(Applied) Cryptography

Week #6: Authenticated Encryption

Manuel Barbosa, mbb@fc.up.pt

MSI/MCC/MERSI – 2022-2023

DCC-FCUP

Part #1: Authenticated Encryption

Why authenticated encryption

Any secure channel in practice uses authenticated encryption:

- Messages need to be confidential
- Messages need to be authentic
- Messages should not be repeated/removed/omitted

Encryption provides confidentiality.

MACs provide authenticity.

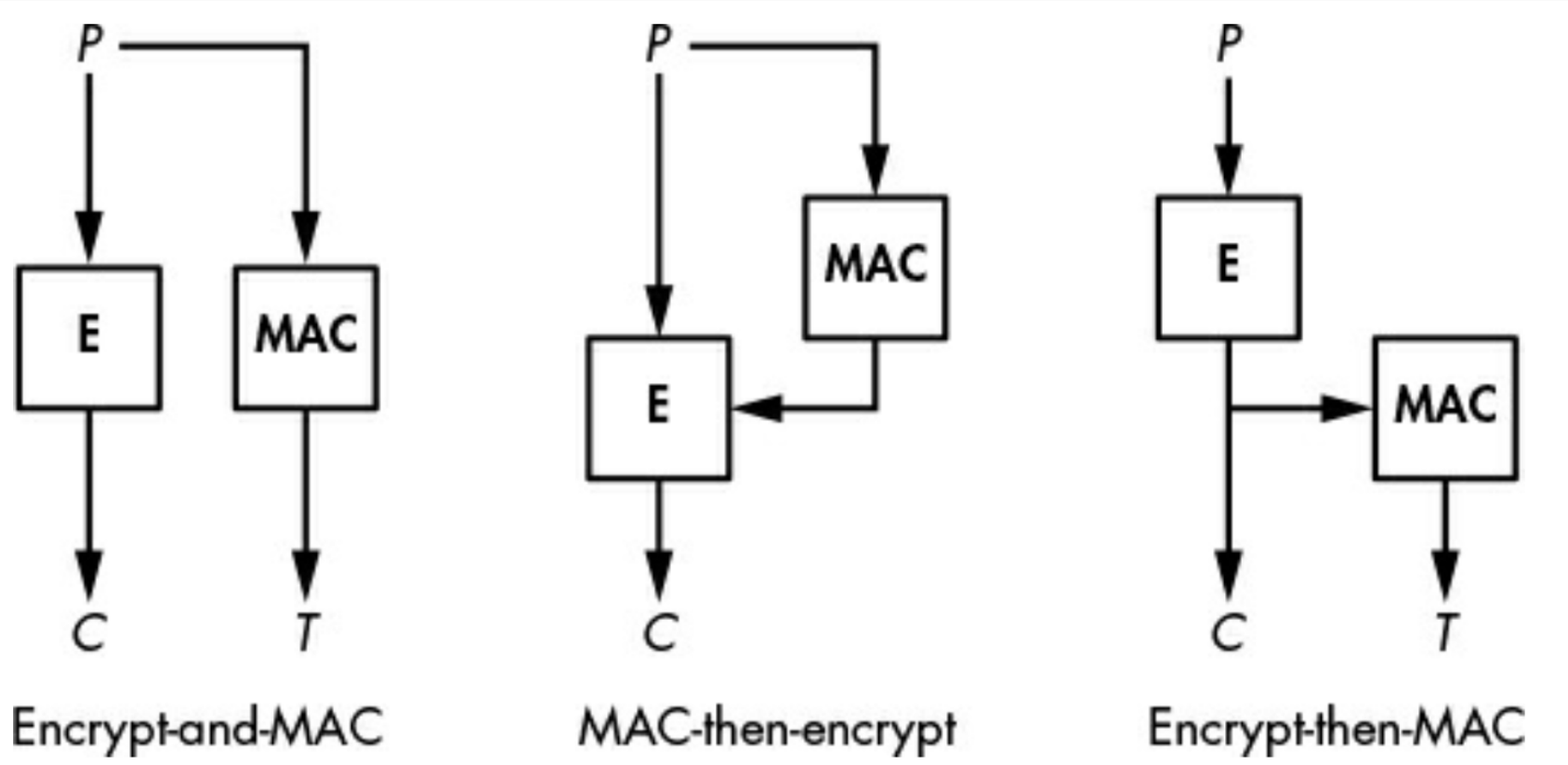
Authenticated public meta-information (e.g., sequence numbers) is used to solve the third point.

Authenticated Encryption with Associated Data (AEAD) guarantees all three.

This abstraction was gradually recognized as the correct one for protocols such as TLS.

Authenticated Encryption using MACs

Authenticated Encryption (w/out AD) using encryption and a MAC:



Encrypt and MAC

AE with a key $K = (K_1, K_2)$ can be done with parallel processing:

- $C \leftarrow\!\!\ll Enc(K_1, M)$
- $T \leftarrow \mathbf{MAC}(K_2, M)$

Output (C, T).

AuthDec cannot be parallel (why?): only accepts if MAC verifies.

Problems:

- Potentially malicious C decrypted before authenticated
- MACs are not designed to ensure confidentiality!
- Construction can be secure for some MACs, but not recommended
- Used in SSH: $\mathbf{MAC}(K_2, M \| N)$ where N is sequence number

MAC then Encrypt

AE with a key $K = (K_1, K_2)$ done sequentially:

- $T \leftarrow \mathbf{MAC}(K_1, M)$
- $C \twoheadleftarrow \mathbf{Enc}(K_2, M \| T)$

Output C.

AuthDec recovers message: only accept if MAC verifies.

Problems:

- Potentially malicious C decrypted before authenticated
- Used in TLS until version 1.3; painful story with padding oracle attacks

Encrypt then MAC

AE with a key $K = (K_1, K_2)$ done sequentially:

- $C \xleftarrow{\$} \mathbf{Enc}(K_1, M)$
- $T \leftarrow \mathbf{MAC}(K_1, C)$

AuthDec first verifies MAC: only decrypts C if successful.

Advantages:

- Ciphertext not decrypted unless it is authenticated
- Useful, e.g., against DoS attacks (MAC verification typically very fast)
- Preferred composition method except in legacy systems

Authenticated Encryption and its Security

AE is used everywhere: some constructions optimize the whole process.

These are called authenticated ciphers.

The syntax is consistent with the compositions we saw before:

- Authenticated Encryption: $(C, T) \twoheadleftarrow \mathbf{AEnc}(K, M)$
- Authenticated Decryption: $M/\perp \leftarrow \mathbf{ADec}(K, C, T)$

In a correct scheme, **ADec** should recover M from (C, T)

AE should provide the guarantees of encryption and authentication:

- IND-CPA security defined exactly as for encryption schemes
- Ciphertext unforgeability: not knowing K it is impossible to find new (C, T) that is accepted by **ADec**

Authenticated Encryption with Associated Data

The syntax is extended to include public meta-data:

- AEAD Encryption: $(C, T) \twoheadleftarrow \mathbf{AEnc}(K, A, M)$
- AEAD Decryption: $M/\perp \leftarrow \mathbf{ADec}(K, A, C, T)$

In a correct scheme, **ADec** should recover M from (C, T) **if same** A **provided at encryption and decryption**

AEAD should provide the guarantees of encryption and authentication:

- IND-CPA security defined exactly as for encryption schemes
- Authentication is extended to cover meta-data
- Unforgeability: not knowing K it is impossible to find new (A, C, T) that is accepted by **ADec**

If A is ommitted, we recover a standard AE scheme.

Nonce-Based AEAD

AEAD schemes can be constructed deterministically by using nonces.

The syntax is extended to include a nonce:

- AEAD Encryption: $(C, T) \leftarrow \mathbf{AEnc}(K, N, A, M)$
- AEAD Decryption: $M/\perp \leftarrow \mathbf{ADec}(K, N, A, C, T)$

In a correct scheme, **ADec** should recover M from (C, T) **if same** N **and** A **provided at encryption and decryption**

Security notions are the same, but:

- Attacker gets to choose the nonces in encryption and decryption
- It is restricted to *not repeat nonces* when asking for encryptions

There are no security guarantees if nonces repeat!

(Some ciphers degrade more graciously than others)

Part #2: Optimized Authenticated Encryption Constructions

Authenticated Encryption from Scratch

Modern AEADs are not black-box compositions of encryption/MAC.

Encryption/authentication layers visible in all constructions.

But they are optimized for performance:

- Encryption/decryption of blocks performed in parallel
- Throughput, *streamability*, memory requirements, input fixing:
  - implementation cost, e.g., need block cipher inverse?
  - can start transmitting before encryption complete?
  - can start decrypting before ciphertext fully received?
  - can discard plaintext/ciphertext block immediately (aka online)?
  - Can metadata be given at any point? Or must be fixed initially?

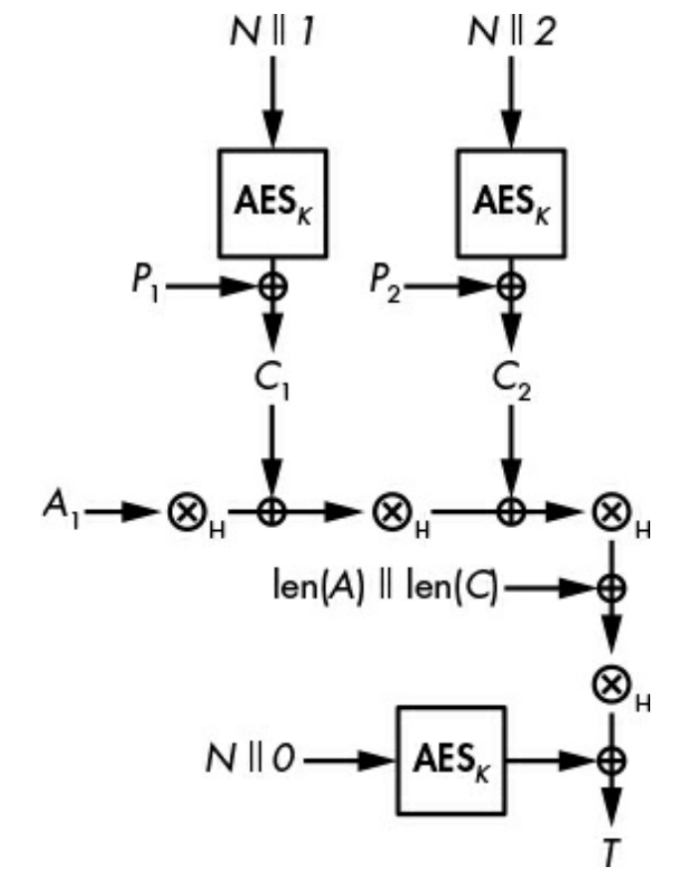
Streamability (aka online) $\Rightarrow$ low memory requirements:

- Very important for routers, TLS/https termination, etc.

Galois-Counter Mode (GCM)

AES-GCM is the most widely used AEAD: IPSec, SSH, TLS.

GCM = Counter-Mode with an authentication layer on top.



Key: 128-bits; Nonce: 96-bits; CTR starts at 1. Max P length?

Galois-Counter Mode (GCM): Autenticação

Tag computed over ciphertext blocks: encrypt then MAC.

MAC uses Wegman-Carter construction:

- Universal Hash function is called GHASH
- AES used as PRF to mask the hash value on input $N\|0$

GHASH$(HK, (C_1, \ldots, C_n))$ is defined as follows:

- $HK \leftarrow$ **AES**$(K, 0)$
- Evaluate $P(X)$ defined by $(\text{pad0}(A), \text{pad0}(C), |A|\|\|C|)$ at HK
- Horner's formula: $P(X) := X * (X * (X * (\ldots) + a_2) + a_1) + a_0$
- Algebraic magic:
  - operations defined over a Galois field (similar to LFSR)
  - super efficient in hardware (special processor instructions)

GCM Efficiency

Encryption layer inherits parallelism from CTR mode.

Authentication layer blocks if A known only in the end.

If A is fixed from the start, then GCM is streamable:

- Ciphertext blocks computed and authenticated on the fly
- Authentication of previous block is accumulated while current block is encrypted

Could be faster:

- Authentication usually not computed fully in parallel.
- HW implementations of auth layer slower than AES-CTR!

(We won't see ChaCha20-Poly1305, but has similar structure.)

Syntethic IV Mode (SIV)

Synthetic IV: security under nonce-reuse.

Generic composition of nonce-based encryption and PRF:

- $T \leftarrow \mathbf{PRF}(K_1, A \| P \| N)$
- $C \leftarrow \mathbf{Enc}(K_2, N = T, P)$
- Output (C, T)

Tag is used as encryption nonce and will be fresh w/ high prob.

What happens if:

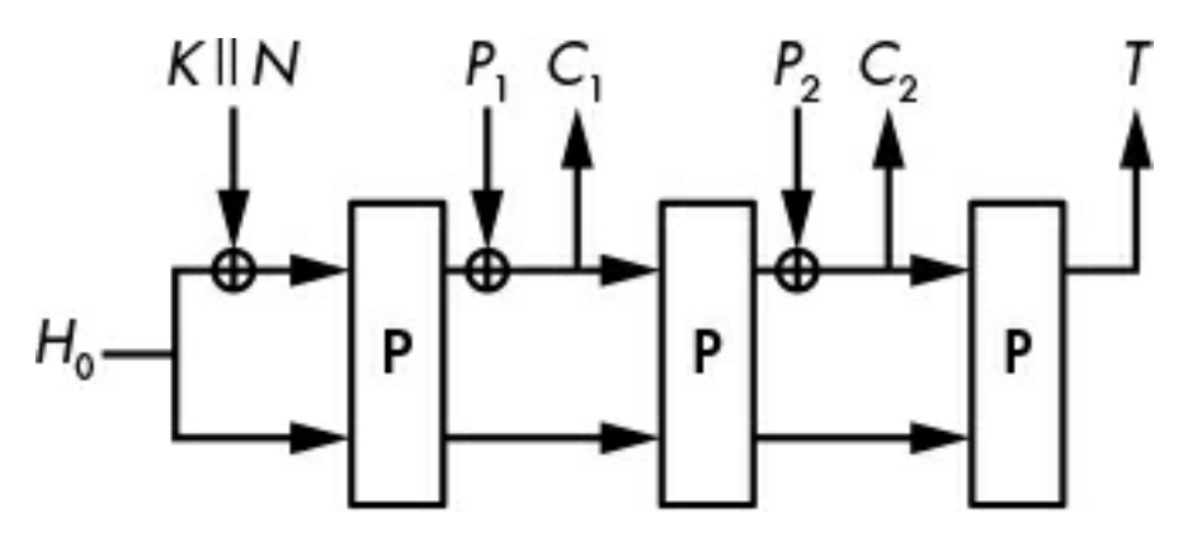
- N repeats but A or P changes?
- All of A, N, P repeat?

This scheme is not streamable: all of P must be stored!

Building AE from Permutations

Recall the Sponge construction used in SHA-3.

A closely related construction, called Duplex, gives an AEAD:



P is a fixed (unkeyed) permutation and H_0 is a fixed public value.

Last block must be padded: different lengths $\Rightarrow$ different tags.

AEAD version is slightly more involved, so we won't cover it.

Building AE from Permutations (2)

The resulting constructions are:

- Fast
- Streamable

Interesting nonce reuse resilience:

- Unforgeability not affected
- Plaintexts compromised:
  - first block
  - subsequent blocks if common prefix
- Plaintexts still hidden after plaintexts diverge

Thank you!

mbb@fc.up.pt

http://www.dcc.fc.up.pt/~mbb