Applied Cryptography
Week 6: Authenticated Encryption

Bernardo Portela

M:ERSI, M:SI, M:CC - 24

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/omitted/removed

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/omitted/removed

Encryption provides confidentiality

MACs provide authenticity

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/omitted/removed

Encryption provides confidentiality

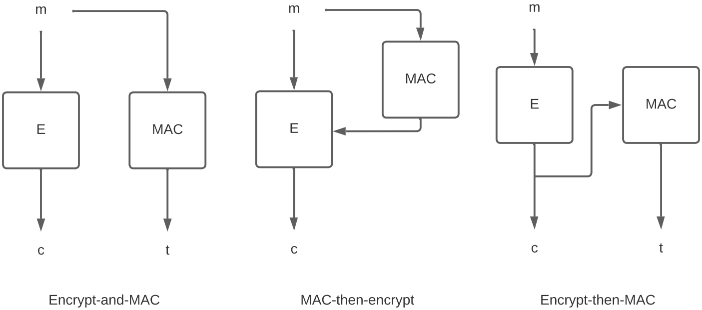
MACs provide authenticity

Authenticated public meta-information (e.g. sequence numbers) is used to solve the third point – Authenticated Encryption with Associated Data (AEAD)

Authenticated Encryption using MACs

The good, the bad and the ugly

- Encrypt-and-MAC: encryption and MAC of the message
- MAC-then-Encrypt: Encrypt message and its authentication
- Encrypt-then-MAC: Authenticate the message encryption



Encrypt-and-MAC

AE with $k = (k_1, k_2)$ done with parallel processing

- $c \leftarrow_{\$} E(k_1, m)$
- $t \leftarrow MAC(k_2, m)$
- Output: (c, t)

Encrypt-and-MAC

AE with $k = (k_1, k_2)$ done with parallel processing

- $c \leftarrow_{\$} E(k_1, m)$
- $t \leftarrow MAC(k_2, m)$
- Output: (c, t)

Problems – The bad

- Potentially malicious c decrypted before authentication
- MACs not designed to ensure confidentiality!
- Construction can be secure for *some* MACs...
  - Very easy to make mistakes
- Used in SSH: $MAC(k_2, m||n)$, where n is the sequence number

MAC-then-Encrypt

AE with $k = (k_1, k_2)$ done sequentially

- $t \leftarrow MAC(k_1, m)$
- $c \leftarrow_{\$} E(k_2, m||t)$
- Output: c

MAC-then-Encrypt

AE with $k = (k_1, k_2)$ done sequentially

- $t \leftarrow MAC(k_1, m)$
- $c \leftarrow_\$ E(k_2, m||t)$
- Output: c

Problems – The ugly

- Potentially malicious c decrypted before authentication
- Used in TLS until version 1.3
- Painful story with *padding oracle attacks*
  - Issue arises from the decryption before authentication
  - Did the decryption fail because of the padding, or because of the MAC?
  - Theoretical attack found – disregarded at first
  - Practical attack found a couple of years later: Lucky 13

Encrypt-then-MAC

AE with $k = (k_1, k_2)$ done sequentially

- $c \leftarrow_{\$} E(k_1, m)$
- $t \leftarrow MAC(k_2, c)$
- Output: c

Encrypt-then-MAC

AE with $k = (k_1, k_2)$ done sequentially

- $c \leftarrow_{\$} E(k_1, m)$
- $t \leftarrow MAC(k_2, c)$
- Output: c

Advantages – The good

- Ciphertext not decrypted **unless it is authenticated**
- Useful against DoS attacks
  - MAC verification typically very fast
- Preferred method, except in legacy systems

Security of Authenticated Encryption

AE is used everywhere: some constructions optimize the whole process: authenticated ciphers

Security of Authenticated Encryption

AE is used everywhere: some constructions optimize the whole process: authenticated ciphers

Syntax is consistent with the compositions saw before:

- Encryption: $(c, t) \leftarrow_{\$} AEnc(k, m)$
- Decryption: $m/\perp \leftarrow AEnc(k, c, t)$

Security of Authenticated Encryption

AE is used everywhere: some constructions optimize the whole process: authenticated ciphers

Syntax is consistent with the compositions saw before:

- Encryption: $(c, t) \leftarrow\!\!\$\ AEnc(k, m)$
- Decryption: $m/\perp \leftarrow AEnc(k, c, t)$

Correctness: $\forall m\,,\ m = ADec(k, AEnc(k, m))$

Security:

- Adversary cannot distinguish two encryptions of any message
- Adversary cannot forge new messages
- Security experiment: IND-CCA

Indistinguishability against Chosen Ciphertext Attacks

Changes from IND-CPA in blue

Security – (IND-CCA1 and IND-CCA2)

- Experiment samples k and bit b uniformly at random
- Attacker can query encryptions of chosen messages
- Attacker can query decryptions of chosen ciphertexts
- Attacker outputs (m_0, m_1) s.t. $|m_0| = |m_1|$
- Attacker gets $c \leftarrow_{\$} E(k, m_b)$
- (CCA2) Attacker can query enc/dec of chosen ciphertexts
- (CCA2) (...) but not of c!
- Attacker outputs b' and wins if $b = b'$

Advantage: $|\Pr[b = b'] - \frac{1}{2}|$

Authenticated Encryption with Associated Data

The syntax is extended to include public meta-data

- AEAD Encryption: $(c, t) \leftarrow\$ \, AEnc(k, a, m)$
- AEAD Decryption: $m/\perp \leftarrow ADec(k, a, c, t)$

Authenticated Encryption with Associated Data

The syntax is extended to include public meta-data

- AEAD Encryption: $(c, t) \leftarrow\$ AEnc(k, a, m)$
- AEAD Decryption: $m/\perp \leftarrow ADec(k, a, c, t)$

Correctness: $\forall m\,,\ m = ADec(k, a, AEnc(k, a, m))$

Security:

- 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 a new (a, c, t) for which $m \leftarrow ADec(k, a, c, t)$, $m \neq \perp$

If we take a out, we get exactly authenticated encryption!

Nonce-Based AEAD

AEAD schemes constructed deterministically by using nonces

Nonce-Based AEAD

AEAD schemes constructed deterministically by using nonces

The syntax is extended to include a nonce

- AEAD Encryption: $(c, t) \leftarrow AEnc(k, n, a, m)$
- AEAD Decryption: $m/\perp \leftarrow ADec(k, n, a, c, t)$

Nonce-Based AEAD

AEAD schemes constructed deterministically by using nonces

The syntax is extended to include a nonce

- AEAD Encryption: $(c, t) \leftarrow AEnc(k, n, a, m)$
- AEAD Decryption: $m/\perp \leftarrow ADec(k, n, a, c, t)$

Correctness: $\forall m\,,\ m = ADec(k, n, a, AEnc(k, n, a, m))$

Security:

- The same, but...
- Attacker chooses the nonces in encryption and decryption
- It is restricted to *to not repeat nonces* when asking for encryption – week #3 extra
  - No security guarantees if **nonces repeat**!

Key Takeaways

- Authenticated Encryption ensures confidentiality and integrity
- Necessary in the majority of real-world use cases

Key Takeaways

- Authenticated Encryption ensures <u>confidentiality</u> and <u>integrity</u>
- Necessary in the majority of real-world use cases

- Encrypt-and-MAC
  - Exposed MAC. Not good!
- MAC-then-Encrypt
  - Possible decryption of malicious ciphertexts
  - Not necessarily broken; has subtle issues
- Encrypt-then-MAC
  - The “safest” way to do it

Key Takeaways

- Authenticated Encryption ensures confidentiality and integrity
- Necessary in the majority of real-world use cases

- Encrypt-and-MAC
  - Exposed MAC. Not good!
- MAC-then-Encrypt
  - Possible decryption of malicious ciphertexts
  - Not necessarily broken; has subtle issues
- Encrypt-then-MAC
  - The "safest" way to do it

- Security model allows encryptions and decryptions
- AEAD (AE with associated data)
- Some data with integrity, but not encrypted
  - Protocol parameters; message context; etc

Authenticated Encryption from Scratch

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

Encryption/authentication layers visible in all constructions

Authenticated Encryption from Scratch

Modern AEADs are not black-box compositions of enc/MAC.
Encryption/authentication layers visible in all constructions

Optimized for Performance

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

Authenticated Encryption from Scratch

Modern AEADs are not black-box compositions of enc/MAC.
Encryption/authentication layers visible in all constructions

Optimized for Performance

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

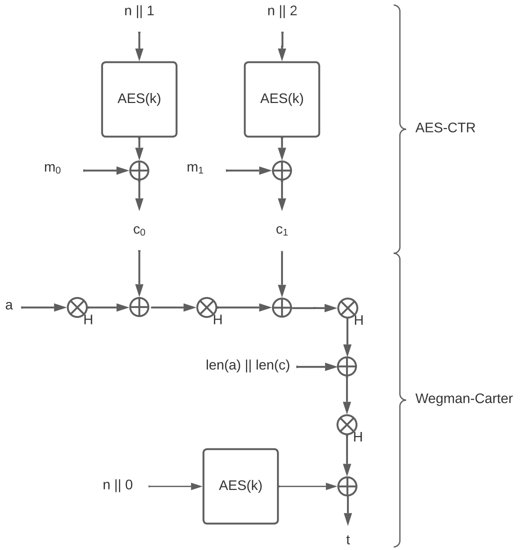
Streamability (a.k.a. online) $\rightarrow$ low memory requirements

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

Galois-Counter Mode

Most widely used AEAD:
IPSec, SSH, TLS

- GCM:
  - CTR plus auth layer
- Keys: 128 bits
- Nonce: 96 bits
- CTR starts at 1



Galois-Counter Mode: Authentication

Q: What type of AE is this?

Galois-Counter Mode: Authentication

Q: What type of AE is this? Encrypt-then-MAC!

MAC uses Wegman-Carter construction

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

Galois-Counter Mode: Authentication

Q: What type of AE is this? Encrypt-then-MAC!

MAC uses Wegman-Carter construction

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

$GHASH(hk, c_1, \ldots, c_n)$ 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)

Efficiency of Galois Counter Mode

Encryption layer inherits parallelism from CTR mode

Authentication layer blocks if *a* is known only in the end

Efficiency of Galois Counter Mode

Encryption layer inherits parallelism from CTR mode

Authentication layer blocks if a is known only in the end

a known from the start $\rightarrow$ GCM streamable

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

Efficiency of Galois Counter Mode

Encryption layer inherits parallelism from CTR mode

Authentication layer blocks if a is known only in the end

a known from the start $\rightarrow$ GCM streamable

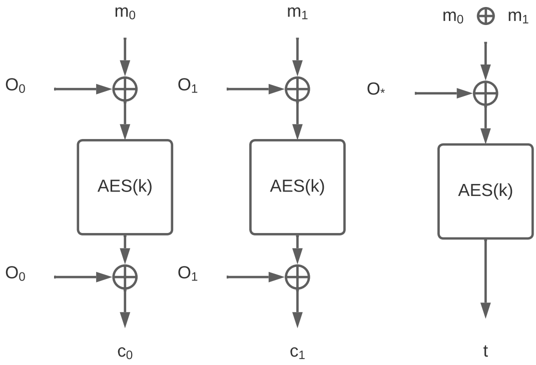
- Ciphertext blocks computed and authenticated on the fly
- Authentication of previous block is accumulated while current block is being encrypted

Could be faster...

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

(ChaCha20-Poly1305 not covered, but has similar structure)

Offset Codebook (OCB)



- Offset depends on the key and the nonce
- Incremented for each new block
- Last offset computed from last plaintext block processed

Using OCB as AEAD

- Very secure and efficient
- Implementations required licensing
- Patent-renewal fees intentionally not payed (see here)

Using OCB as AEAD

- Very secure and efficient
- Implementations required licensing
- Patent-renewal fees intentionally not payed (see here)

Similar to AES-GCM, OBC allows authenticated data $a_0, a_1, \ldots, a_n$

Tag computed as

$$T = E(k, S \oplus O_*) \oplus E(k, a_0 \oplus O_0) \oplus E(k, a_1 \oplus O_1) \oplus \ldots$$

where $S = m_0 \oplus m_1 \oplus \ldots$

Offset values for AD different from those used to encrypt $m_0, m_1, \ldots, m_n$

Security and Efficiency of OCB

Nonce reuse

- On nonce reuse, the attacker can identify block duplicates
  - E.g. block 3 of message 1 is similar to block 5 of message 2
  - GCM allows detection of multiple blocks, but also XOR differences for blocks in the same position
- Repeated nonces can break the authenticity of OCB
  - An attacker can combine blocks from messages authenticated with OCB to create another authenticated message
  - ...but unlike GCM, it cannot extract the underlying key!!

Security and Efficiency of OCB

Nonce reuse

- On nonce reuse, the attacker can identify block duplicates
  - E.g. block 3 of message 1 is similar to block 5 of message 2
  - GCM allows detection of multiple blocks, but also XOR differences for blocks in the same position
- Repeated nonces can break the authenticity of OCB
  - An attacker can combine blocks from messages authenticated with OCB to create another authenticated message
  - ...but unlike GCM, it cannot extract the underlying key!!

Efficiency

- OCB and GCM make about as many calls to the block cipher
- GCM used to be 3x slower than OCB
  - AES and GHASH competed for CPU resources
- OCB requires encryption and decryption, contrary to GCM

Synthetic IV Mode (SIV)

Strengthening AES-GCM against nonce-reuse.

Generic composition of nonce-based encryption and PRF

- $t \leftarrow PRF(k_1, a \,||\, p \,||\, n)$
- $c \leftarrow Enc(k_2, n = t, p)$
- Output (c, t)

Synthetic IV Mode (SIV)

Strengthening AES-GCM against nonce-reuse.

Generic composition of nonce-based encryption and PRF

- $t \leftarrow PRF(k_1, a \,||\, p \,||\, n)$
- $c \leftarrow Enc(k_2, n = t, p)$
- Output (c, t)

Q1: What happens if?

- n **repeats, but** a **or** p **changes?**

Synthetic IV Mode (SIV)

Strengthening AES-GCM against nonce-reuse.

Generic composition of nonce-based encryption and PRF

- $t \leftarrow PRF(k_1, a \,||\, p \,||\, n)$
- $c \leftarrow Enc(k_2, n = t, p)$
- Output (c, t)

Q1: What happens if?

- **n repeats, but a or p changes?**
- **All of a, n and p repeat?**

Synthetic IV Mode (SIV)

Strengthening AES-GCM against nonce-reuse.

Generic composition of nonce-based encryption and PRF

- $t \leftarrow PRF(k_1, a \,||\, p \,||\, n)$
- $c \leftarrow Enc(k_2, n = t, p)$
- Output (c, t)

Q1: What happens if?

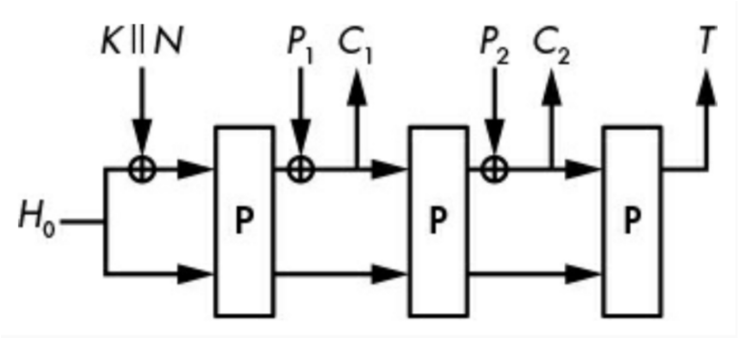
- **n repeats, but a or p changes?**
- **All of a, n and p repeat?**

- Tag used as encryption nonce is fresh w/ high probability
- Scheme is not streamable! **Q2: Why?**

Building AE from Permutations

Recall the sponge construction of SHA-3

Closely related construction – Duplex – gives an AEAD



- P is a fixed (unkeyed) permutation and h_0 is a public value
- Last block must be padded
- AEAD versions slightly more involved. Not covered

Permutation-based AEs

Resulting constructions are

- Fast
- Streamable

Permutation-based AEs

Resulting constructions are

- Fast
- Streamable

Interesting nonce reuse resilience:

- Unforgeability not affected
- Plaintexts compromised
    - First block
    - Subsequent block if there is a common prefix
- Plaintexts remain confidential after divergence

Request For Comments

Internet Engineering Task Force (IETF)
Request for Comments: 8446
Obsoletes: 5077, 5246, 6961
Updates: 5705, 6066
Category: Standards Track
ISSN: 2070-1721

E. Rescorla
Mozilla
August 2018

The Transport Layer Security (TLS) Protocol Version 1.3

Abstract

This document specifies version 1.3 of the Transport Layer Security (TLS) protocol. TLS allows client/server applications to communicate over the Internet in a way that is designed to prevent eavesdropping, tampering, and message forgery.

This document updates RFCs 5705 and 6066, and obsoletes RFCs 5077, 5246, and 6961. This document also specifies new requirements for TLS 1.2 implementations.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8446.

Unambiguous Configurations

Development by Consortium

- Internet Engineering Task Force (IETF)
- Establish how to implement secure systems
- Define multiple criteria
  - E.g. ciphersuites

This specification defines the following cipher suites for use with TLS 1.3.

Description	Value
TLS_AES_128_GCM_SHA256	{0x13,0x01}
TLS_AES_256_GCM_SHA384	{0x13,0x02}
TLS_CHACHA20_POLY1305_SHA256	{0x13,0x03}
TLS_AES_128_CCM_SHA256	{0x13,0x04}
TLS_AES_128_CCM_8_SHA256	{0x13,0x05}

Key Takeaways

- General usage $\rightarrow$ ambitious requirements
  - Implementation cost; Streamability; ...

Key Takeaways

- General usage $\rightarrow$ ambitious requirements
  - Implementation cost; Streamability; ...
- AES-GCM
  - AES-CTR followed by a round of Wegman-Carter
  - ChaCha20-Poly1305 similar

Key Takeaways

- General usage $\rightarrow$ ambitious requirements
  - Implementation cost; Streamability; ...
- AES-GCM
  - AES-CTR followed by a round of Wegman-Carter
  - ChaCha20-Poly1305 similar
- OCB
  - Offset XOR'd with AES input/output
  - Final offet produces authentication tag
  - Licensing issues hindered usability

Key Takeaways

- General usage $\rightarrow$ ambitious requirements
  - Implementation cost; Streamability; ...
- AES-GCM
  - AES-CTR followed by a round of Wegman-Carter
  - ChaCha20-Poly1305 similar
- OCB
  - Offset XOR'd with AES input/output
  - Final offet produces authentication tag
  - Licensing issues hindered usability
- AEAD from permutations
  - Absorb phase from Sponge; gets blocks between permutations
  - Authentication tag recovered at the end

Key Takeaways

- General usage → ambitious requirements
  - Implementation cost; Streamability; ...
- AES-GCM
  - AES-CTR followed by a round of Wegman-Carter
  - ChaCha20-Poly1305 similar
- OCB
  - Offset XOR'd with AES input/output
  - Final offet produces authentication tag
  - Licensing issues hindered usability
- AEAD from permutations
  - Absorb phase from Sponge; gets blocks between permutations
  - Authentication tag recovered at the end
- Request for Comments
  - Key for real-world usage
  - TLS, IPSec, PQC
  - Sets ciphersuites, parameters, implementation details

Applied Cryptography
Week 6: Authenticated Encryption

Bernardo Portela

M:ERSI, M:SI, M:CC - 24