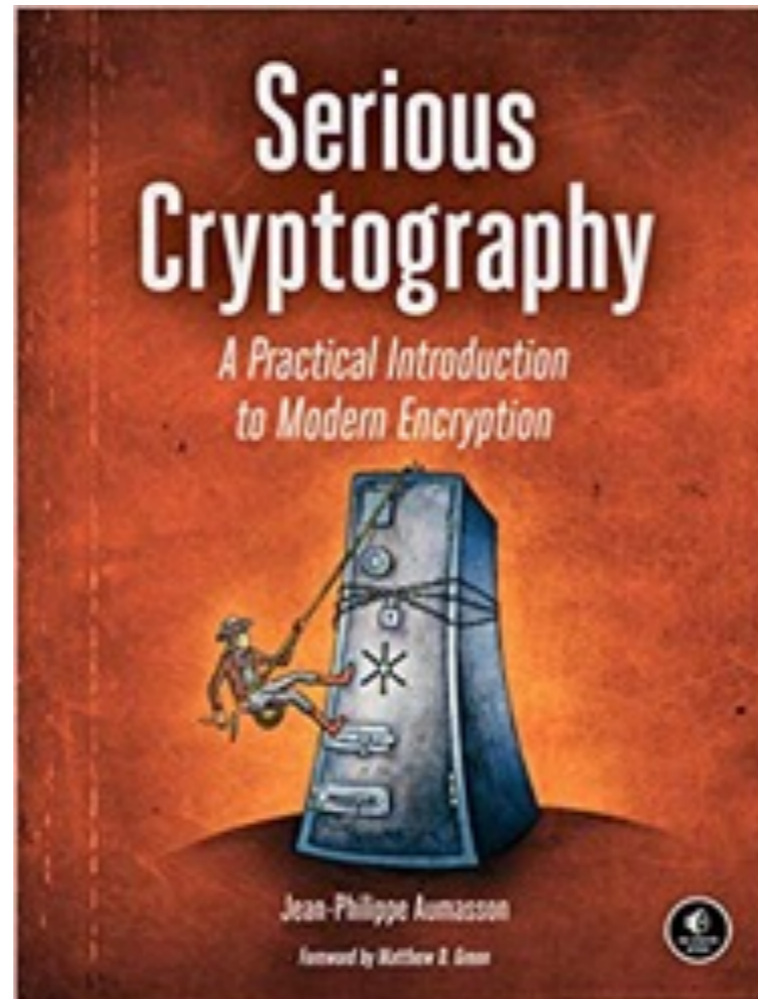


# CNIT 141

## Cryptography for Computer Networks



## 8. Authenticated Encryption

Updated 10-19-22

# MAC v. AE

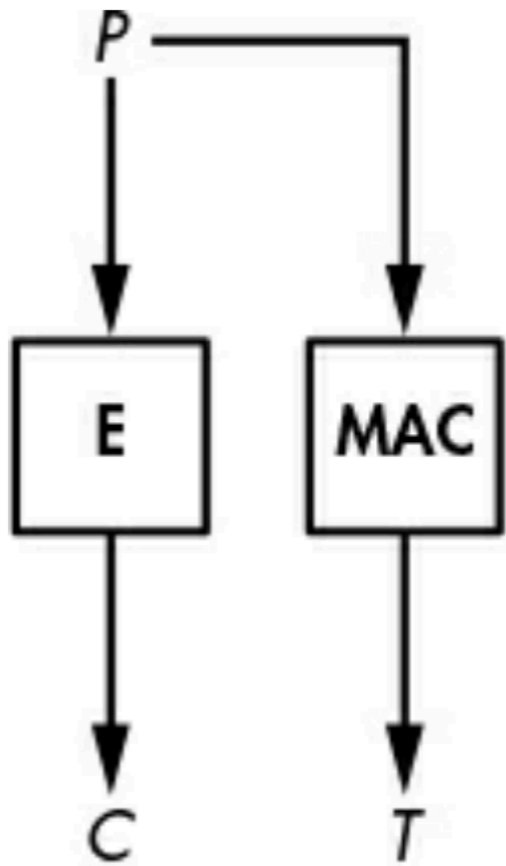
- Message Authentication Code (MAC) from Ch 7
  - Protects a message's **authenticity**
  - With a tag, like a signature
  - But doesn't provide **confidentiality**
- Authenticated Encryption (AE)
  - Produces an **authentication tag**
  - And also encrypts the message
  - Combining a cipher and a MAC

# Topics

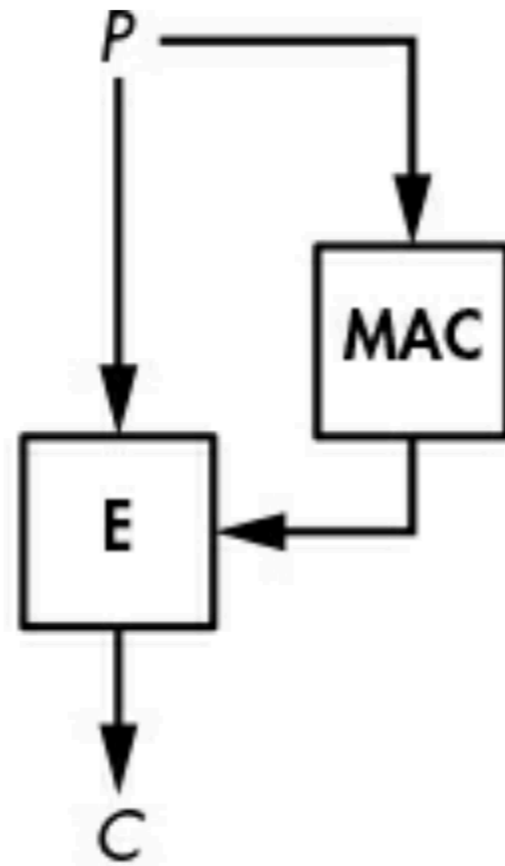
- Authenticated Encryption using MACs
- Authenticated Ciphers
- AES-GCM: The Authenticated Cipher Standard
- OCB: An Authenticated Cipher Faster than GCM
- SIV: The Safest Authenticated Cipher?
- Permutation-Based AEAD
- How Things Can Go Wrong

# Authenticated Encryption using MACs

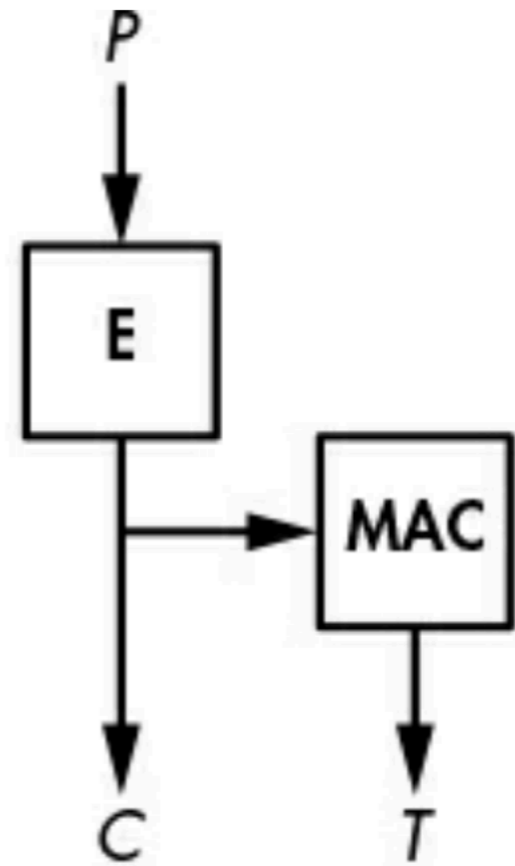
# Three Ways



Encrypt-and-MAC



MAC-then-encrypt



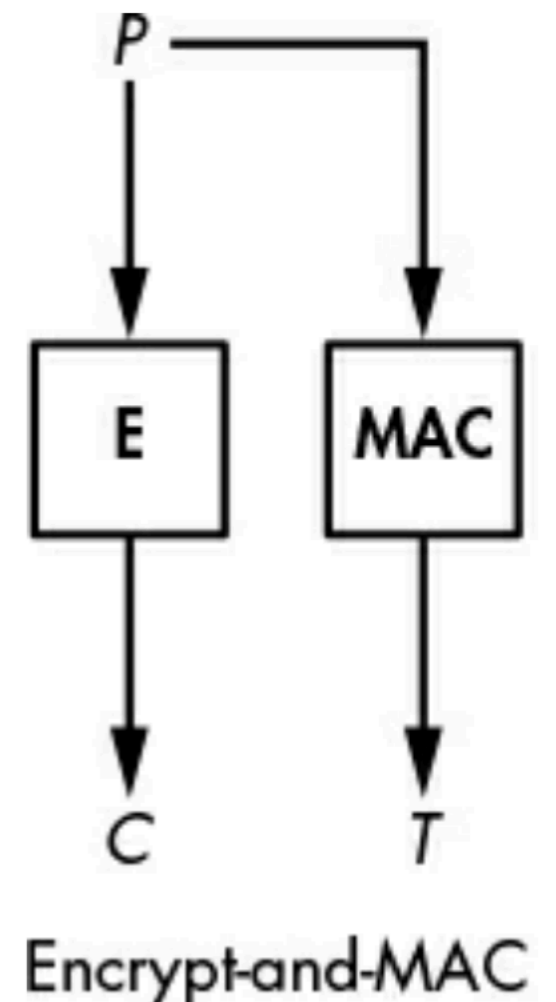
Encrypt-then-MAC

Figure 8-1: Cipher and MAC combinations

E and MAC use different keys

# Encrypt-and-MAC

- Least secure system, in theory
- MAC might leak information about  $P$ 
  - Because MACs are only required to be unforgeable, not random
- Used by SSH



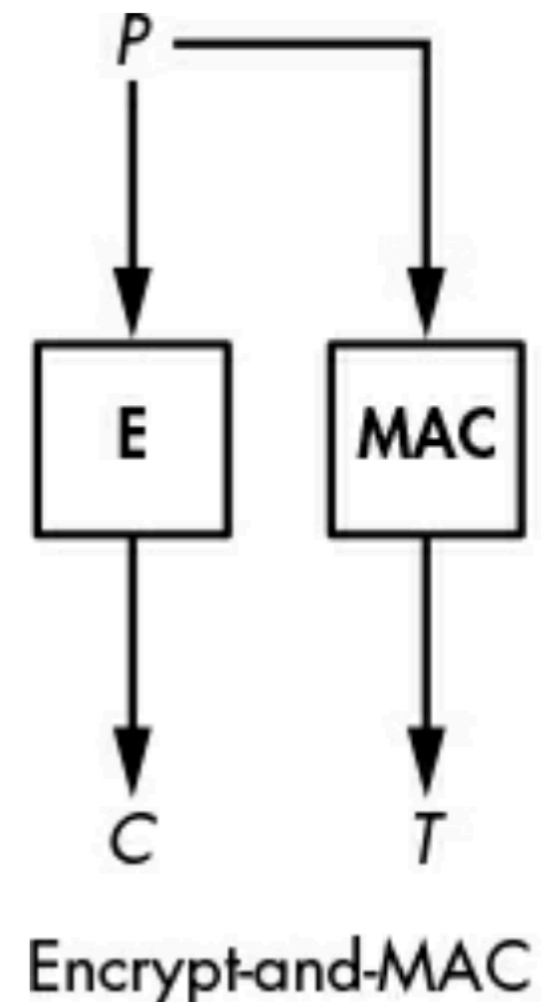
# SSH

- Each encrypted packet **C**
- Is followed by the tag

$$T = \text{MAC}(K, N \parallel P)$$

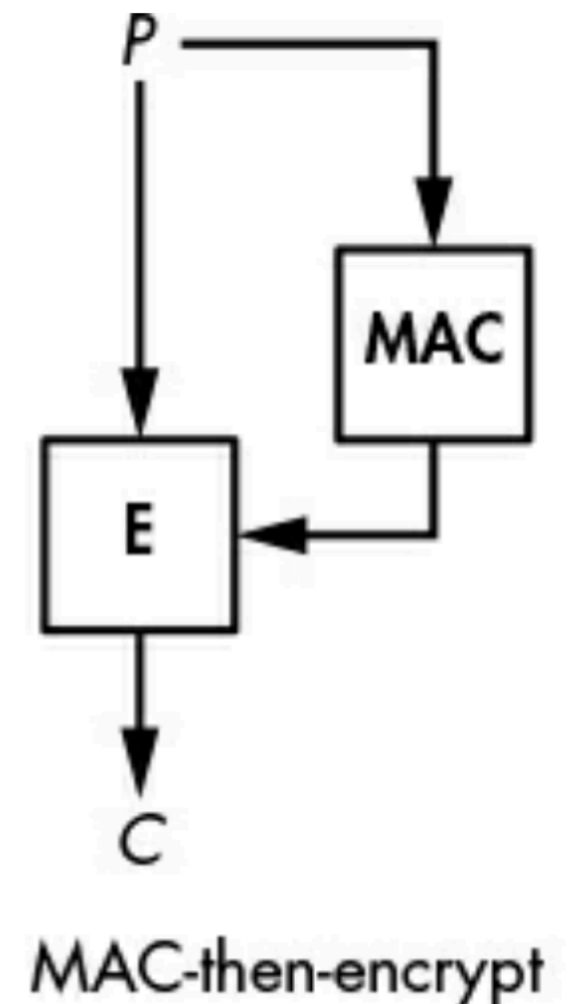
**N** is a sequence number

- Secure in practice because the MAC algorithms actually used are strong and don't leak information about **P**
- Like HMAC-SHA-256



# Mac-then-Encrypt

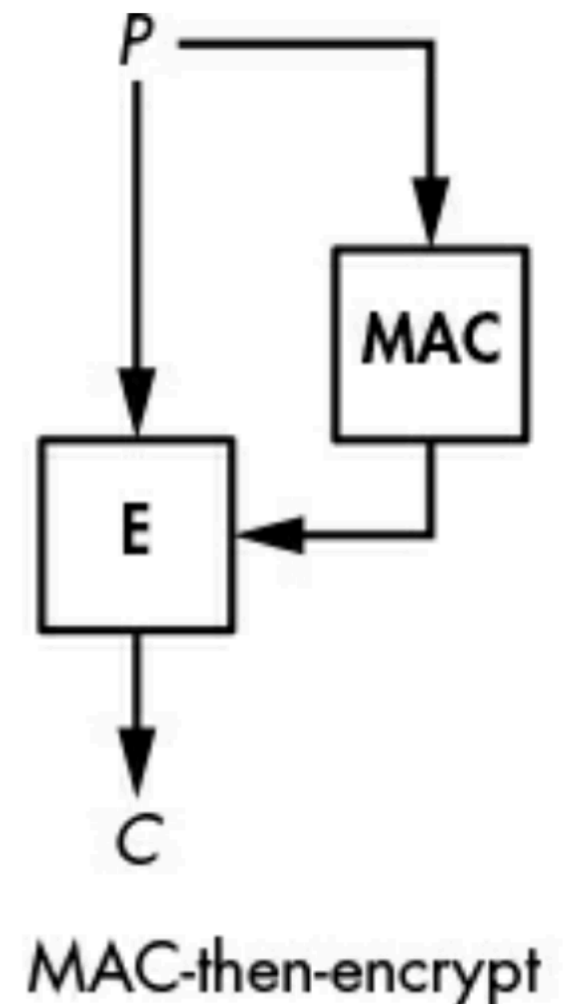
- First compute the tag
  - $T = \text{MAC}(K_2, P)$
- Then create ciphertext
  - $C = \text{E}(K_1, P \parallel T)$
- More secure than encrypt-and-MAC
  - Hides plaintext tag, so it can't leak information about the plaintext





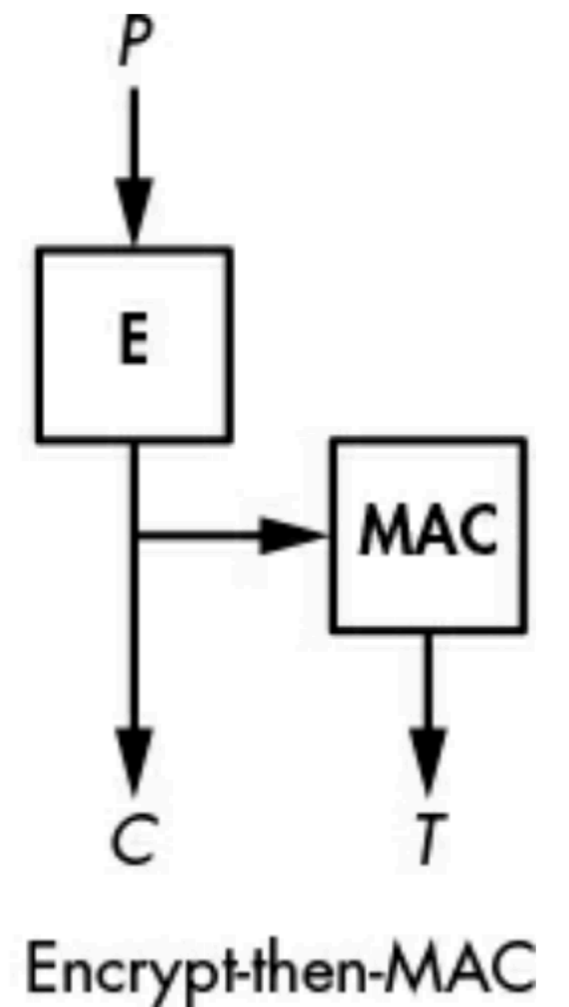
# Mac-then-Encrypt

- Disadvantage
  - Recipient must decrypt  $C$  before checking the MAC
  - Might expose the recipient to corrupted data
- Used by TLS until version 1.3
  - TLS 1.3 uses **authenticated ciphers** (next section)



# Encrypt-then-MAC

- Sends two values
  - Ciphertext  $C = E(K_1, P)$
  - Tag  $T = \text{MAC}(K_2, C)$
- Strongest system
- Used by IPSec



# Authenticated Ciphers

# What is an Authenticated Cipher?

- Different from combining cipher and MAC
- Like a normal cipher, but
- Returns an authentication tag along with ciphertext
  - $AE(K, P) = (C, T)$
- Decryption
  - $AD(K, C, T) = P$

# Security Requirements

- Authentication should be as strong as a MAC
  - Impossible to forge a  $(C, T)$  pair
  - That the **AD** function will accept and decrypt
- Confidentiality is stronger than a basic cipher
  - Decryption will fail unless the tag is valid
  - Prevents chosen-ciphertext attacks
    - Sending ciphertexts and asking for plaintext

# Authenticated Encryption with Associated Data (AEAD)

- **Associated data** is authenticated by the tag
  - But not encrypted
- Example: network header data
  - $\text{AEAD}(K, P, A) = (C, A, T)$
- Tag depends on both  $P$  and  $A$
- Will be rejected if  $C$  or  $A$  is modified

# Authenticated Encryption with Associated Data (AEAD)

- $\text{AEAD}(K, P, A) = (C, A, T)$
- You can leave  $A$  or  $P$  empty
- If  $A$  is empty, AEAD is AE
- If  $P$  is empty, AEAD is MAC

# Avoiding Predictability with Nonces

- Identical plaintext would have identical ciphertext
  - *Predictable* like ECB mode
- Use  $AE(K, P, A, N)$  to eliminate predictability
  - $N$  is a nonce and must not be re-used with the same key



What Makes a Good  
Authenticated Cipher?

# Security Criteria

- Confidentiality must be as strong as the best cipher
- Authenticity must be as strong as the best MAC
- Fragility if nonce is repeated (***misuse resistance***)

# Performance Criteria

- Parallelizability -- can process multiple blocks simultaneously
  - CTR is parallelizable, CBC is not

# Performance Criteria

- Internal structure
  - Two-layer, like AES-CGM,
    - First layer is encryption of plaintext
    - Second layer is authentication
  - One-layer may be simpler and faster

# Performance Criteria

- ***Streamable*** (aka ***online***)
  - Can process a message block by block
    - Discarding completed blocks
    - Reduces RAM requirements
- Nonstreamable ciphers must store the entire message
  - Typically because they must make two passes over the data

# Functional Criteria

- Other features of implementation
- Some AC only allow *A* to precede *P*
  - Others place it after, or anywhere
- Some systems, like AES-CBC, require two algorithms, one for encryption and one for decryption
  - AES-CTR can use the same algorithm for both
  - Cost may matter on low-cost dedicated hardware

**AES-GCM:**

**The Authenticated Cipher Standard**

# AES-GCM

- AES algorithm
- Galois Counter Mode (GCM)
  - Pronounced *gal-waa*
  - A tweak of CTR mode
  - Uses a small, efficient algorithm to compute an authentication tag



# AES-GCM

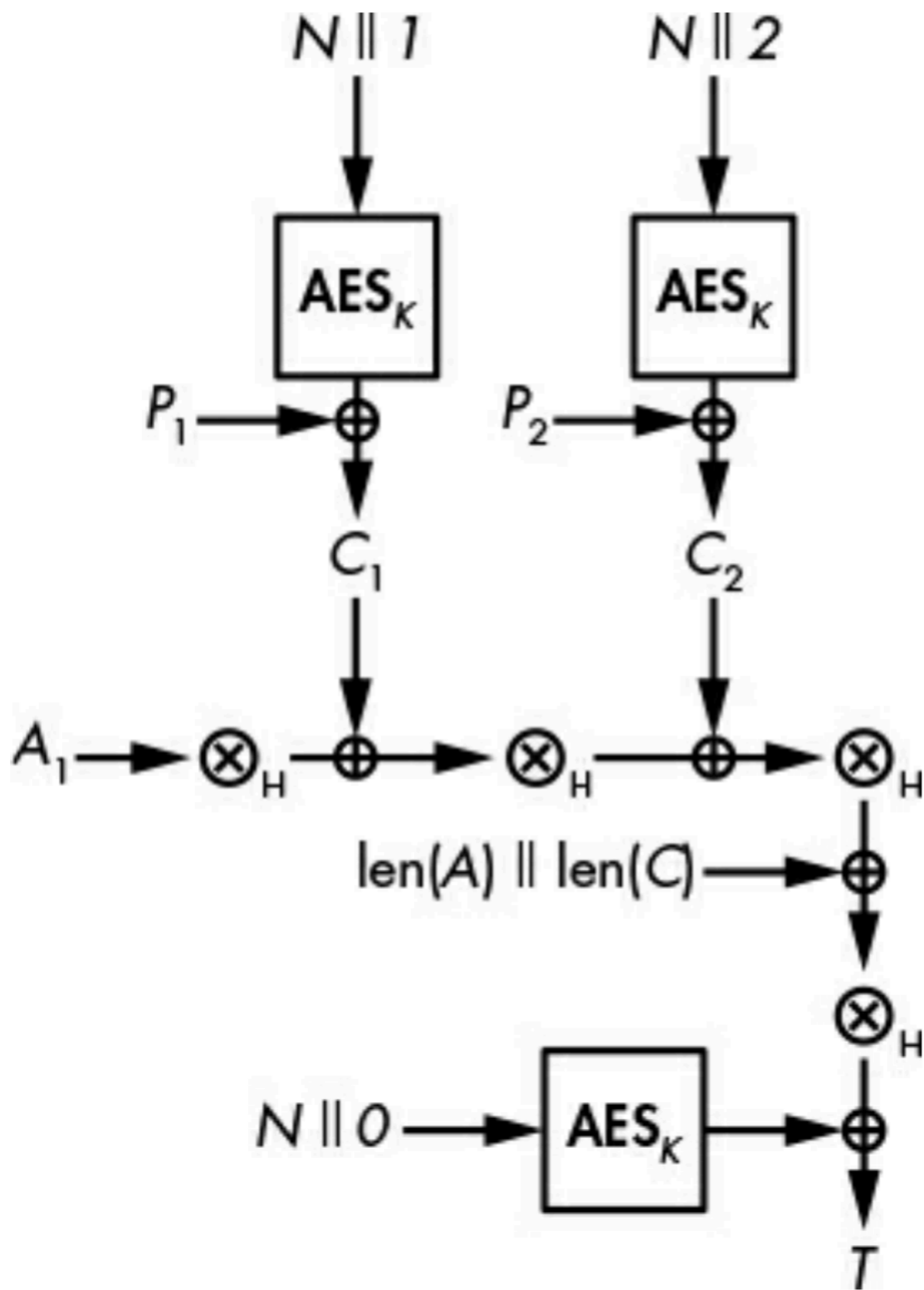
- The most widely used authenticated cipher
- The only authenticated cipher that is a NIST standard (SP 800-38D)
  - Part of NSA's Suite B
  - Used in IPSec, SSH, and TLS 1.2

# AES-GCM

- A tag  $T$  is computed as:

$$T = \mathbf{GHASH}(H, A, C) \oplus \mathbf{AES}(K, N \parallel 0)$$

- GHASH is a universal hash function
  - Linearly related inputs and outputs



- Encrypt-then-MAC
- $K$  is 128-bit key
- $N$  is 96-bit nonce
- Uses "polynomial multiplication" with hash key  $H$

Figure 8-2: The AES-GCM mode, applied to one associated data block,  $A_1$ , and two plaintext blocks,  $P_1$  and  $P_2$ . The circled multiplication sign represents polynomial multiplication by  $H$ , the authentication key derived from  $K$ .

## POLYNOMIAL MULTIPLICATION

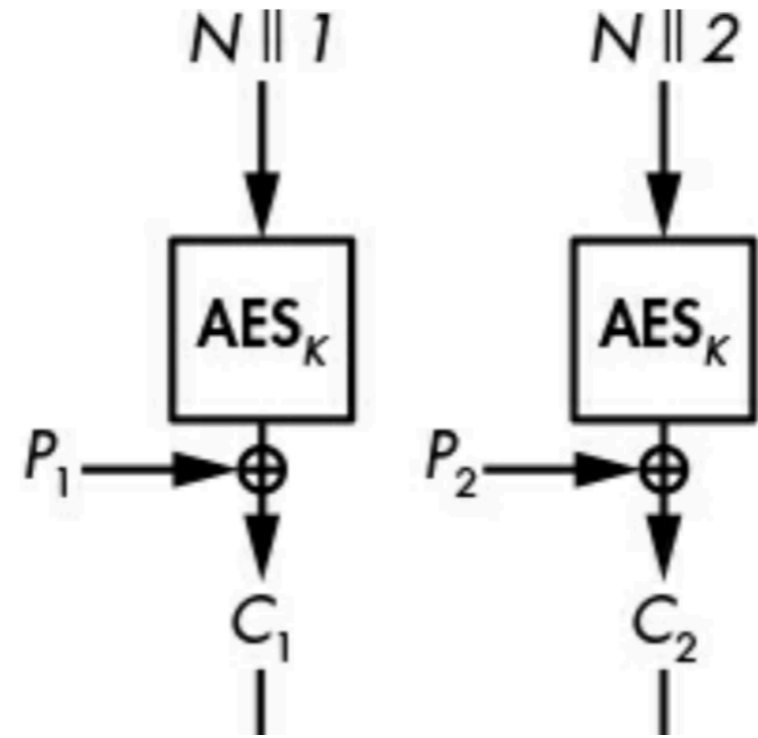
While clearly more complicated for us than classic integer arithmetic, polynomial multiplication is simpler for computers because there are no carries. For example, say we want to compute the product of the polynomials  $(1 + X + X^2)$  and  $(X + X^3)$ . We first multiply the two polynomials  $(1 + X + X^2)$  and  $(X + X^3)$  as though we were doing normal polynomial multiplication, thus giving us the following (the two terms  $X^3$  cancel each other out):

$$(1 + X + X^2) \oplus (X + X^3) = X + X^3 + X^2 + X^4 + X^3 + X^5 = X + X^2 + X^4 + X^5$$

We now apply modulo reduction, reducing  $X + X^2 + X^4 + X^5$  modulo  $1 + X^3 + X^4$  to give us  $X^2$ , because  $X + X^2 + X^4 + X^5$  can be written as  $X + X^2 + X^4 + X^5 = X \otimes (1 + X^3 + X^4) + X^2$ . In more general terms,  $A + BC$  modulo  $B$  is equal to  $A$ , by definition of modular reduction.

# GCM Security

- Fragile when nonce is repeated
  - The AES part is identical
  - If nonce is used twice, attacker can get authentication key  $H$
  - And forge tags
- In 2016, 184 HTTPS servers were found with repeated nonces
  - Including 23 using all-zeroes



# GCM Efficiency

- GCM encryption and decryption are both parallelizable
  - But the MAC is not
- GCM is streamable
  - Because the layers can be pipelined
  - Processing blocks one by one

# OCB: An Authenticated Cipher Faster than GCM

# Offset Codebook (OCB)

- Developed in 2001
- Faster and simpler than GCM
- Limited by license until 2013
- Blends encryption and authentication into one layer
  - With only one key



# OCB Internals

- Each block of plaintext is encrypted with a block cipher
- With a key and an Offset computed from the key and a nonce that increments for each block

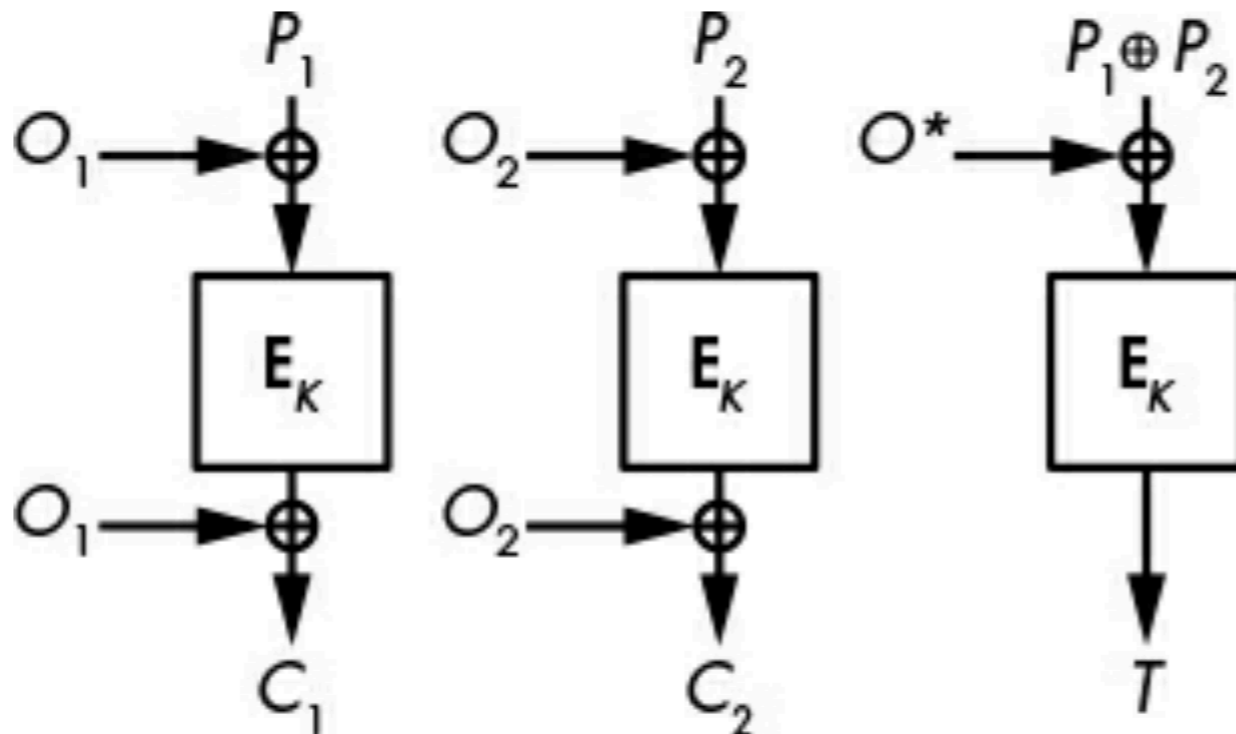


Figure 8-3: The OCB encryption process when run on two plaintext blocks, with no associated data

# OCB Internals

- Tag uses  $S = P_1 \wedge P_2 \wedge P_3$
- XORs  $S$  with an offset computed from the last block's offset

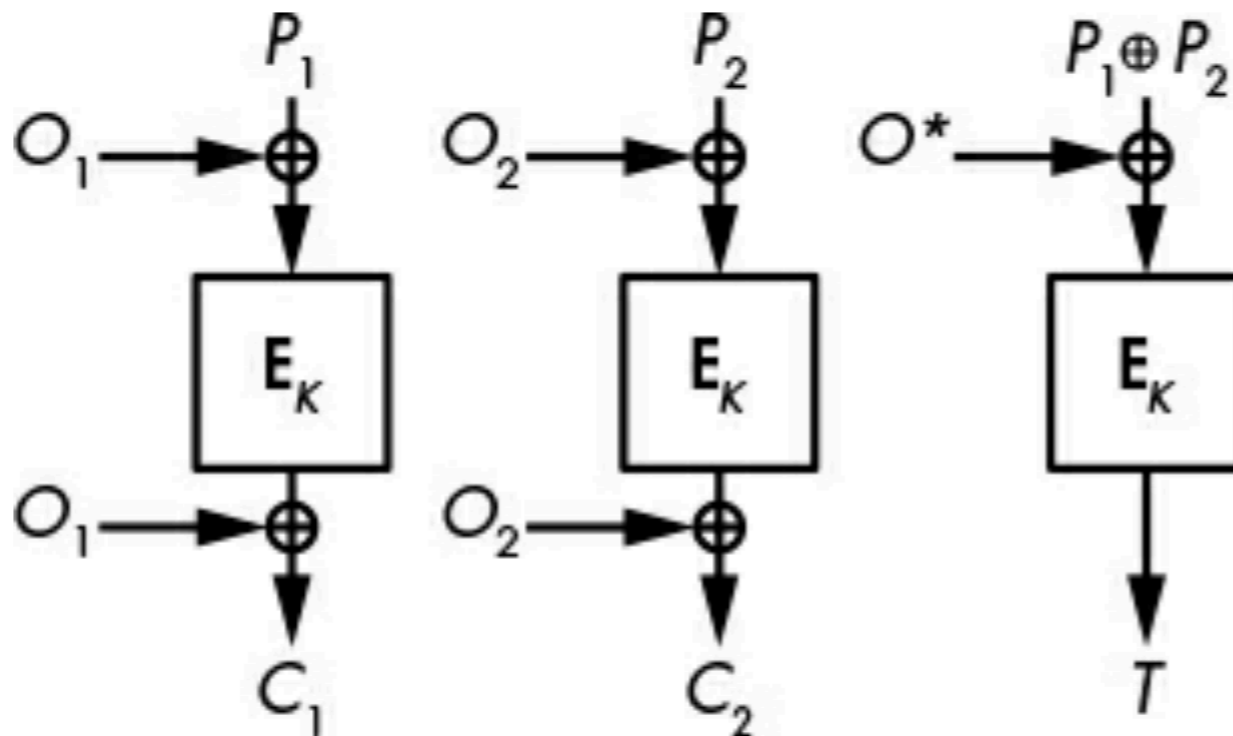


Figure 8-3: The OCB encryption process when run on two plaintext blocks, with no associated data

# OCB

- Support associated data also
- With offset values that are different than those used to encrypt  $P$

$$T = \mathbf{E}(K, S \oplus O^*) \oplus \mathbf{E}(K, A_1 \oplus O_1) \oplus \mathbf{E}(K, A_2 \oplus O_2) \oplus \dots$$

# OCB Security

- Less fragile than GCM against repeated nonces
- Attackers will see identical blocks of ciphertext, like ECB
- But won't be able to find the secret key

# OCB Efficiency

- OCB and GCM are about equally fast
- Both are *parallelizable* and *streamable*
  - On early Intel processors, AES-GCM used to be three times slower than AES-OCB
  - Because the GHASH calculation is slower than the XORs used by OCB
- GCM uses the same function for encryption and decryption
- OCB requires two functions

# SIV: The Safest Authenticated Cipher?

# Synthetic IV (SIV)

- An authenticated cipher mode
- Typically used with AES
- Secure even if you use the same nonce twice
  - Unlike GCM and OCB
  - Attacker will only see a repeat if the same complete plaintext was repeated
  - Not if only the first block is repeated

# SIV Construction

- Combine encryption function **E**
- And a pseudorandom function **PRG**
- Using two keys  $K_1$  and  $K_2$
- And a nonce  $N$ 
  - *Tag:*  $T = \text{PRF}(K_1, N||P)$
  - *Ciphertext:*  $C = E(K_2, T, P)$
- $T$  acts as the nonce of **E**



# SIV Performance

- SIV is not streamable
  - After computing  $T$ , it must keep the entire plaintext  $P$  in memory
  - To encrypt 100 GB of plaintext, you must store 100 GB

# Permutation-Based AEAD

# Permutation

- Not a block cipher like AES
- Simply transforms input to output of the same size
- Reversibly, without using a key
- Fast, secure, and more resistant to nonce reuse than GCM and OCB

# Permutation-Based AEAD

- Start with initial state  $H_0$
- XOR with key  $K$  and nonce  $N$
- Permute with plaintext blocks  $P_1, P_2, \dots$  to get new internal states

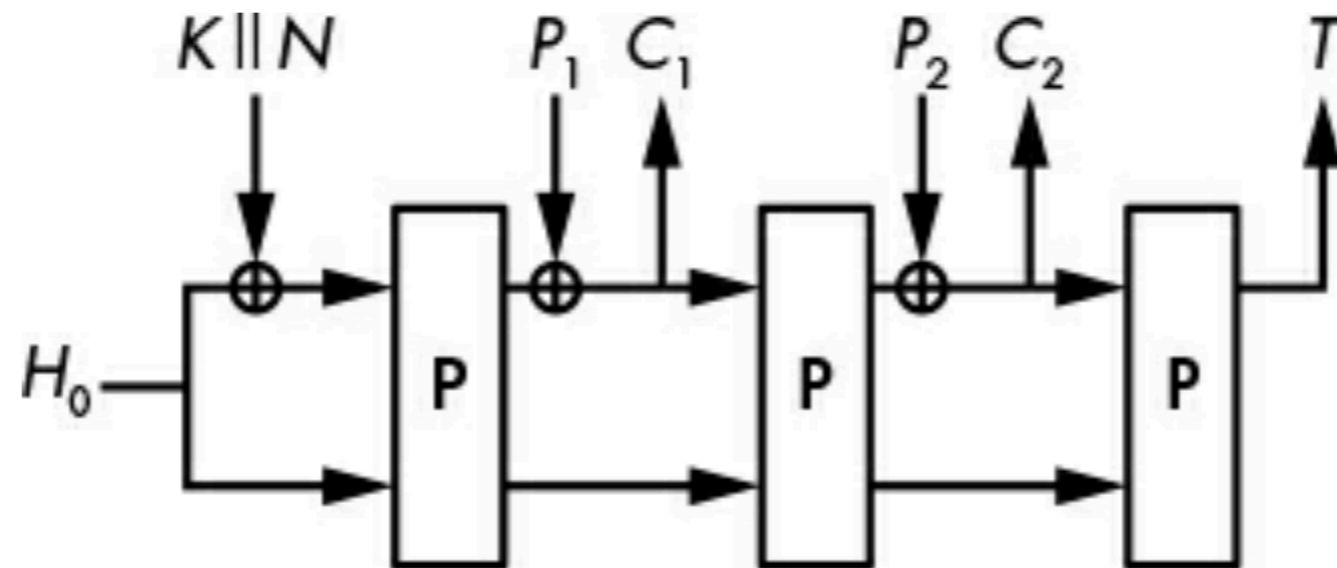


Figure 8-4: Permutation-based authenticated cipher

# Permutation-Based AEAD

- This produces a series of pseudorandom blocks
- XOR them with plaintext blocks to form ciphertext blocks

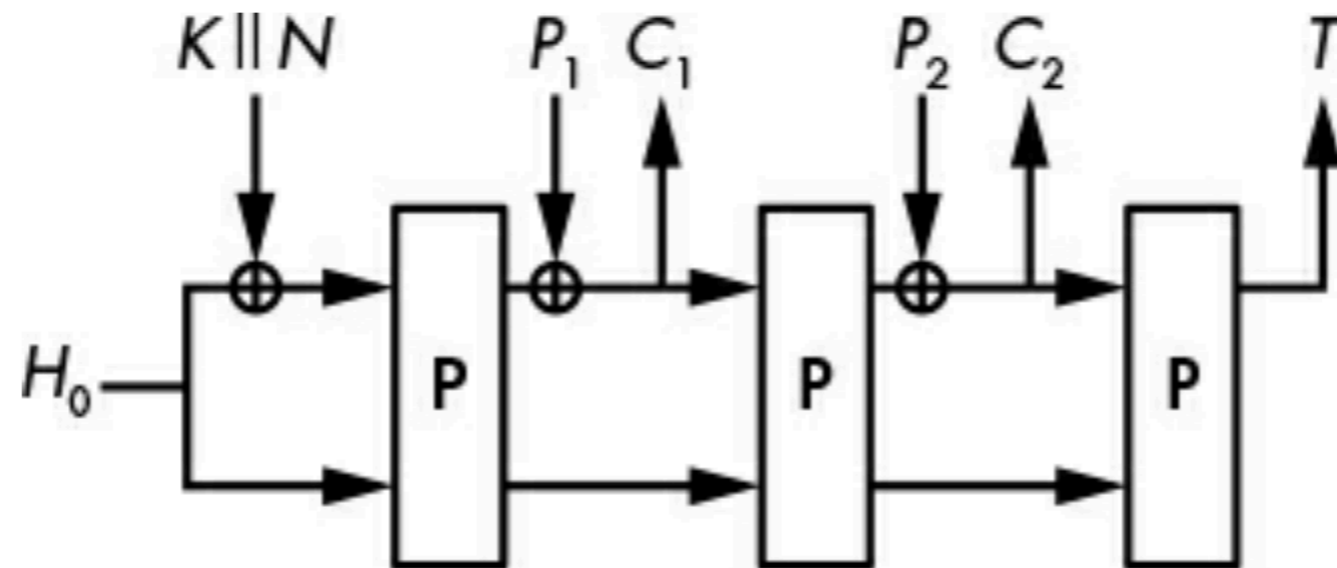


Figure 8-4: Permutation-based authenticated cipher

# Permutation-Based AEAD

- Ciphertext is same length as plaintext
- Internal state is larger than block size
- Bits from last internal state form the tag

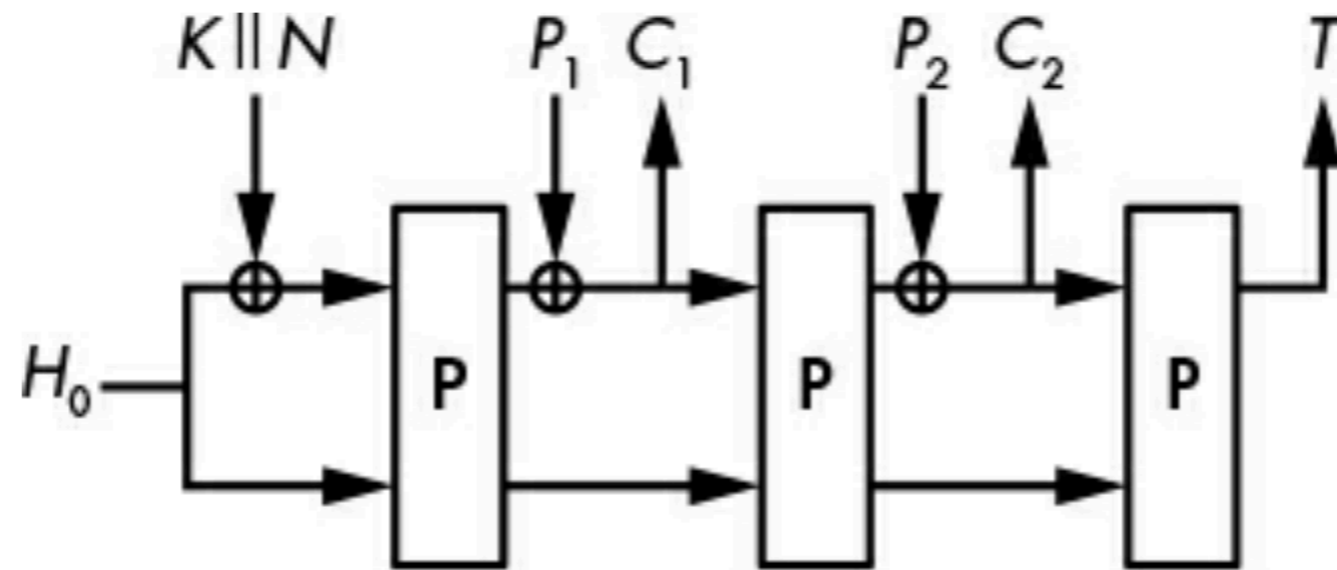


Figure 8-4: Permutation-based authenticated cipher

# Permutation-Based AEAD Security

- Security relies on secrecy of the internal state
- Blocks must be padded carefully
- Nonce re-use is only a small problem
  - Attacker can only tell that messages began with same value

# Permutation-Based AEAD Performance

- A single layer of operations
- ***Streamable***
- A single core algorithm for encryption and decryption
- But not ***parallelizable*** like GCM or OCB
  - New calls to **P** must wait for previous call to complete



# How Things Can Go Wrong

# Attack Surface

- Authenticated ciphers must provide both ***confidentiality*** and ***authenticity***
- Take two values: plaintext ***P*** and associated data ***A***
- Must remain secure for all values of ***P*** and ***A***
  - Even when one is absent, all zeroes, or very large

# Attack Surface

- Must remain secure against attackers who collect numerous message/tag pairs
- And against accidental repetition of nonces

# AES-GCM and Weak Hash Keys

- GHASH uses a hash key  $H$  over and over for each block

$$X_n = H^n \oplus H^{n-1} \oplus H^{n-2} \oplus \dots \oplus H^2 \oplus H$$

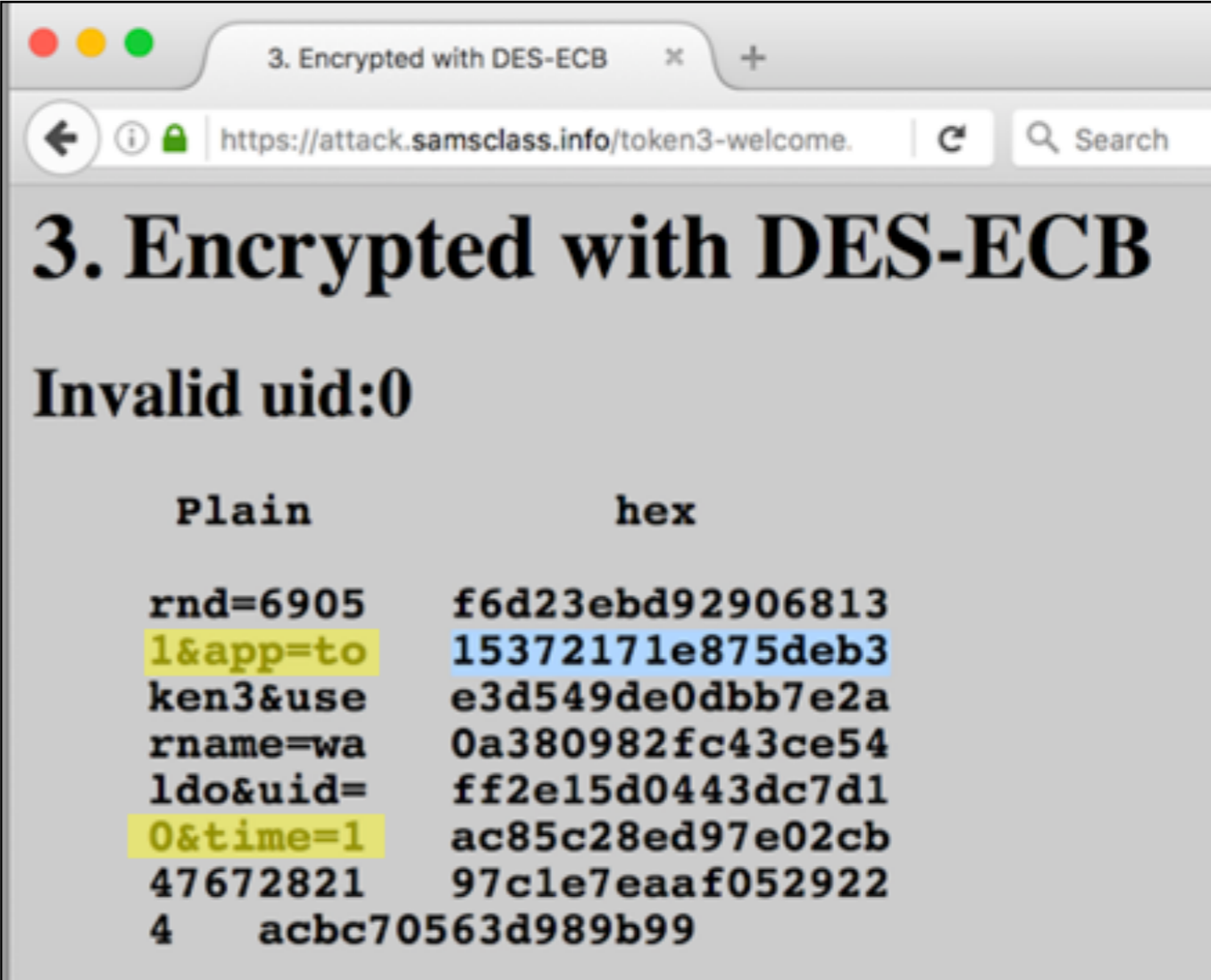
- Certain values of  $H$  make the tag weak
  - Because they form a "short cycle" and repeat every few blocks

# Tag Forgery

- If  $H$  has a cycle of five
  - An attacker could swap the first and sixth block of ciphertext
  - And get the same authentication tag
  - Constructing a new valid message without knowing the key

# Consequences of Tag Forgery

- To become root, swap second and sixth block
- Changes uid to 1



3. Encrypted with DES-ECB

Invalid uid:0

Plain	hex
rnd=6905	f6d23ebd92906813
1&app=token3	15372171e875deb3
ken3&use	e3d549de0dbb7e2a
rname=wa	0a380982fc43ce54
ldo&uid=	ff2e15d0443dc7d1
0&time=1	ac85c28ed97e02cb
47672821	97c1e7eaaaf052922
4	acbc70563d989b99

# AES-GCM and Weak Hash Keys

- Not practical to exploit
  - Attacker needs to know  $H$  or  $K$  to find cycle length
- But still a theoretical defect of AES-GCM

# AES-GCM and Small Tags

- Normally returns 128-bit tags
  - Those are secure
- But can produce shorter tags, like 32 or 48 bits
- Those are much weaker than they appear, for long messages



# AES-GCM and Small Tags

- The probability of a forged tag being accepted is
  - $2^m/2^n$
- Where  $m$  is the number of blocks in the message, and
- $n$  is the number of bits in the tag

# AES-GCM and Small Tags

- 48-bit tags for messages 4 GB long
  - Have one chance in a million of a forgery succeeding

# Kahoot!

**Ch 8**