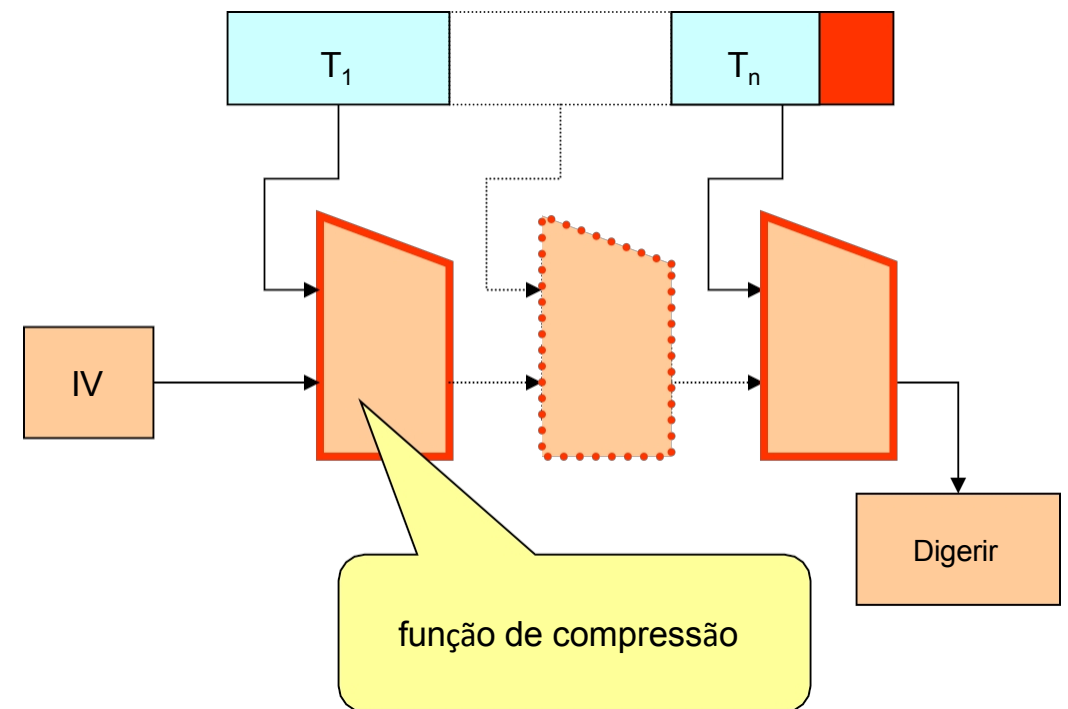
## Vamos verificar: Dependência de conteúdo

- Considerando os textos semelhantes, mas diferentes (1 bit de diferença 'B' -> 'C'):
  - T1: "Olá Utilizador\_B!", [0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20, 0x55, 0x73, 0x65, 0x72, 0x5f, **0x42**, 0x21]
  - T2: "Olá Utilizador\_C!", [0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20, 0x55, 0x73, 0x65, 0x72, 0x5f, **0x43**, 0x21]
- Uma pequena diferença no texto (1 bit) resulta num **digest completamente diferente**
  - MD5:
    - T1: c32e0f62a7c9c815063d373acac80c37
    - T2: 324a1bfc3041259480c6ad164cf0529f
  - SHA-1:
    - T1: bab31eb62f961266758524071a7ad8221bc8700b
    - T2: bd758d82899d132cd2af66dc3402b948d98de62d
  - SHA-256:
    - T1: e663a01d3bec4f35a470aba4baccece79bf484b5d0bffa88b59a9bb08707758a
    - T2: 69f78345da90c6b8d4785b769cd6ae09e0531716fe5f5a392fde1bdc70a2bb7d

# Funções de digestão

## Abordagens

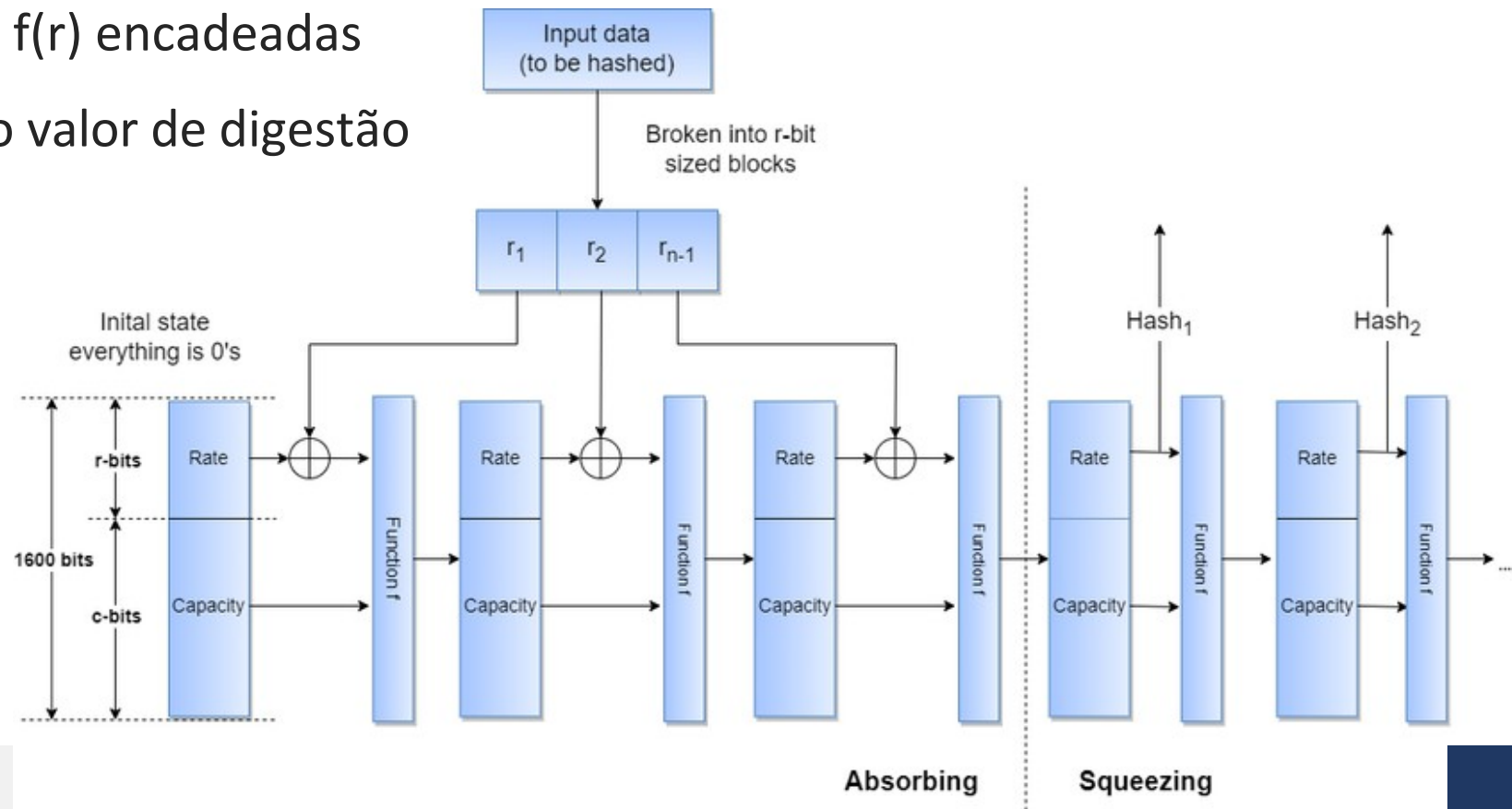
- Construção de Merkle-Damgård
  - Funções de compressão unidirecional resistente à colisão
    - Pode ser uma cifra de bloco!
  - Compressão iterativa
  - Acolchoamento do comprimento
  - O tamanho do resumo é o último bloco
  - Pode ser retomado!
    - Digesto é o estado em  $T_n$
  - Algoritmos: MD5, SHA1, SHA2



# Funções de digestão

## Abordagens

- Funções da esponja
  - Dados divididos em blocos de tamanho  $r$
  - Fase de absorção: chamadas  $f(r)$  encadeadas
  - Squeezing: extrair bits para o valor de digestão
  - Algoritmos: SHA3

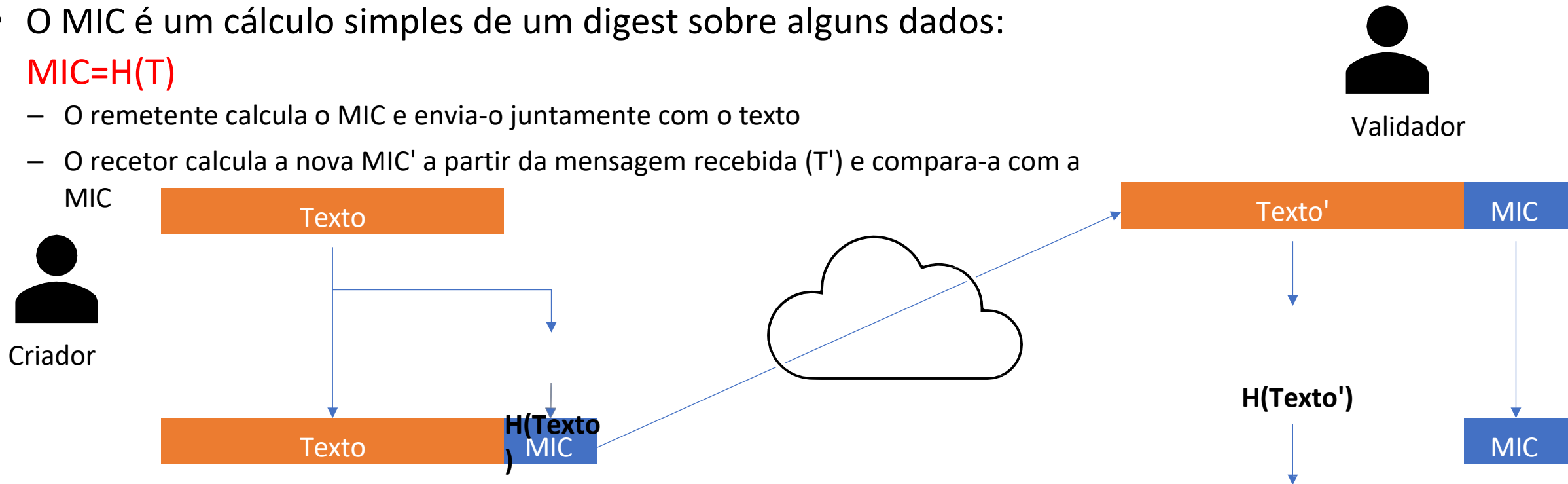


# Código de Integridade da Mensagem (MIC)

- Fornecer a capacidade de detetar alterações **arbitrárias** nos dados
  - Erros de comunicação/armazenamento de um processo aleatório ou sem controlo de integridade
  - Os humanos/atacantes podem alterar o texto e calcular um novo MIC!
- O MIC é um cálculo simples de um digest sobre alguns dados:

$$\text{MIC} = H(T)$$

- O remetente calcula o MIC e envia-o juntamente com o texto
- O recetor calcula a nova MIC' a partir da mensagem recebida (T') e compara-a com a MIC

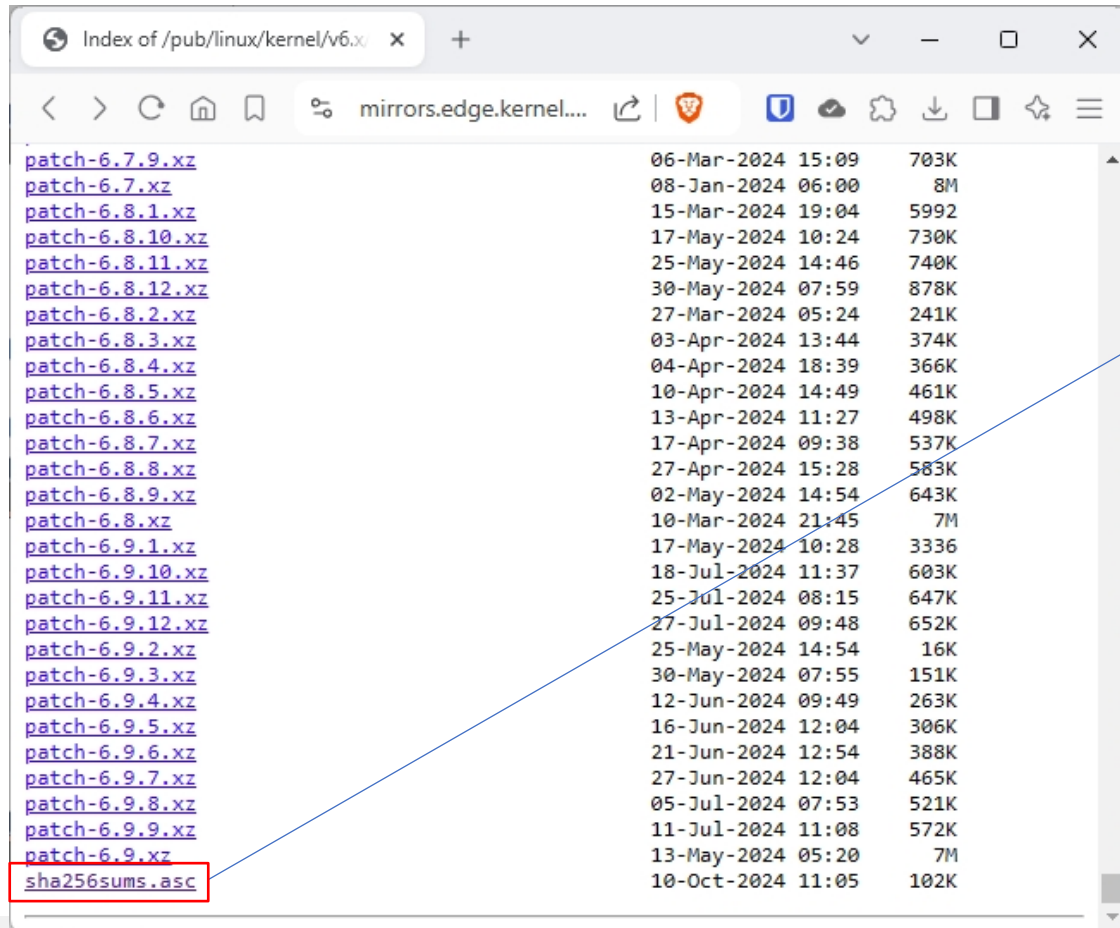




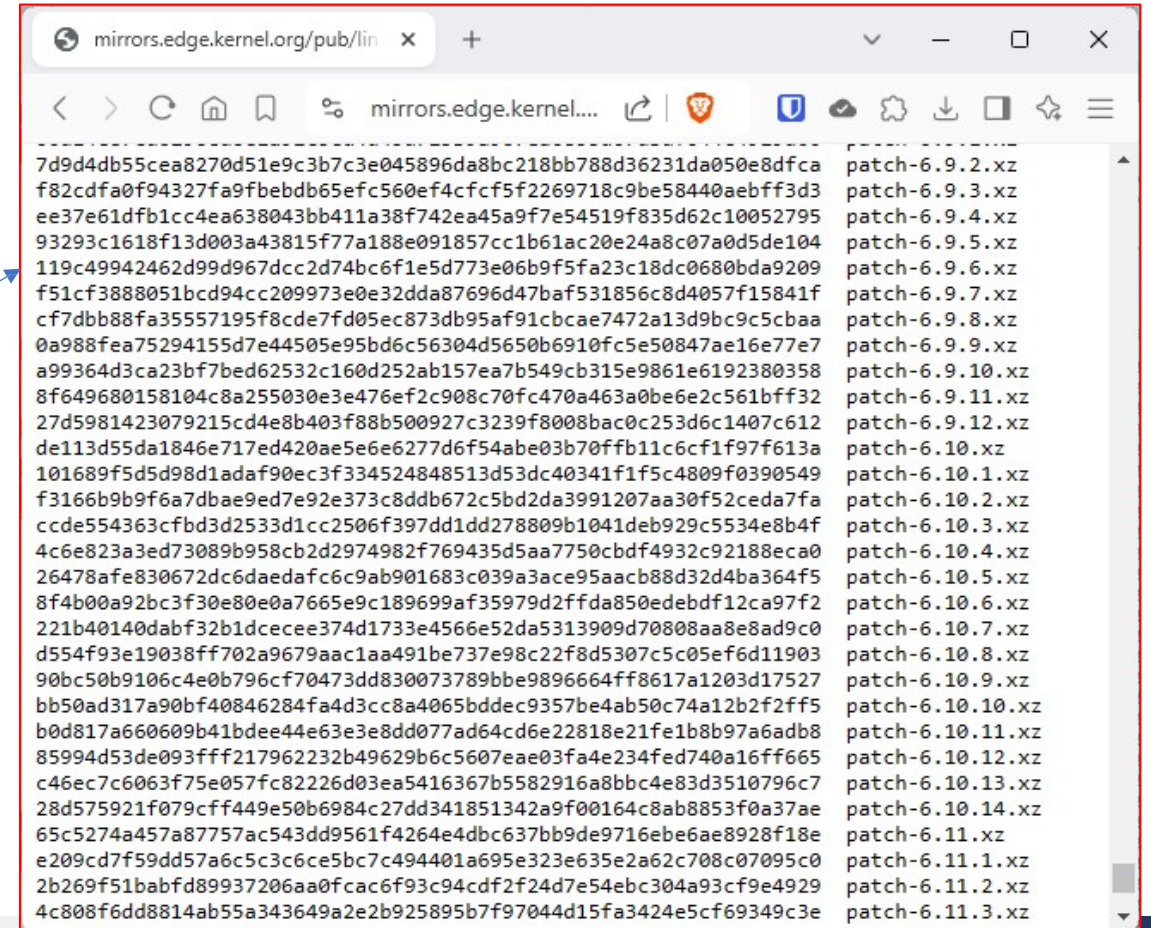
MIC'

**iguais?**

# Exemplo de utilização em kernel.org para validar a integridade do ficheiro



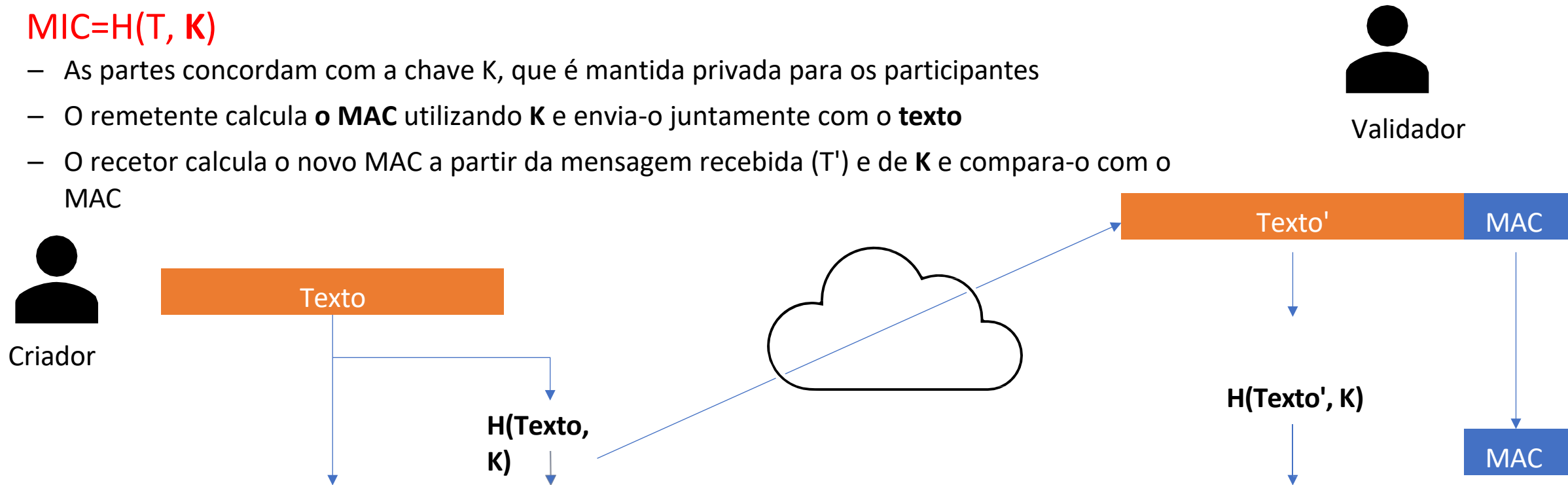
<a href="#">patch-6.7.9.xz</a>	06-Mar-2024 15:09	703K
<a href="#">patch-6.7.xz</a>	08-Jan-2024 06:00	8M
<a href="#">patch-6.8.1.xz</a>	15-Mar-2024 19:04	5992
<a href="#">patch-6.8.10.xz</a>	17-May-2024 10:24	730K
<a href="#">patch-6.8.11.xz</a>	25-May-2024 14:46	740K
<a href="#">patch-6.8.12.xz</a>	30-May-2024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



7d9d4db55cea8270d51e9c3b7c3e045896da8bc218bb788d36231da050e8dfca f82cdfa0f94327fa9fbebdb65efc560ef4cfcf5f2269718c9be58440aebff3d3 ee37e61dfb1cc4ea638043bb411a38f742ea45a9f7e54519f835d62c10052795 93293c1618f13d003a43815f77a188e091857cc1b61ac20e24a8c07a0d5de104 119c49942462d99d967dcc2d74bc6f1e5d773e06b9f5fa23c18dc0680bda9209 f51cf3888051bcd94cc209973e0e32dda87696d47baf531856c8d4057f15841f cf7dbb88fa35557195f8cde7fd05ec873db95af91cbcae7472a13d9bc9c5cbaa 0a988fea75294155d7e44505e95bd6c56304d5650b6910fc5e50847ae16e77e7 a99364d3ca23bf7bed62532c160d252ab157ea7b549cb315e9861e6192380358 8f649680158104c8a255030e3e476ef2c908c70fc470a463a0be6e2c561bff32 27d5981423079215cd4e8b403f88b500927c3239f8008bac0c253d6c1407c612 de113d55da1846e717ed420ae5e6e6277d6f54abe03b70ffb11c6cf1f97f613a 101689f5d5d98d1adaf90ec3f334524848513d53dc40341f1f5c4809f0390549 f3166b9b9f6a7dbae9ed7e92e373c8ddb672c5bd2da3991207aa30f52ceda7fa ccde554363cfbd3d2533d1cc2506f397dd1dd278809b1041deb929c5534e8b4f 4c6e823a3ed73089b958cb2d2974982f769435d5aa7750cbdf4932c92188eca0 26478afe830672dc6daedafcc6c9ab901683c039a3ace95aacb88d32d4ba364f5 8f4b00a92bc3f30e80e0a7665e9c189699af35979d2ffda850edebdf12ca97f2 221b40140dabf32b1dcecee374d1733e4566e52da5313909d70808aa8e8ad9c0 d554f93e19038ff702a9679aac1aa491be737e98c22f8d5307c5c05ef6d11903 90bc50b9106c4e0b796cf70473dd830073789bbe9896664ff8617a1203d17527 bb50ad317a90bf40846284fa4d3cc8a4065bddec9357be4ab50c74a12b2f2ff5 b0d817a660609b41bdee44e63e3e8dd077ad64cd6e22818e21fe1b8b97a6adb8 85994d53de093fff217962232b49629b6c5607eae03fa4e234fed740a16ff665 c46ec7c6063f75e057fc82226d03ea5416367b5582916a8bbc4e83d3510796c7 28d575921f079cff449e50b6984c27dd341851342a9f00164c8ab8853f0a37ae 65c5274a457a87757ac543dd9561f4264e4dbc637bb9de9716ebe6ae8928f18e e209cd7f59dd57a6c5c3c6ce5bc7c494401a695e323e635e2a62c708c07095c0 2b269f51babfd89937206aa0fcac6f93c94cdf2f24d7e54ebc304a93cf9e4929 4c808f6dd8814ab55a343649a2e2b925895b7f97044d15fa3424e5cf69349c3e	patch-6.9.12.xz
---	-----------------

# Código de autenticação de mensagem (MAC)

- Fornecer a capacidade de detetar alterações **deliberadas** nos dados
  - Qualquer alteração nos dados, mesmo que seja efectuada por atacantes!
- O MAC é um cálculo chaveado de um resumo sobre alguns dados:  
 **$MAC = H(T, K)$** 
  - As partes concordam com a chave  $K$ , que é mantida privada para os participantes
  - O remetente calcula o **MAC** utilizando  $K$  e envia-o juntamente com o **texto**
  - O recetor calcula o novo MAC a partir da mensagem recebida ( $T'$ ) e de  $K$  e compara-o com o MAC



Texto

MAC

MAC

**iguais?**

## Exemplo de utilização em JWT

Encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ..syti9TdagSl-vSnVExnCUD460QVKX7BxQR1YomY9cA

Cookie fornecido  
na página Web  
aos Clientes

Os clientes não podem  
alterar o Cookie devido  
ao MAC

## Decoded EDIT THE PAYLOAD AND SECRET

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

## Algoritmo

PAYLOAD: DATA

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

Dados no  
cookie

VERIFY SIGNATURE

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload), ←  
    secret_key  
)
```

☒ secret base64 encoded

MAC calculado  
com secret\_key.  
**A chave é privada para o  
servidor**

<https://jwt.io/#debugger-io?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyOQ.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1dWUiOiJkZW50aGVkaWwifQ>

# Código de autenticação de mensagem (MAC)

## Abordagens

- Encriptação de um resumo normal (por exemplo, de SHA3)
  - Utilizando, por exemplo, uma cifra de bloco simétrica
- Utilizar a encriptação com feedback e propagação de erros
  - CBC-MAC ou GCM
- Adicionar uma chave aos dados com hash
  - MD5 com chave (128 bits)
    - $\text{MD5}(K, \text{keyfill}, \text{texto}, K, \text{MD5fill})$
  - HMAC (o comprimento de saída depende da função H utilizada)
    - $H(K, \text{opad}, H(K, \text{ipad}, \text{texto}))$
    - $\text{ipad} = 0x36 \text{ B}$        $\text{timesopad} = 0x5C \text{ B times B} = \text{tamanho do bloco de entrada H}$ 
      - HMAC-MD5, HMAC-SHA-1, etc.

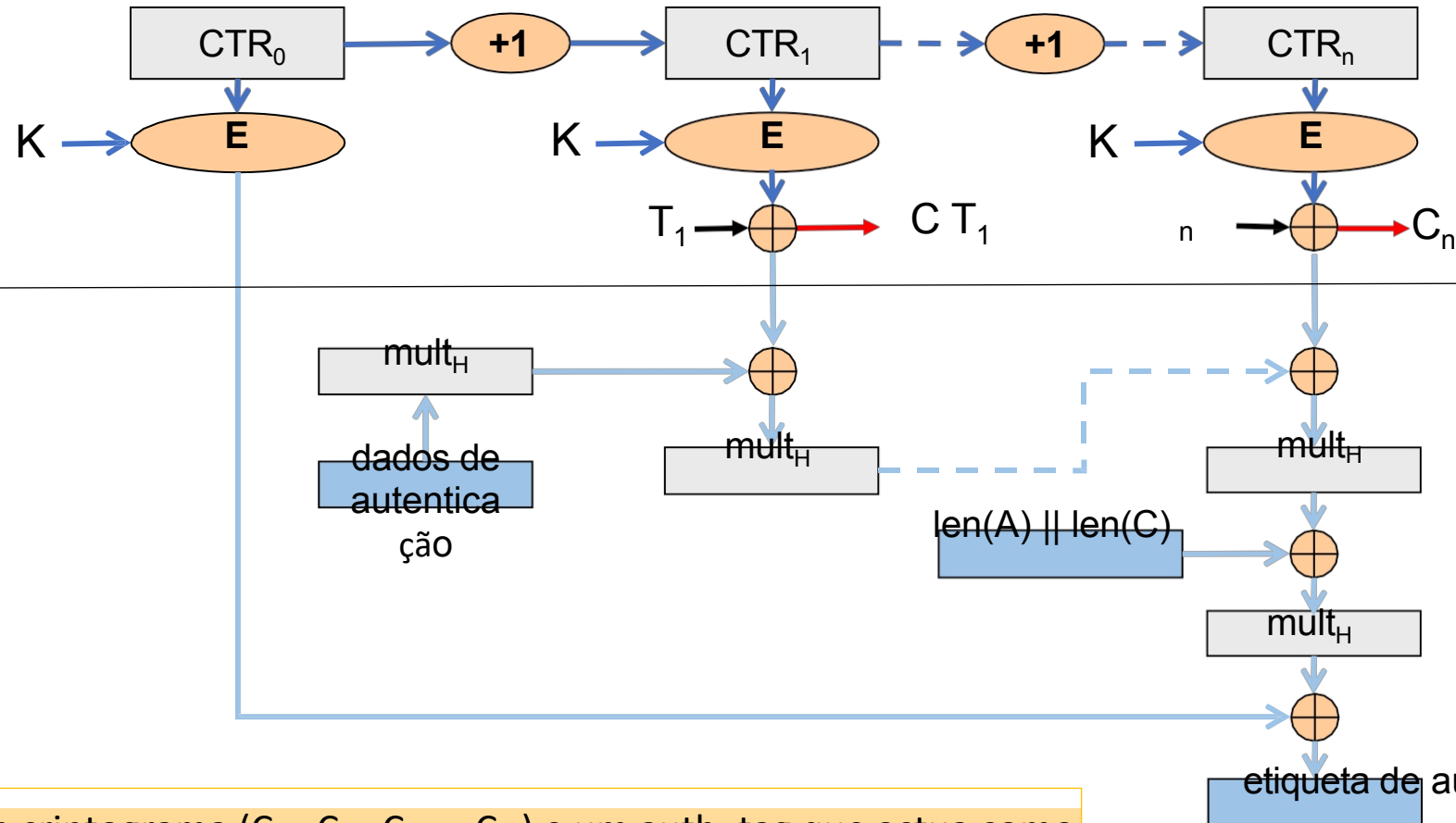
# Código de autenticação de mensagem (MAC)

## Quando utilizado com encriptação

- **Encriptar e depois MAC:** O MAC é calculado a partir do criptograma:  $M = C \parallel \text{MAC}(C, K_2)$ ,  $C = E(T, K_1)$ 
  - Permite verificar a integridade antes da descriptação
  - O cálculo do MAC é frequentemente mais rápido do que a descriptação
- **Encriptar-e-MAC:** O MAC é calculado a partir do texto simples:  $M = E(T, K_1) \parallel \text{MAC}(T, K_2)$ 
  - Pode dar informações sobre o texto original (se for semelhante a outro texto)
  - O recetor verificará que o texto **só** foi manipulado **após a descriptação e o cálculo do MAC (mais lento)**
  - O texto cifrado manipulado pode atacar o algoritmo de descriptação sem ser detectado
- **MAC-then-Encrypt:** O MAC é calculado a partir do texto simples:  $M = E(T \parallel \text{MAC}(T, K_2), K_1)$ 
  - O MAC é encriptado (o que não é mau)
  - O recetor verificará que o texto **só** foi manipulado **após a descriptação e o cálculo do MAC (mais lento)**



# Exemplo: GCM (Modo Contador Galois)



CTR padrão  
processo de  
criptação

Construção de  
digestores

Resulta num criptograma ( $C_1, C_2, C_3 \dots C_n$ ) e um auth\_tag que actua como MAC

Requer um auth\_data adicional



# Derivação de chaves

## Motivação

- Os algoritmos de cifra requerem chaves de dimensão fixa  
-56 , 128, 256... bits
- Podemos precisar de derivar chaves de várias fontes
  - Segredos partilhados
  - Palavras-passe geradas por humanos
  - Códigos PIN e segredos de pequena dimensão
- A fonte original pode ter baixa entropia
  - Reduz a dificuldade de um ataque de força bruta
  - Embora tenhamos de ter uma relação forte com uma chave útil
- Por vezes, são necessárias várias chaves do mesmo material
  - Embora não permita encontrar o material (uma palavra-passe, outra chave) a partir da nova chave

# Derivação de chaves

## Objectivos

- **Reforço de chaves:** aumentar a segurança de uma palavra-passe
  - Normalmente definido por humanos
  - Para tornar os ataques de dicionário impraticáveis
- **Expansão da chave:** aumentar/diminuir o comprimento de uma chave
  - Expansão para um tamanho adequado a um algoritmo
  - Eventualmente derivar outras chaves relacionadas para outros algoritmos (por exemplo, MAC)

# Derivação de chaves

- A derivação de chaves requer a existência de:
  - Um **sal** que torna a derivação única
  - Um problema difícil
  - Um nível de complexidade escolhido
- Dificuldade computacional
  - A transformação requer recursos computacionais relevantes
- Dificuldade de memória
  - A transformação requer recursos de armazenamento relevantes
  - Limita os ataques que utilizam aceleradores de hardware dedicados

# Derivação de chaves

## Abordagem simples: Uma função Digest

- Argumentos:
    - Salt = Um valor aleatório
    - Palavra-passe = um segredo (fornecido por humanos)
    - H = Uma função de digestão adequada
- key = H(password, salt)**
- Vantagens:
    - A chave tem um comprimento grande e pode ser truncada para o comprimento adequado
    - Duas palavras-passe resultarão em chaves diferentes
    - Encontrar a chave não conduzirá à palavra-passe
  - Problemas: simples, permitindo ataques de força bruta/dicionários

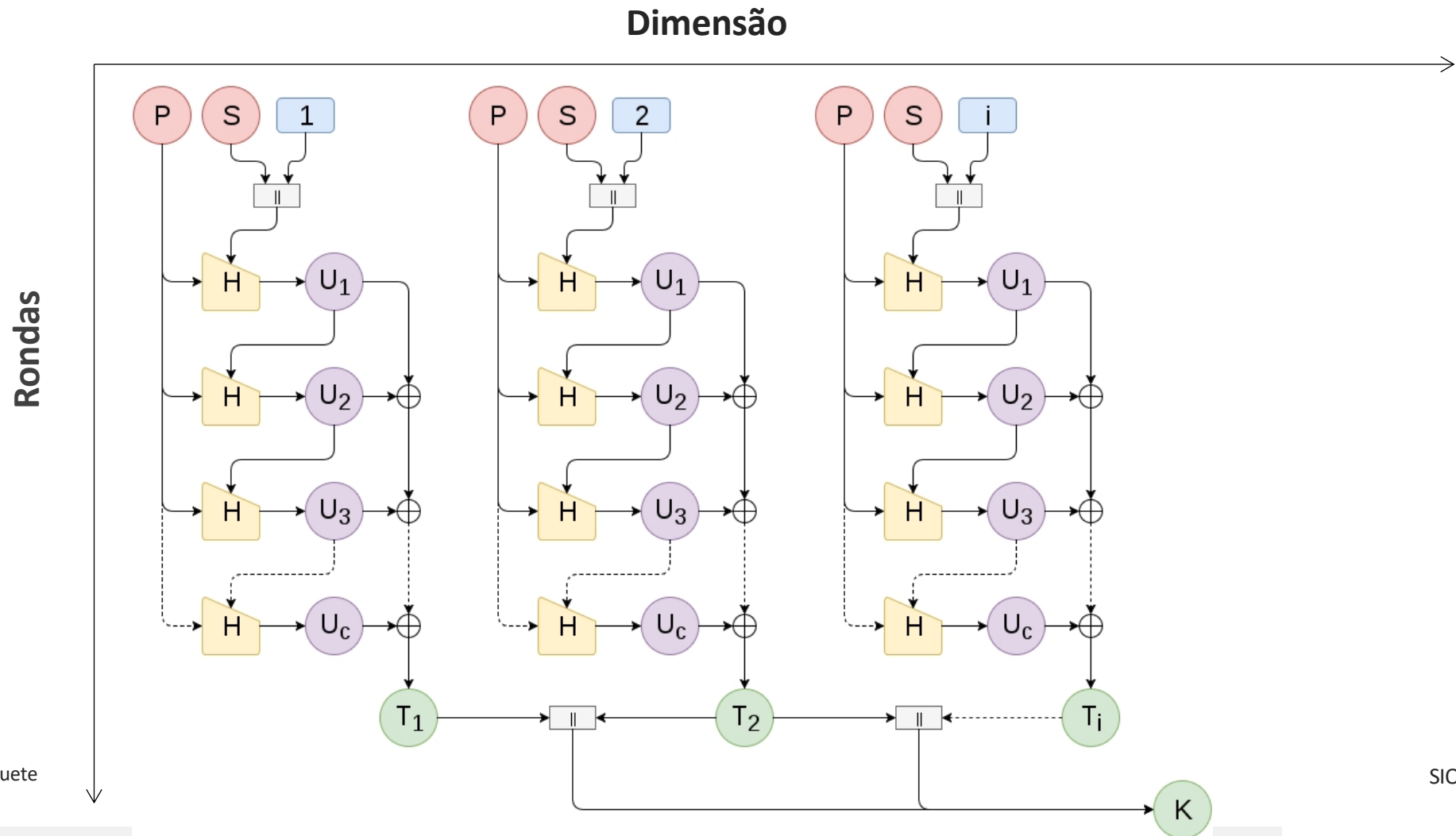
# Derivação de chaves

## Função de derivação de chaves baseada em palavra-passe (PBKDF2)

- Produz uma chave a partir de uma palavra-passe, com uma dificuldade escolhida
- **$K = \text{PBKDF2}(\text{PRF}, \text{Salt}, \text{rounds}, \text{dim}, \text{password})$** 
  - PRF: Função Pseudo-Aleatória: uma função de digestão
  - Sal: um valor aleatório
  - Rondas: o custo computacional (centenas de milhares)
  - Dim: o tamanho do resultado pretendido
- Operação: calcular as operações  $\text{ROUNDS} \times \text{DIM}$  da PRF utilizando o SALT e a Password
  - Um maior número de rondas aumentará o custo dos ataques de força bruta/dicionários

# Derivação de chaves

## Função de derivação de chave baseada em palavra-passe (PBKDF2)



# Derivação de chaves

## criptografar

- Produz uma chave com um custo de computação e armazenamento escolhido
- **$K = \text{script}(\text{password}, \text{salt}, n, p, \text{dim}, r, \text{hLen}, \text{Mflen})$** 
  - Palavra-passe: um segredo
  - Sal: um valor aleatório
  - N: o parâmetro de custo
  - P: o parâmetro de paralelização.  $p \leq (2^{32} - 1) * \text{hLen} / \text{MFLen}$
  - Dim: o tamanho do resultado
  - R: o tamanho dos blocos a utilizar (a predefinição é 8)
  - hLen: o tamanho da função de digestão (32 para SHA256)
  - Mflen: bytes na mistura interna (a predefinição é  $8 \times R$ )

# Derivação de chaves: scrypt

- Produz uma chave com um custo de armazenamento selecionado
- $K = \text{scrypt}(\text{password}, \text{salt}, n, p, \text{dim}, r, \text{hLen}, \text{Mflen})$ 
  - Palavra-passe: um segredo
  - Sal: um valor aleatório
  - N: o parâmetro de custo
  - P: o parâmetro de paralelização.  $p \leq (2^{32} - 1) * \text{hLen} / \text{MFLen}$
  - Dim: o tamanho do resultado
  - R: o tamanho dos blocos a utilizar (a predefinição é 8)
  - hLen: o tamanho da função de digestão (32 para SHA256)
  - Mflen: bytes na mistura interna (a predefinição é  $8 \times R$ )



# Derivação de chaves

## criptografar

