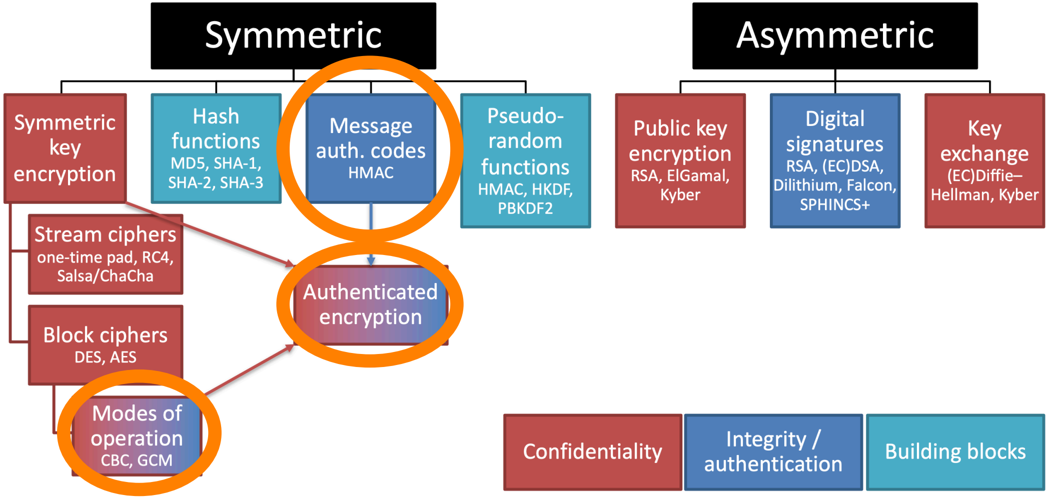
Topic 2.2
Integrity – Message authentication codes and authenticated encryption

Douglas Stebila

CO 487/687: Applied Cryptography

Fall 2024

Map of cryptographic primitives



Outline

Outline

Definition

Definition (Message authentication code)

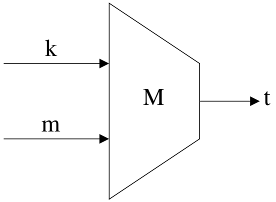
A message authentication code (MAC) scheme is an efficiently computable function

$$M\colon \{0,1\}^{\ell} \times \{0,1\}^{*} \to \{0,1\}^{n}$$

written

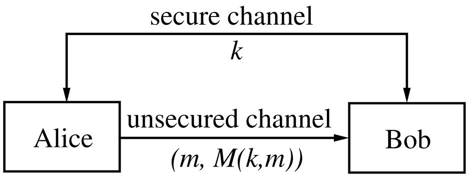
$$M(k, m) = t$$

where k is the key, m is the message, and t is the tag.



MAC schemes are used for providing (symmetric-key) data integrity and data origin authentication.

Applications of Message Authentication Codes



To provide data integrity and data origin authentication:

1. Alice and Bob establish a secret key $k \in \{0,1\}^\ell$.
2. Alice computes $t = M(k,m)$ and sends (m,t) to Bob.
3. Bob verifies that $t = M(k,m)$.

To avoid replay, add a timestamp, or sequence number.

Widely used in communication protocols.

No confidentiality or non-repudiation.

Security definition

Let k be the secret key shared by Alice and Bob.

The adversary knows everything about the MAC scheme except the value of k.

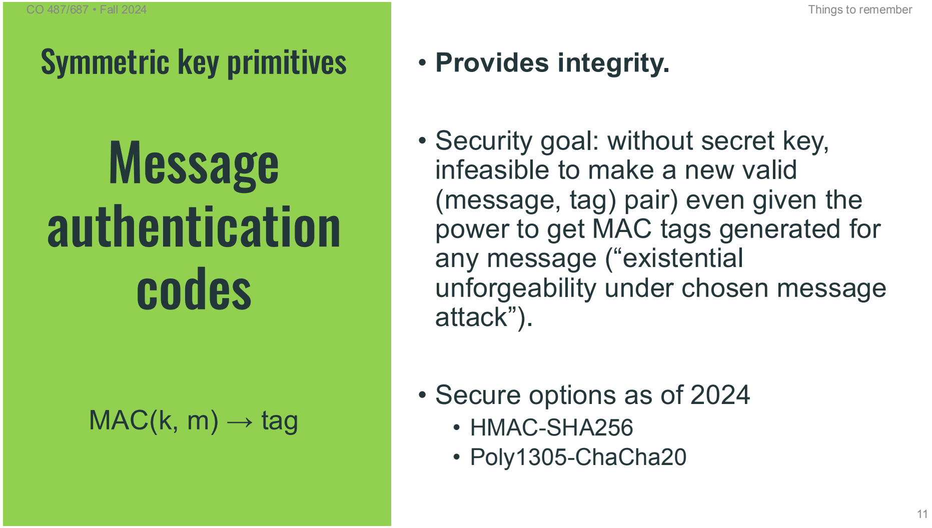
Definition (MAC security)

A MAC scheme is secure if:

- Given some number of MAC tags $M(k, m_i)$ for messages m_i chosen adaptively by the adversary (interaction),
- it is computationally infeasible (computational resources)
- to compute (with non-negligible probability of success) the value of $M(k, m)$ for any message $m \neq m_i$ (goal).

In other words, a MAC scheme is secure if it is existentially unforgeable against chosen-message attack.

4+8+7 things to remember from CO 487



CO 487/687 • Fall 2024

Things to remember

Symmetric key primitives

Message authentication codes

MAC(k, m) → tag

- **Provides integrity.**
- Security goal: without secret key, infeasible to make a new valid (message, tag) pair) even given the power to get MAC tags generated for any message ("existential unforgeability under chosen message attack").
- Secure options as of 2024
  - HMAC-SHA256
  - Poly1305-ChaCha20

11

Generic attacks

Guessing the MAC of a message m:

- Select $y \in \{0,1\}^n$ and guess that $M(k, m) = y$.
- Assuming that $M(k, \cdot)$ is a random function, the probability of success is $1/2^n$.
- Note: Guesses cannot be directly checked without interaction.
- Depending on the application where the MAC algorithm is employed, one could choose n as small as 32 (say). In general, $n \geq 128$ is preferred.

Generic attacks

Exhaustive search on the key space:

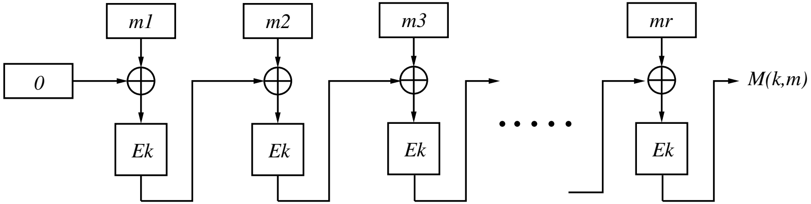
- Given r known message-MAC pairs: $(m_1, t_1), \ldots, (m_r, t_r)$, one can check whether a guess k of the key is correct by verifying that $M(k, m_i) = t_i$, for $i = 1, 2, \ldots, r$.
- Assuming that the $M(k, \cdot)$'s are random functions, the expected number of keys for which the tags verify is $K = 2^\ell / 2^{nr}$.
    - Example: If $\ell = 56$, $n = 64$, $r = 2$, then $K \approx 1/2^{72}$.
- Requires $\approx 2^\ell$ computations.
- Exhaustive search is infeasible if $\ell \geq 128$.

Outline

MACs Based on Block Ciphers

CBC-MAC

- Let E be an n-bit block cipher with key space $\{0,1\}^\ell$.
- To compute $M(k, m)$:
    1. Divide (padded) m into n-bit blocks $m_1, m_2, \ldots, m_r$
    2. Compute $H_1 = E(k, m_1)$.
    3. For $2 \le i \le r$, compute $H_i = E(k, H_{i-1} \oplus m_i)$.
    4. Then $M(k, m) = H_r$.



CBC-MAC

- DES CBC-MAC (with $\ell = 56, n = 64$) was widely used in banking applications.
    - ANSI X9.9: Financial institution message authentication (wholesale).
    - ANSI X9.19: Financial institution message authentication (retail).

- Rigorous security analysis [Bellare, Kilian & Rogaway 1994]:
  Informal statement of a Theorem: Suppose that E is an “ideal” encryption scheme. (That is, for each $k \in \{0,1\}^\ell$, $E_k \colon \{0,1\}^n \to \{0,1\}^n$ is a ‘random’ permutation.) Then CBC-MAC with fixed-length inputs is a secure MAC algorithm.

Security of CBC-MAC

- CBC-MAC (as described here without additional measures) is not secure if variable length messages are allowed.
- Here is a chosen-message attack on CBC-MAC:
    1. Let m_1 be an n-bit block.
    2. Let (m_1, t_1) be a known message-MAC pair.
    3. Request the MAC t_2 of the message t_1.
    4. Then t_2 is also the MAC of the 2-block message $(m_1, 0)$. (Since $t_2 = E_k(E_k(m_1))$.)

Encrypted CBC-MAC (EMAC)

- One solution for variable-length messages is Encrypted CBC-MAC:
  - Encrypt the last block under a second key s:

    $$M((k,s),m) = E_s(H_r)$$

    where H_r is the output of CBC-MAC.
  - For example, ANSI X9.19 specifies that $M((k,s),m) = E_k(E_s^{-1}(H_r))$.

- Rigorous security analysis [Petrank & Rackoff 2000]:
  Informal statement of a Theorem: Suppose that E is an "ideal" encryption scheme. Then EMAC is a secure MAC algorithm (for inputs of any length).

MACs based on hash functions

Hash functions were not originally designed for message authentication; in particular they are not “keyed” primitives.

Question: How to use them to construct secure MACs?

MACs based on hash functions

Let H be an iterated n-bit hash function (without the final length block consisting of padding).

Let $n + r$ be the length of the input of the compression function $f\colon \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^n$.

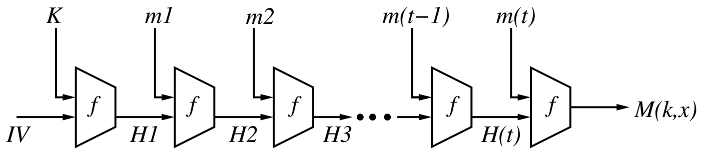
- Example: For SHA-1, $n = 160$, $r = 512$.

Let $k \in \{0,1\}^n$.

Let K denote k padded with $(r - n)$ 0's.
(So K has bitlength r.)

MACs based on hash functions: Secret prefix method

MAC definition: $M(k,m) = H(K\|m)$

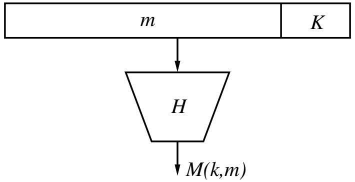


This is insecure. Here is a length extension attack:

- Suppose that $(m, M(k,m))$ is known.
- Suppose that the bitlength of m is a multiple of r.
- Then $M(k, m\|m')$ can be computed for any m' (without knowledge of k).

Also insecure if a length block is postpended to $K\|m$ prior to application of H. [Exercise]

MACs based on hash functions: Secret suffix method



MAC definition: $M(k, m) = H(m \| K)$

The attack on the secret prefix method does not work here.

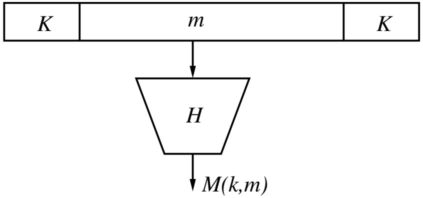
Here is a different attack which works if H is not collision resistant:

- Suppose that a collision (m_1, m_2) can be found for H (i.e., $H(m_1) = H(m_2)$). Assume that m_1 and m_2 both have bitlengths that are multiples of r.
- Then $H(m_1 \| K) = H(m_2 \| K)$, and so $M(k, m_1) = M(k, m_2)$.
- The MAC for m_1 can be requested, giving the MAC for m_2.
- Hence if H is not collision resistant, then the secret suffix method MAC is insecure.

MACs based on hash functions: Envelope method a.k.a. Sandwich method

The MAC key is used both at the start and end of the MAC computation.

MAC definition:
$M(k, m) = H(K \| m \| K)$



Theorem (Koblitz and Menezes, 2013)

Suppose that the compression function used in H is a secure MAC with fixed length messages and a secret IV as the key. Then Envelope MAC ***with m padded to a multiple of the block length of H*** *is a secure MAC algorithm.*

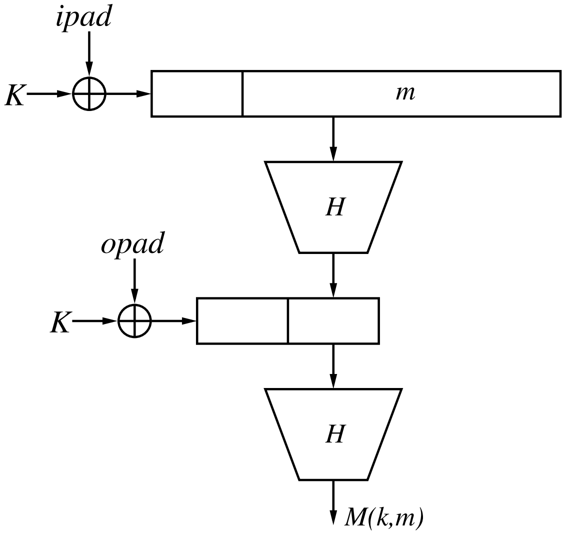
HMAC

"Hash-based" MAC; Bellare, Canetti & Krawczyk (1996).

Define 2 r-bit strings (in hexadecimal notation): ipad = 0x36, opad = 0x5C; each repeated $r/8$ times.

MAC definition:

$$M(k, m) = H\Big((K \oplus \text{opad}) \| H\big((K \oplus \text{ipad}) \| m\big)\Big).$$



HMAC

Rigorous security analysis of HMAC:

Theorem (Bellare, Canetti & Krawczyk, 1996)

Suppose that the compression function used in H is a secure MAC with fixed length messages and a secret IV as the key. Then HMAC is a secure MAC algorithm.

- Recommended for practice: use SHA-256 or better as the hash function.
    - The collision weaknesses in SHA-1 do not appear to affect the security of HMAC, but still best to use SHA-256 or better.
- HMAC is used in IPsec (Internet Protocol Security), TLS (Transport Layer Security), and many others.

Outline

Combining encryption and authentication

- In many situations, one wishes to achieve both data confidentiality and data authentication.
- Given a shared secret key:
    - A symmetric-key encryption scheme achieves confidentiality.
    - A message authentication code achieves data authentication.
- Combining the two is trivial. Right? Well, not really...

“people had been doing rather poorly when they tried to glue together a traditional (privacy-only) encryption scheme and a message authentication code (MAC)”

—M. Bellare, P. Rogaway, and D. Wagner

“it is very easy to accidentally combine secure encryption schemes with secure MACs and still get insecure authenticated encryption schemes”

—T. Kohno, J. Viega, and D. Whiting

Preliminaries

Some obvious remarks:

- Don’t use the same key for both sending and receiving.
    - Totally insecure with a stream cipher (one-time pad, etc.)
    - Still not a good idea even with a block cipher.
- Don’t use the same key for authentication and encryption.
    - Exception: First-class primitives for authenticated encryption are designed to use only one key.

Recommendation:

- Use separate keys for authentication and encryption.
- Use separate keys for each party in the conversation.
- Create keys with a key derivation function (HKDF, PBKDF2, scrypt).
- The key derivation function can use in part a common shared key as input.

Possible strategies

Let m be a message. There are (at least) three obvious ways to combine encryption and authentication:

- MAC-then-encrypt (MtE):

$$\text{compute } t = \mathrm{MAC}(m) \text{ and } c = \mathrm{Enc}(m\|t), \text{ transmit } c$$

- encrypt-then-MAC (EtM):

$$\text{compute } c = \mathrm{Enc}(m) \text{ and } t = \mathrm{MAC}(c), \text{ transmit } c\|t$$

- encrypt-and-MAC (E&M):

$$\text{compute } c = \mathrm{Enc}(m) \text{ and } t = \mathrm{MAC}(m), \text{ transmit } c\|t$$

These constructions are called *composed primitives* because they combine two primitives into one.

Security goals

Confidentiality: semantic security under chosen plaintext or chosen ciphertext attack

Integrity: existential unforgeability under chosen message attack

Examples

- MAC-then-encrypt: SSL/TLS up to v1.2
- encrypt-then-MAC: IPsec
- encrypt-and-MAC: SSH

Encrypt and MAC (E&M)

Consider encrypt-and-MAC:

$$\text{compute } c = \mathrm{Enc}(m) \text{ and } t = \mathrm{MAC}(m), \text{ transmit } c \| t$$

When decrypting, the recipient checks that the MAC is correct.

- Problem: MACs are not required to ensure confidentiality.
    - For example, the MAC might leak one plaintext bit, and still be “secure” as a MAC. Violates semantic security!
- Even if the SKES is secure and the MAC is secure, the encrypt-and-MAC combination might be insecure.

MAC then encrypt (MtE)

Consider MAC-then-encrypt:

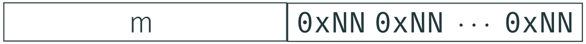
$$\text{compute } t = \mathrm{MAC}(m) \text{ and } c = \mathrm{Enc}(m \| t), \text{ transmit } c$$

When decrypting, the recipient checks that the MAC is correct.

- Problem: SKESs are not required to ensure integrity.
    - For example, changing the ciphertext might not change the plaintext for certain values of plaintext. Violates integrity (of ciphertexts).
    - One can often then also learn information about the plaintext.
- Even if the SKES is secure and the MAC is secure, the MAC-then-encrypt combination might be insecure.

An extra complication: padding

- Often, when using block ciphers, a message needs to be padded before encrypting, to align it with block boundaries.
- Example — PKCS #7:



  where `NN` is the number of bytes of padding (in hexadecimal).
- Should the padding be included in the MAC input?

Padding example

Suppose we are designing SSL/TLS using MAC-then-encrypt.

Option 1. Apply MAC, then pad, then encrypt:

$$\text{compute } t = \mathrm{MAC}(m) \text{ and } c = \mathrm{Encrypt}(m\|p\|t), \text{ transmit } c$$

Option 2. Apply padding, then MAC, then encrypt:

$$\text{compute } t = \mathrm{MAC}(m\|p) \text{ and } c = \mathrm{Encrypt}(m\|p\|t), \text{ transmit } c$$

SSL/TLS uses Option 1. Unfortunately, this is the wrong choice.

Padding oracle attacks

- SSL/TLS:
    - BEAST (2011)
    - POODLE (2014)
- SSH:
    - Plaintext recovery attack (2008)

Encrypt then MAC (EtM)

The good news: encrypt-then-MAC is safe!

Theorem (Canetti & Krawczyk, 2001)

Suppose the SKES is semantically secure under chosen plaintext attack, and the MAC is existentially unforgeable under chosen message attack. Then encrypt-then-MAC is semantically secure and provides integrity (specifically "ciphertext unforgeability under chosen plaintext attack").

Ferguson & Schneier, 1999:

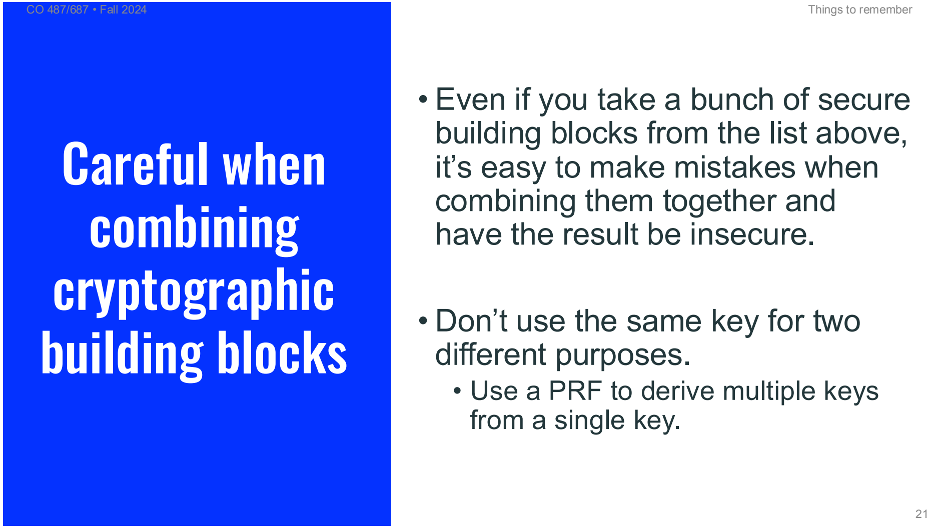
> *When both encryption and authentication are provided, IPsec performs the encryption first, followed by the authentication. In our opinion, this is the wrong order. Going by the "Horton principle", the protocol should authenticate what was meant, not what was said. Authentication should thus be applied to the plaintext, and not to the ciphertext.*

WRONG!

The Cryptographic Doom Principle (Moxie Marlinspike, 2011):

> *If you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will* somehow *inevitably lead to doom.*

4+8+7 things to remember from CO 487



CO 487/687 • Fall 2024

Things to remember

Careful when combining cryptographic building blocks

- Even if you take a bunch of secure building blocks from the list above, it’s easy to make mistakes when combining them together and have the result be insecure.
- Don’t use the same key for two different purposes.
  - Use a PRF to derive multiple keys from a single key.

21

Outline

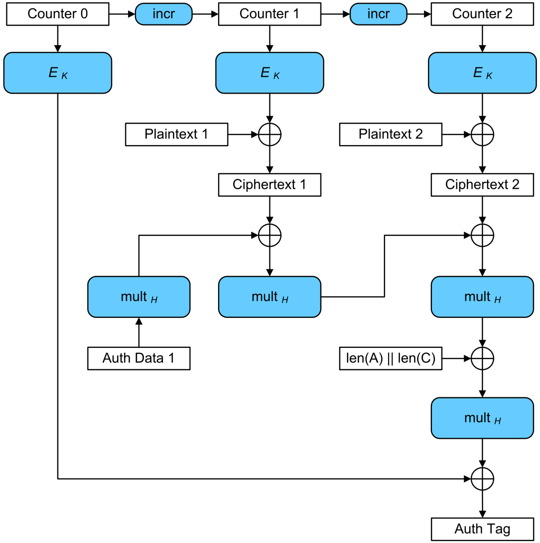
First-class primitives for authenticated encryption

The fastest way to achieve authenticated encryption is to use a block cipher mode of operation with authentication built-in:

- CCM mode (Counter with CBC-MAC)
- EAX mode
- GCM mode (Galois/Counter Mode)
- OCB mode (Offset Codebook Mode) [patented!]

These modes of operation require a block cipher, but not a separate MAC (authentication functionality is built-in).

Example: Galois Counter Mode



GCM mode of operation

- GCM ciphertexts (ignoring the authentication tag) are **identical** to counter (CTR) mode ciphertexts.
    - In particular, the last ciphertext block is truncated if the plaintext length is not an integral number of blocks.
- Authentication tags are computed in

$$\mathrm{GF}(2^{128}) = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1).$$

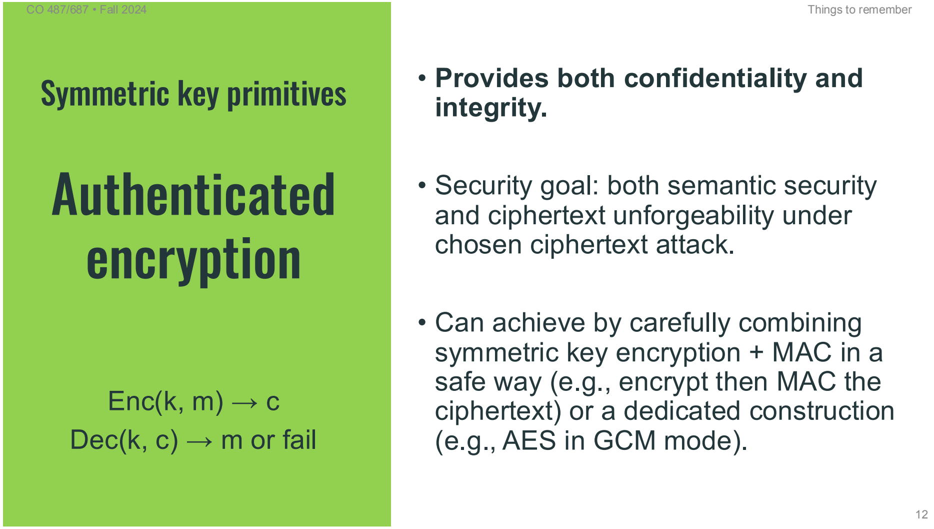
- Hence, GCM requires a 128-bit block size (e.g. AES).
- "Auth Data 1" is a 128-bit block of authenticated unencrypted data, viewed as an element of $\mathrm{GF}(2^{128})$.
    - More than one such block is supported, but only one is shown.
- H is defined as $H = E_k(0^{128}) = E_k(\mathbf{0}) \in \mathrm{GF}(2^{128})$. Computing H requires knowledge of the key.
- Computing authentication tags can be parallelized using field arithmetic.

Some Features of AES-GCM

1. Performs authentication and encryption.
2. Authentication is significantly faster than encryption.
3. Supports authentication only (by using empty M).
4. Very fast implementations on Intel and AMD processors because of special AES-NI and PCLMULQDQ instructions for the AES and $\bullet$ operations.
5. Encryption and decryption can be parallelized.
6. AES-GCM can be used in streaming mode.
7. Security is justified by a security proof:
   - Original McGrew-Viega security proof (2004) was wrong.
   - The proof was fixed in 2012 by Iwata-Ohashi-Minematsu.

AES-GCM is widely used today.

4+8+7 things to remember from CO 487



CO 487/687 • Fall 2024

Things to remember

Symmetric key primitives

Authenticated encryption

Enc(k, m) → c

Dec(k, c) → m or fail

- **Provides both confidentiality and integrity.**
- Security goal: both semantic security and ciphertext unforgeability under chosen ciphertext attack.
- Can achieve by carefully combining symmetric key encryption + MAC in a safe way (e.g., encrypt then MAC the ciphertext) or a dedicated construction (e.g., AES in GCM mode).

12