

Fundamentos de Segurança Informática (FSI)

2021/2022 - LEIC

Manuel Barbosa
mbb@fc.up.pt

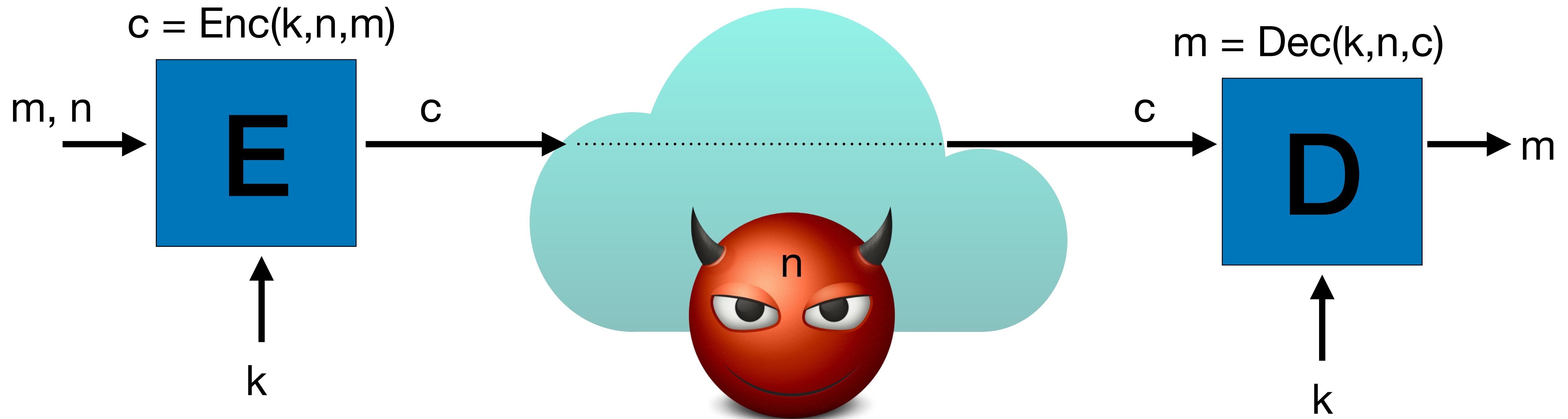
Aula 15

Criptografia: Parte 2

Lembrar aula anterior

- Cifras simétricas:
 - permitem obter **confidencialidade**
 - assumindo uma chave secreta pré-partilhada entre emissor e recetor
- Exemplos:
 - cifra de bloco AES utilizada em counter mode (AES-CTR)
 - cifra dedicada ChaCha20

Cifras Simétricas



- E, D: algoritmos (encrypt, decrypt) => **públicos e standard!!!**
- k: chave secreta (hoje tipicamente 128 bits)
- n, m, c: nonce (non-repeating, **público**), texto-limpo, criptograma

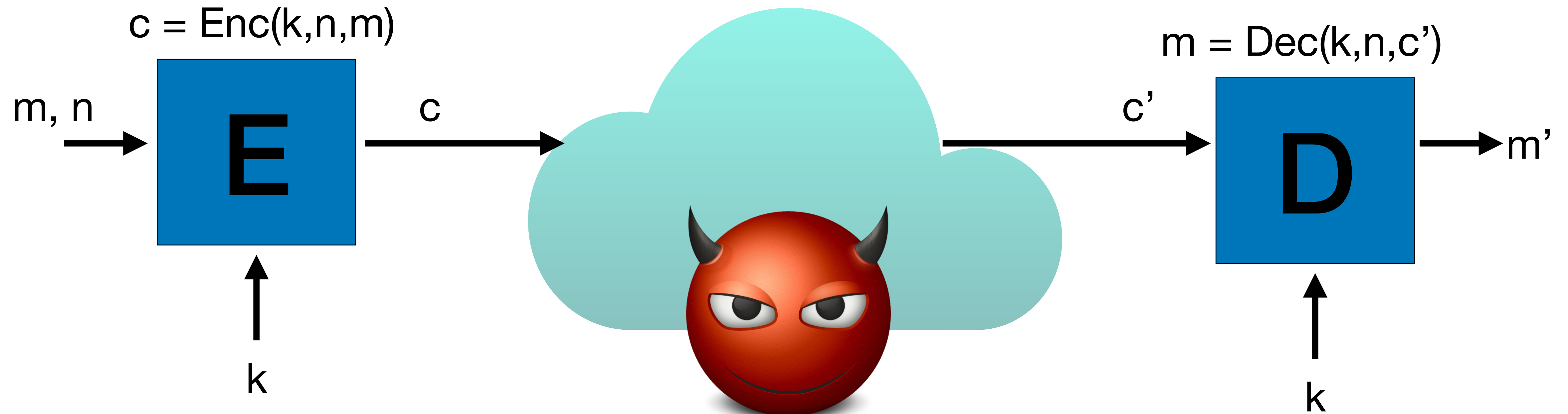
Poder do adversário

- Confidencialidade em que condições:
 - adversário apenas observa a rede?
 - adversário pode alterar, remover ou inserir mensagens?
 - neste último caso, que garantias tem o receptor?
- As cifras que vimos não protegem contra ataques ativos:
 - se o adversário alterar o criptograma, o recetor pode não perceber
 - processar um criptograma alterado pode comprometer confidencialidade

Poder do adversário

- Na prática é **sempre necessário acautelar ataques ativos**:
 - integridade: mensagem não é alterada
 - autenticidade:
 - A e B partilham uma chave secreta
 - B apenas aceita mensagens transmitidas por A
 - autenticidade \Rightarrow integridade (porquê?)

Exemplo de ataque ativo

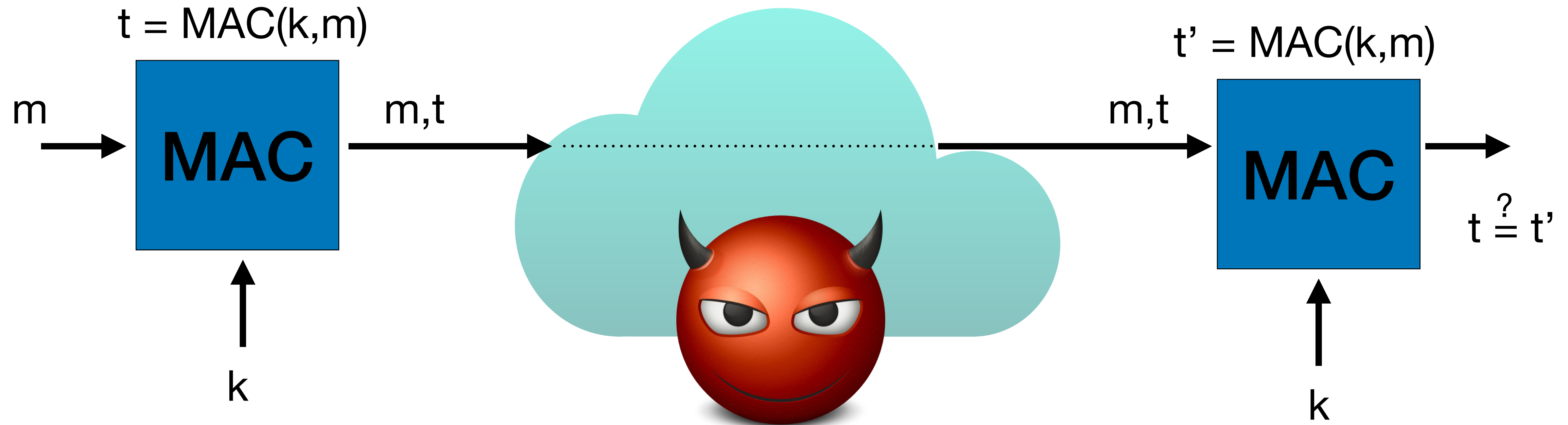


$m = \text{'transferir 00000001'}$
 $c = \text{'0111101010001111010'}$

Bit Flip

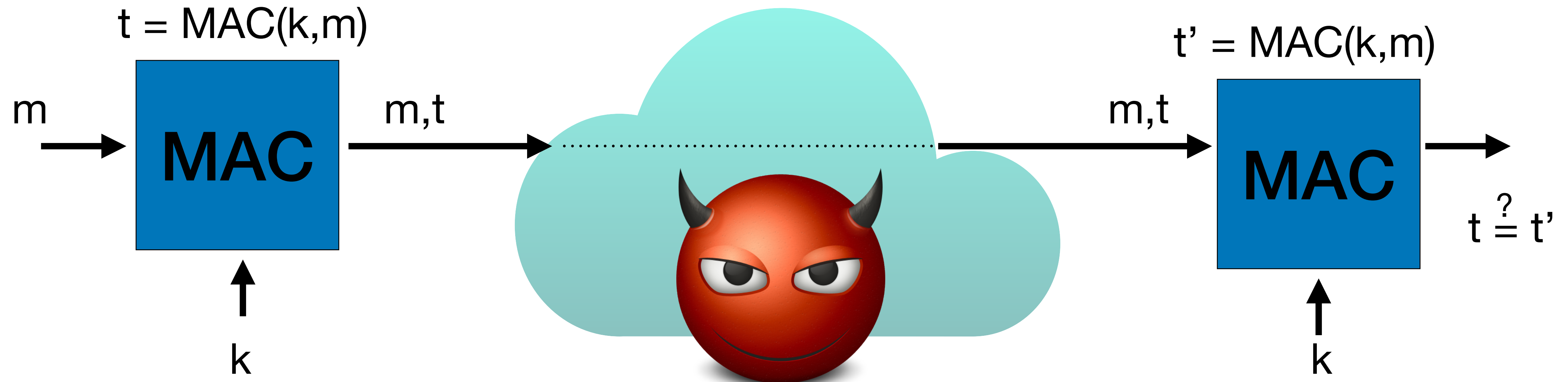
$c = \text{'0111101010011111010'}$
 $m = \text{'transferir 10000001'}$

Message Authentication Codes



- MAC: algoritmo => **público e standard!!!**
- k : chave secreta (hoje tipicamente 128 bits)
- m, t : mensagem (**pública**), tag (pequena, tipicamente 256 bits)

Message Authentication Codes



- Usamos MACs para autenticidade e integridade: **tags não “encriptam” m**
- tag = checksum criptográfico, apenas calculável com k
- garantia: tag correta => t foi calculada como $t := \text{MAC}(k,m)$
- emissor conhecia a chave k e usou-a para autenticar m

Garantias MAC

- Recebemos (m,t) com $t = \text{MAC}(k,m)$, e apenas Alice conhece k
 - Alice transmitiu certamente (m,t)
 - Mas ...
 - se atacante não entregar?
 - se atacante entregar múltiplas vezes?
- MAC não permite detetar (só por si) estes ataques

Garantias MAC

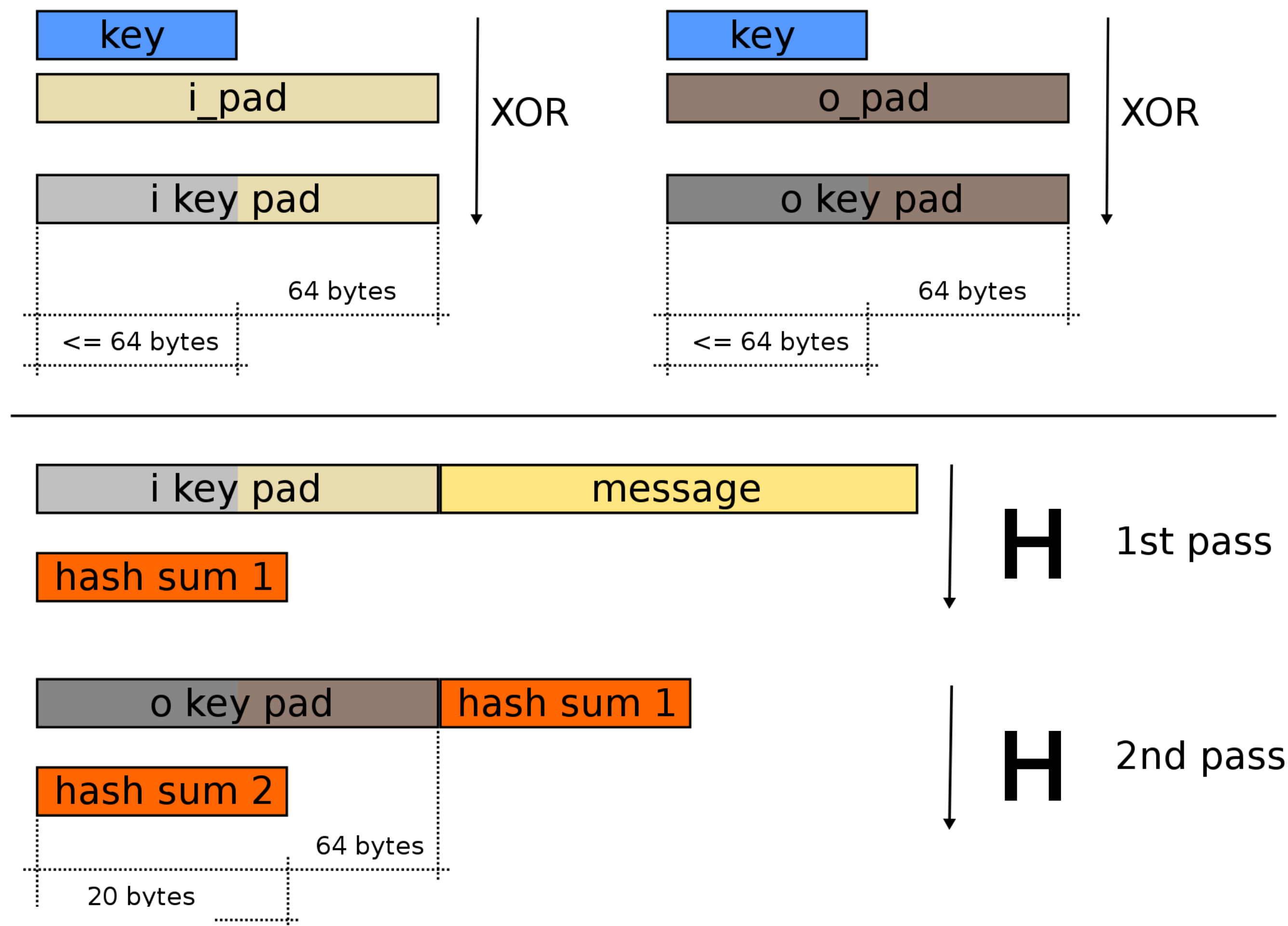
- Solução para detetar remoção, duplicação e reordenação de mensagens:
 - garantir que Alice apenas transmite m uma vez
 - Como? Pré-processamos m para que seja única em cada transmissão
 - usamos um número de sequência e autenticamos $t = \text{MAC}(k, n || m)$
 - transmitimos (m, t) , mas recetor mantém o mesmo número
 - verificação da tag \Rightarrow mesmo n dos dois lados!

Exemplo de Aplicação MAC

- Cookie poisoning:
 - servidor Web fixa cookie, por exemplo: créditos num jogo
 - utilizador edita cookie e aumenta créditos
 - solução: servidor autentica informação na cookie incluindo um MAC
 - necessário número de sequência?
 - problema de partilha de chaves?

Construções MAC: HMAC

- $t = H(okey || H(ikkey || m))$
- MAC a partir de função de hash:
 - hoje em dia SHA-256
 - $t \Rightarrow$ 256 bits, 32 bytes
 - porque é seguro?
 - aprender criptografia



Construções MAC: Poly1305

- $t = f(m, r) + s$
- Função de hash algébrica
 - polinómio definido por m
 - avaliado no ponto r
- Chave secreta $(r, s) \Rightarrow 256$ bits
- Pode ser usada apenas uma vez
- One-time MAC



$$f(r) = C_1 r^n + C_2 r^{n-1} + C_3 r^{n-2} + \dots + C_{n-1} r^2 + C_n r$$

$$((\dots (C_1 r + C_2) r + C_3) r + \dots + C_{n-1}) r + C_n) r$$

Confidencialidade e Autenticidade

- Como obter ambas?
 - precisamos de usar: cifra (confidencialidade) + MAC (autenticidade)
 - precisamos de duas chaves secretas
 - como combinar a utilização de uma cifra com um MAC?

Confidencialidade e Autenticidade

- Três hipóteses

- Encrypt and Mac (SSH)

$$c = \text{Enc}(k1, m)$$

t

$$t = \text{MAC}(m, k2)$$

- Mac Then Encrypt (SSL)

$$c = \text{Enc}(k1, m \parallel t)$$

$$t = \text{MAC}(m, k2)$$

- Encrypt Then Mac (IPSEC)

$$c = \text{Enc}(k1, m)$$

t

$$t = \text{MAC}(c, k2)$$

Confidencialidade e Autenticidade

- Teoria: Encrypt Then Mac
 - Robusto: funciona sempre desde que componentes sejam seguros independentemente
 - Seguro e eficiente: permite rejeitar criptograma antes de decifrar
 - Base para construções integradas de cifras autenticadas adotadas no TLS 1.3

$c = \text{Enc}(k1, m)$

t

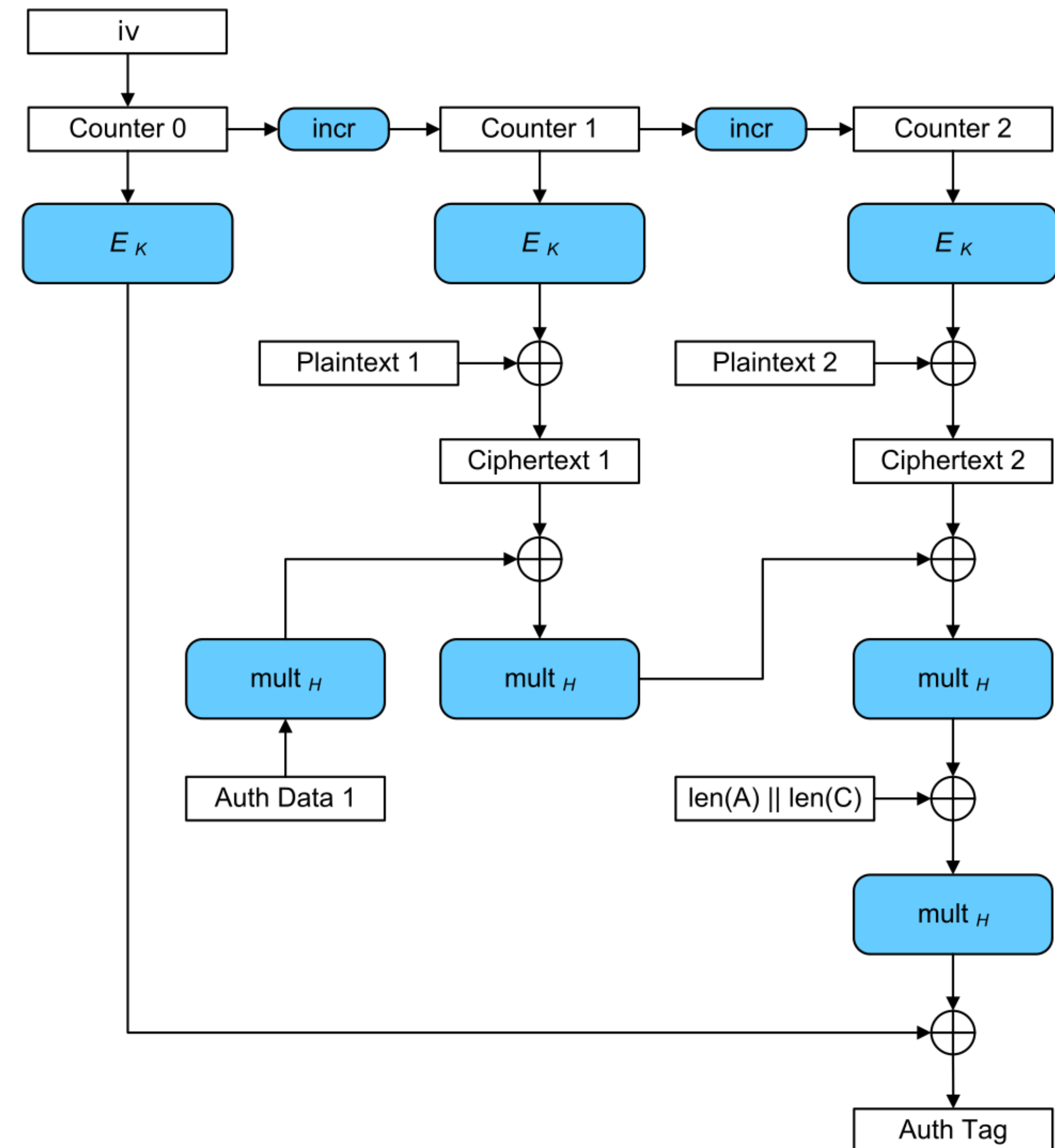
$t = \text{MAC}(c, k2)$

AEAD

- Authenticated Encryption with Associated Data
 - Abstração correta para a implementação de um canal seguro com criptografia simétrica
 - Garante a confidencialidade da mensagem/payload
 - Garante a autenticidade do criptograma e de metadados ($data$):
 - $Enc(n, k, m, data) \Rightarrow (c, t)$
 - $Dec(n, k, c, data) \Rightarrow m$, mas apenas se $(c, data)$ autênticos
 - chave pode ser usada múltiplas vezes se n não repetir
 - $data$ utilizada para vincular criptograma a contexto (e.g, endereço ip, número de sequência)
 - mais eficiente do que composição de cifra com MAC

AEAD: AES-GCM

- AES em Galois-Counter-Mode:
 - AEAD mais utilizado
 - muito eficiente
 - suporte em HW



AEAD: ChaCha20-Poly1305

- Alternativa a AES GCM recomendado no TLS 1.3
- Vantajosa em implementações SW
- É essencialmente um Encrypt-Then-Mac, mas
 - chave do MAC/Poly1305 := início da key stream ChaCha20
 - MAC/Poly1305 calculado sobre metadados + criptograma

Aleatoriedade

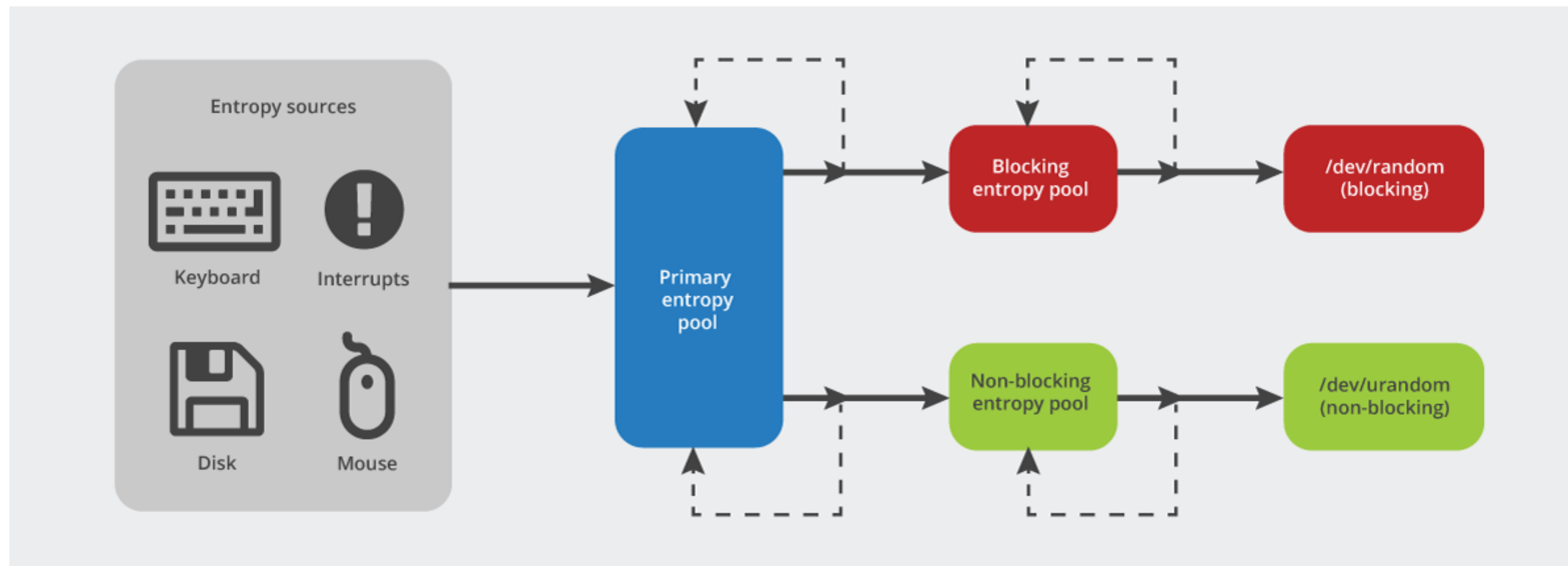
- De onde vêm os nonces, chaves, etc.?
- Na análise teórica de cripto geralmente assume-se aleatoriedade perfeita:
 - sequências de bits
 - cada bit é independente de todos os outros
 - a probabilidade de ser 1/0 é exatamente 50%
- Na prática gerar aleatoriedade é difícil

Aleatoriedade

- Mecanismos mais comuns nos sistemas operativos:
 - fontes de entropia:
 - medição de strings de bits a partir de processos físicos
 - temperatura, atividade do processador, atividade do utilizador
 - geradores pseudo-aleatórios:
 - estado: inicializado e actualizado periodicamente com fontes de entropia
 - algoritmo realimentado invocado quando SO precisa:
 - produz bits pseudoaleatórios, actualiza estado
- segurança destas construções é heurística: best effort, documentação de ataques (e.g., cold boot)

Aleatoriedade

- Exemplo Linux /dev/urandom



<https://unix.stackexchange.com/questions/324209/when-to-use-dev-random-vs-dev-urandom>