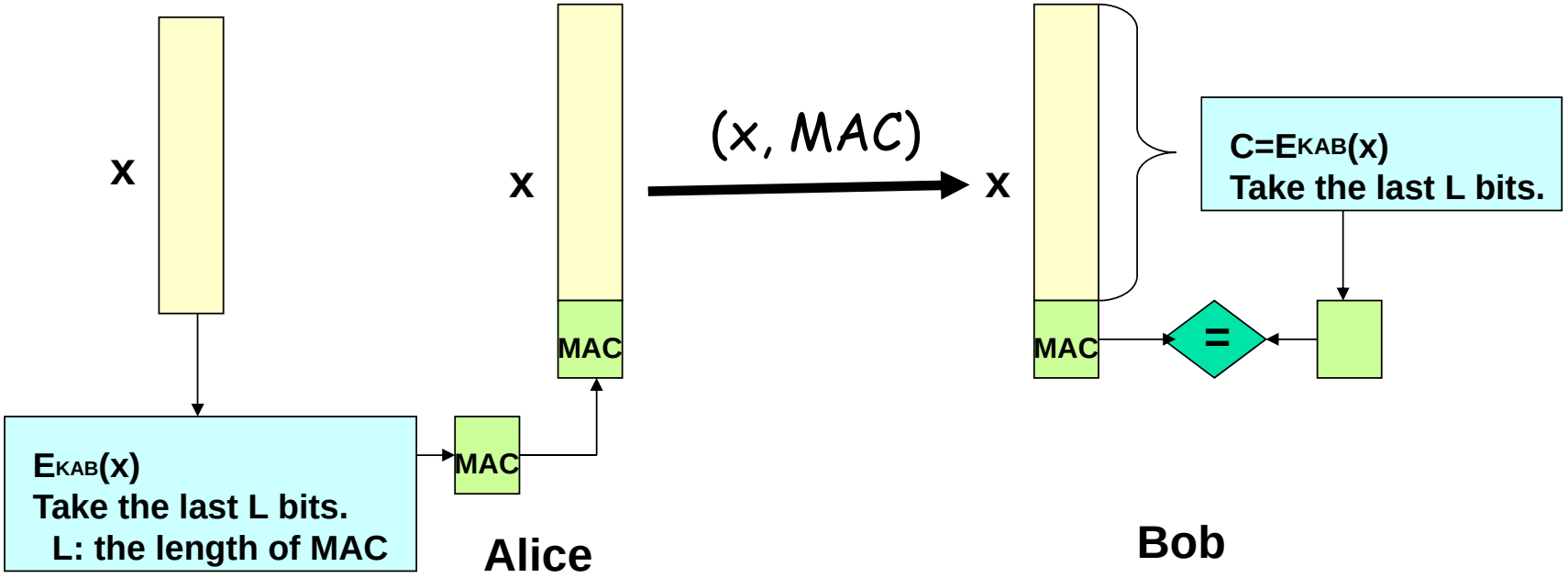
Contents

- Introduction
- Symmetric-key cryptography
  - Block ciphers
  - Symmetric-key algorithms
  - Cipher block modes
  - Stream cipher
- Public-key cryptography
  - RSA
  - Diffie-Hellman
  - ECC
  - Digital signature
  - Public key Infrastructure
- Cryptographic hash function
  - Attack complexity
  - Hash Function algorithm
- Integrity and Authentication
  - Message authentication code
  - Authentication encryption
  - Digital signature
- Key establishment
  - server-based
  - Public-key based
  - Key agreement (Diffie-Hellman)

Authenticated Encryption

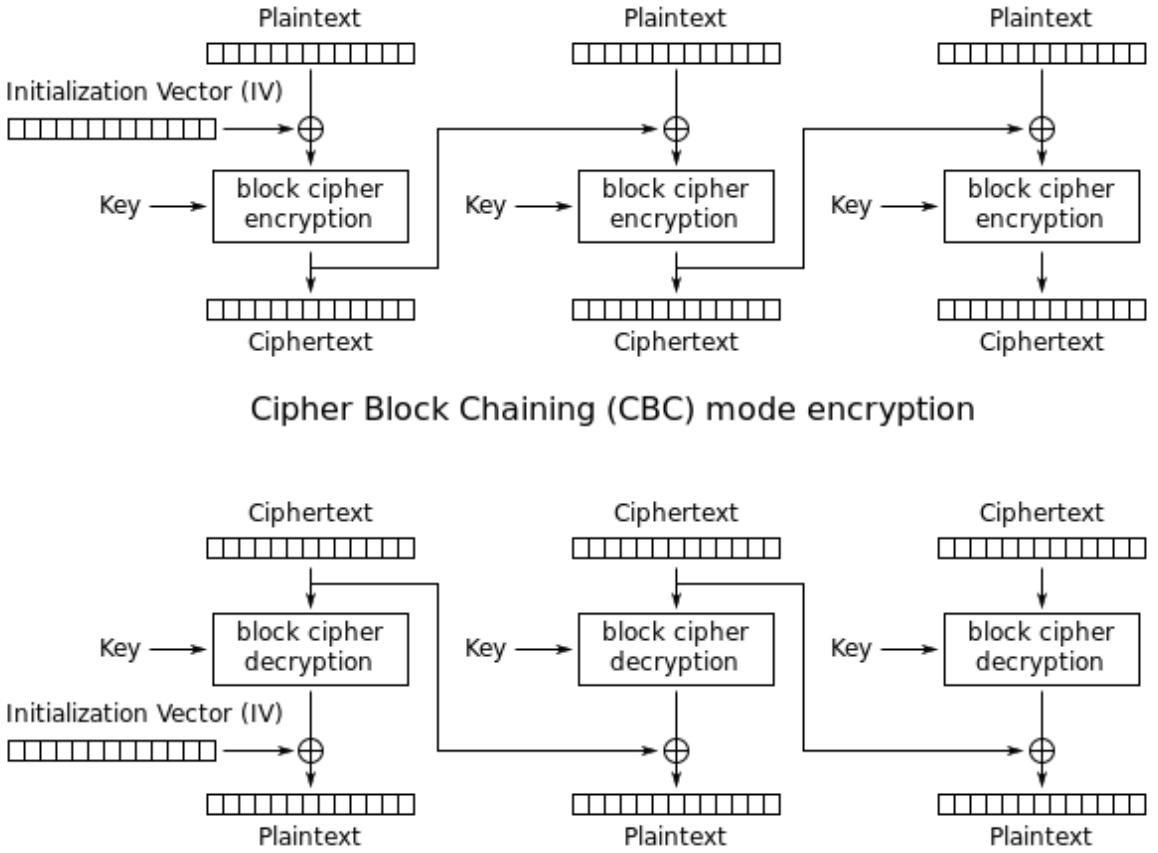
MAC using symmetric key

- In the previous section we saw one possible way of computing MAC using symmetric key.
- But this method generates n MACs for n blocks of a message.



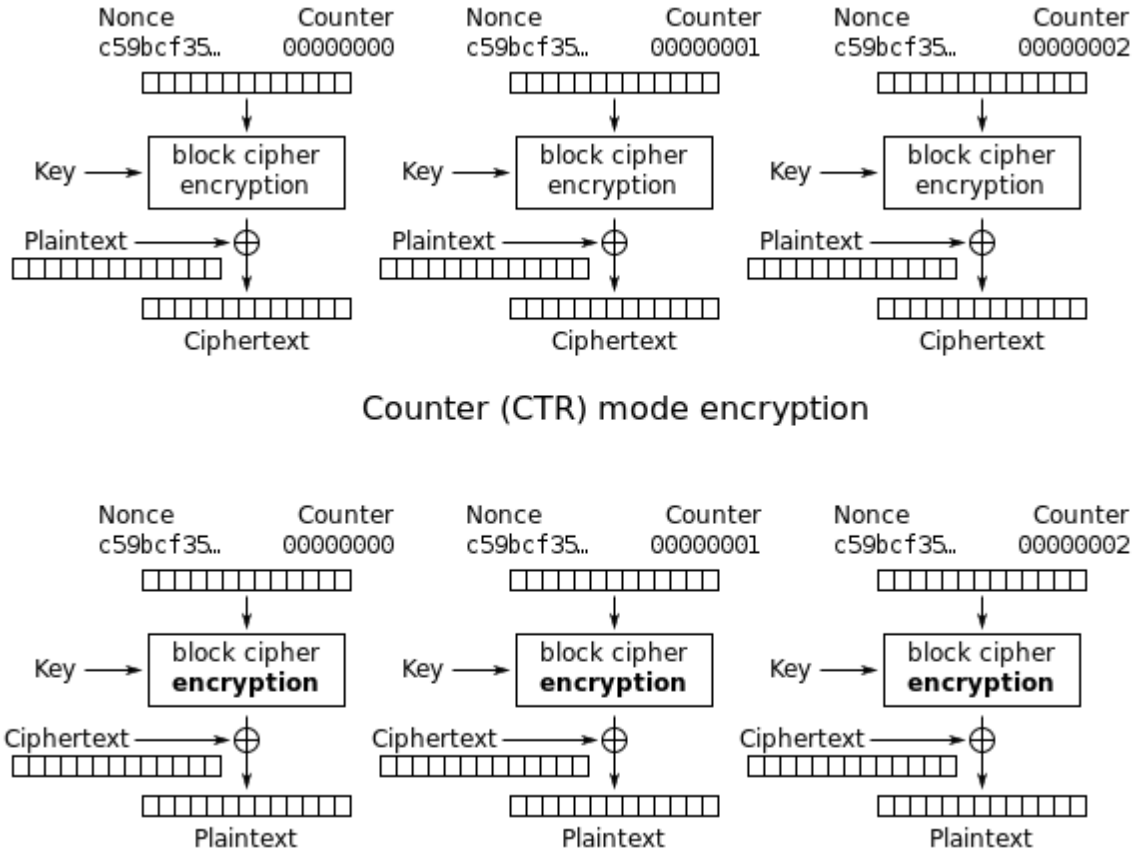
Reminder: Block cipher operation modes

- CBC mode



Reminder: Block cipher operation modes

- CTR mode



MAC using symmetric key block cipher

- We can compute MAC based on the block cipher mode of operation.
- The last result of the block cipher mode computation can be used as MAC.

CBC-MAC Computation

- Assume a message M has N blocks and K is a AES key.

  $M=(M_0, M_1, M_2,\ldots, M_{N-1})$

  $C_0 = E_K(IV \oplus M_0)$,

  $C_1 = E_K(C_0 \oplus M_1)$,

  $C_2 = E_K(C_1 \oplus M_2),\ldots$

  $C_{N-1} = E_K(C_{N-2} \oplus M_{N-1}) = MAC$

- Alice sends plaintext and MAC with IV to Bob.
- Bob does the same computation and verifies that result agrees with MAC
- Note: Bob must know the key K
  - Guarantee message integrity and authentication

Does a CBC-MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $C_0 = E_K(IV \oplus M_0)$, $C_1 = E_K(C_0 \oplus M_1)$,

  $C_2 = E_K(C_1 \oplus M_2)$, $C_3 = E_K(C_2 \oplus M_3) = MAC$
- Alice sends IV, M_0, M_1, M_2, M_3 and **MAC** to Bob
- Suppose Trudy changes M_1 to X
- Bob computes

  $C_0 = E_K(IV \oplus M_0)$, $C_1 = E_K(C_0 \oplus X)$,

  $C_2 = E_K(C_1 \oplus M_2)$, $C_3 = E_K(C_2 \oplus M_3) = MAC \neq MAC$
- That is, error propagates into **MAC**
- Trudy can't make *MAC* == **MAC** without K

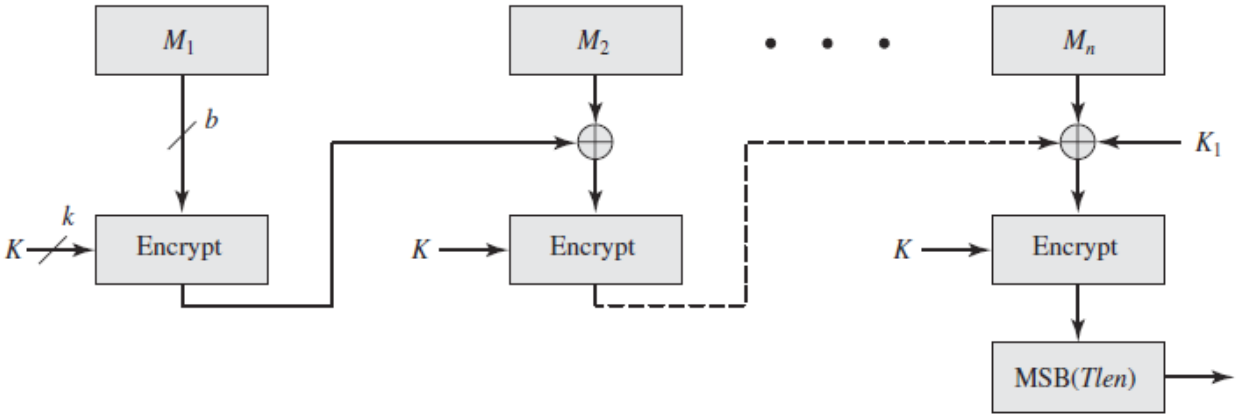
Cipher-based Message Authentication(CMAC)

- CMAC is a refinement of CBC-MAC.
- CMAC uses a k-bit symmetric key K and n-bit constant K_1 or K_2 for n-block message,

$L = E_K(0n)$
$K_1 = L * x$
$K_2 = L * x2 = (L * x) * x$
Where multiplication (*) is done in finite field $GF(2^n)$ and
x and x^2 are 1st and 2nd order polynomials that are elements of $GF(2^n)$.

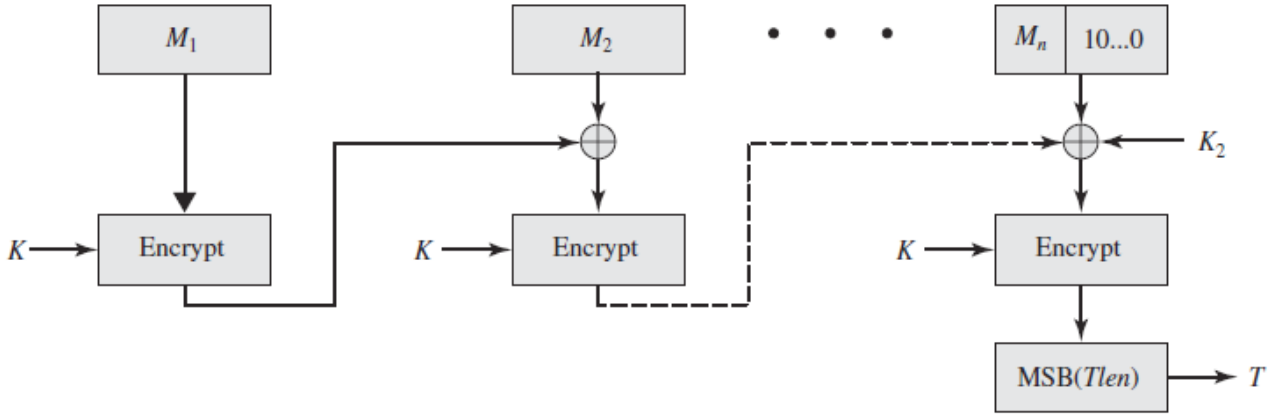
Message length is integer multiple of block size

b: block size
k: key size
T(Tag) : $MSB_{Tlen}(C_n)$
Tlen: bit length of T
$Cn=E_K(M_n \oplus (C_{n-1} \oplus K_1))$



(MAC)

Message length is not integer multiple of block size



Authenticated Encryption

- Motivation:
  - Most applications require confidentiality as well as integrity and authentication simultaneously.
- Common approaches to provide encryption and MAC simultaneously
  - HtE (Hash-then-encryption) : $E_K(M||h(M))$, WEP
  - MtE(MAC-then-encryption) : $C=E_{K2}(M||MAC_{K1}(M))$
    - SSL/TLS
  - EtM(Encrypt-then-MAC) : $C=E_{K2}(M)$, $T=MAC_{K1}(C)$, send (C,T)
    - IPsec
  - E&M(Encrypt-and-MAC) : $C=E_{K2}(M)$, $T=MAC_{K1}(M)$, send (C,T)
    - SSH

Counter with CBC-MAC (CCM)

- NIST standard(NIST SP800-38C to support IEEE 802.11
- Variation of E&M scheme
- Designed for AES-128
- One key for authentication and encryption
- Patent free
- CCM has to do relatively complex computation.
- Not possible parallel computation

CCM authentication

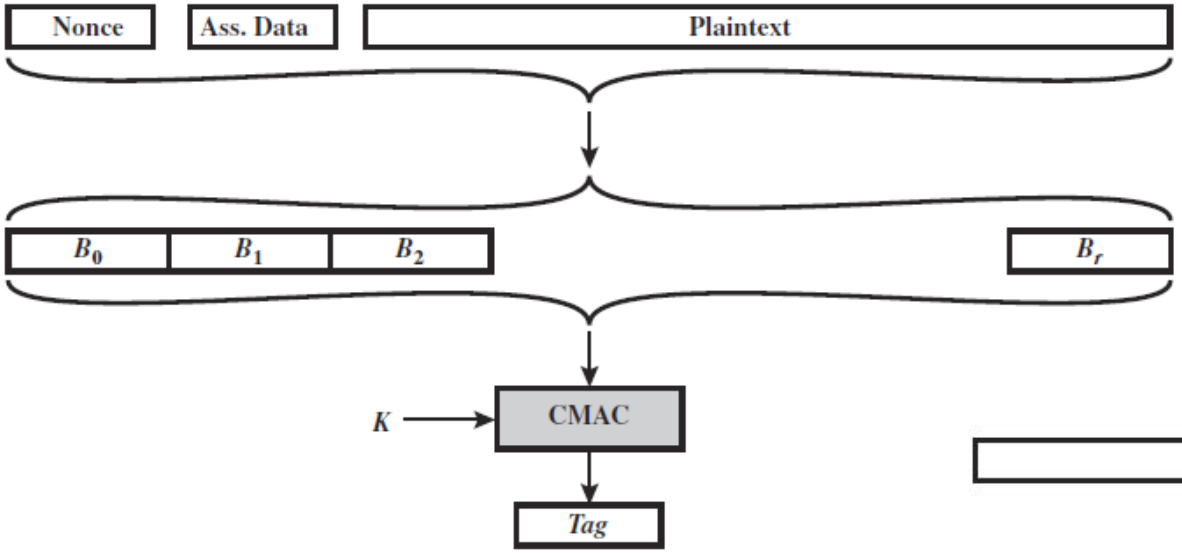
- For authentication
  - Input : r-blocks (B1,B2,…,Br) consist of
    - Nonce; random value that is different for each plaintext
    - Associated data : authenticated but not encrypted, eg., protocol header
    - Plaintext
  - Use CMAC
  - Output: Tag with length less than or equal to the block length

CCM encryption

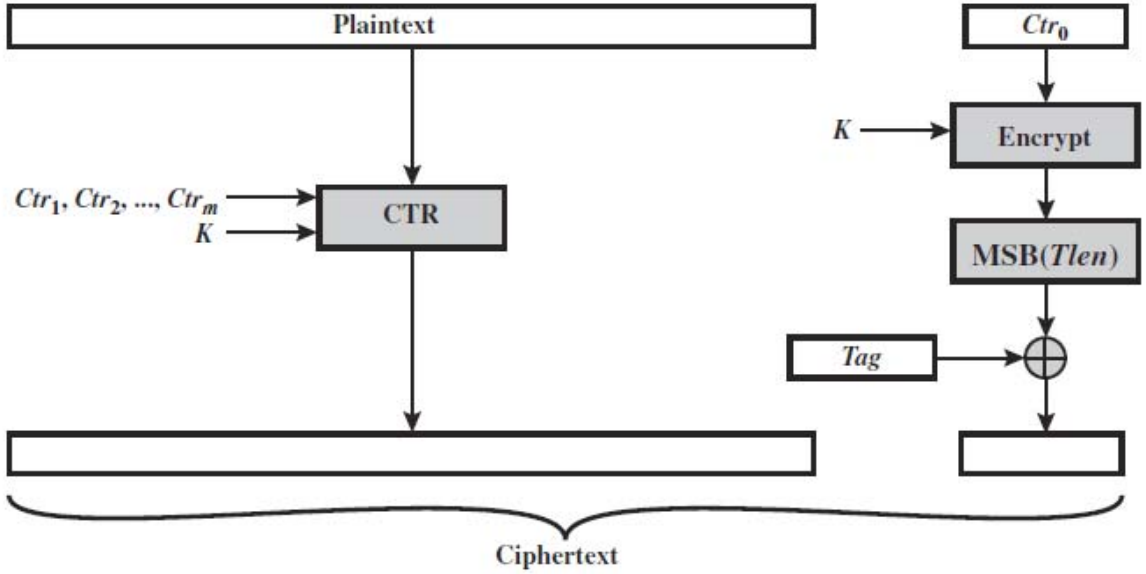
- Input
  - Plaintext with length Plen
  - Generate counter blocks: (ctr_0, ctr_1 , ctr_0 ctr_0)

CCM encryption

- Input
  - Plaintext with length *Plen*
  - Generate counter blocks: (ctr_0, ctr_1, ctr_2,…,ctr_m)
    - where m = $\left\lceil \frac{plaintext\ lenth}{128} \right\rceil$
- Apply encryption algorithm with CTR mode
  - For j=0 to m, $S_j=E(ctr_j)$
  - $S=S_1||S_2||\ldots||S_m$
- output
  - Ciphertext $C=(P \oplus MSB_{plen}(S))$, $MAC=(T \oplus MSB_{Tlen}(S_0))$



Authentication



Encryption

Galois/Counter Mode (GCM)

- Very fast: the speed of 10 Gbits per sec and above in H/W.
- Easy to implement in H/W and S/W
- Allow Pipelined and parallelized implementation
- Can be used for authentication, if desired
- Optimized for AES-128
- Accept initialization vectors of arbitrary length
- Only one key for authentication and encryption
- Free of intellectual property restrictions
- NIST standard (NIST SP 800-38D)

Input: plaintext X with n blocks ($x_1, x_2, \ldots, x_{n-1}, x_n^*$)

x_n^*: last block may be less than 128bits

initial vector (ICV):

initial counter $Cntl_0$ = ICV||padding if ICV < 128

K : encryption key

A: additional authentication data with m blocks ($A_1, A_2, \ldots, A_{m-1}, A_m^*$)

(authenticated but not encrypted)

If the length of A is less than 128, $A=A_0$ in the following figure.

Output: ciphertext $C = (C_1, C_2, \ldots, C_{n-1}, C_n^*)$

authentication tag : $0 < T < 128$

Send: [ICV, A, $(C_1, C_2, \ldots, C_{n-1}, C_n^*)$, Tag]

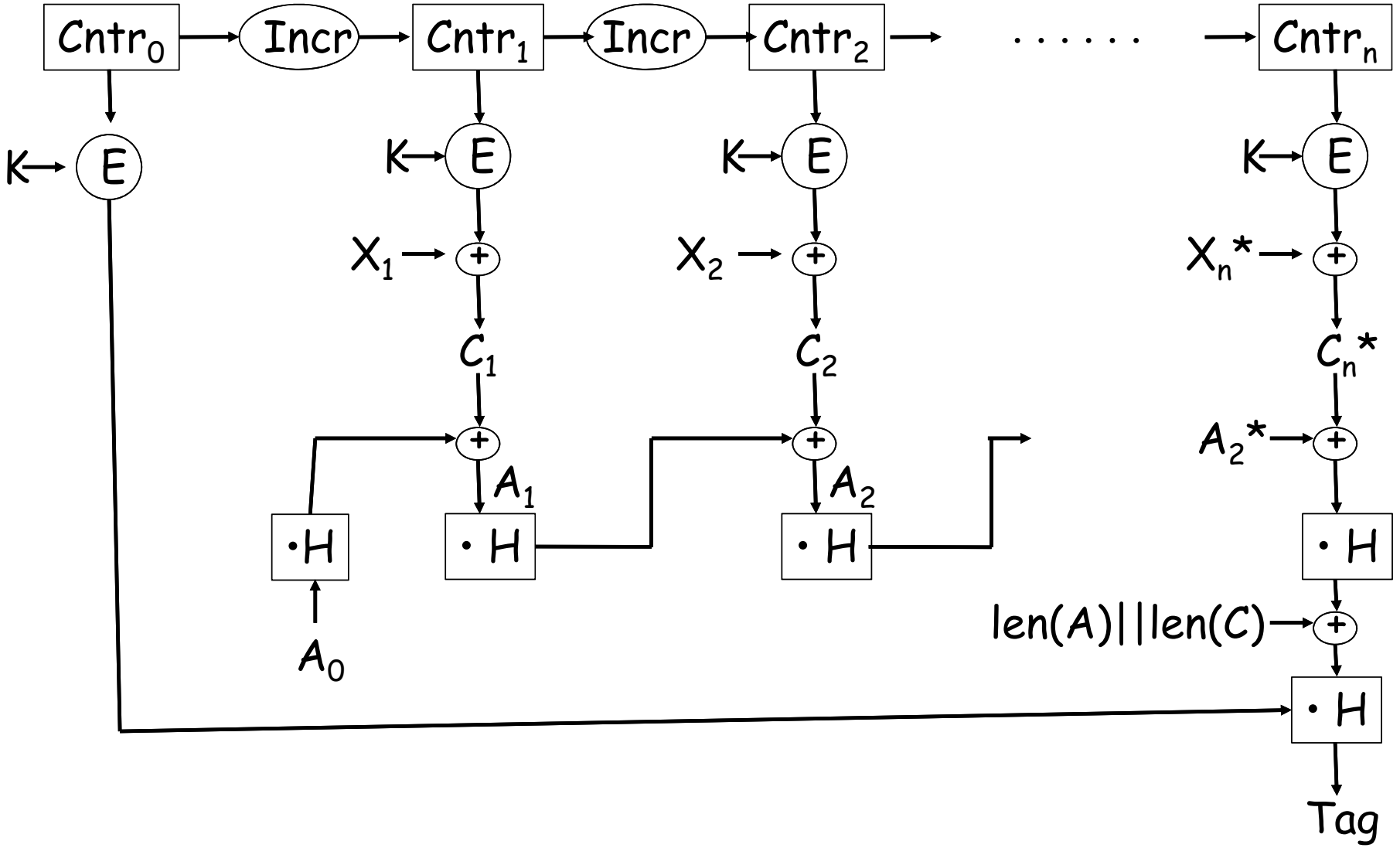
$H = E_K(0^{128})$

Increment function

$Incr_{32}(S)$ = increments the rightmost 32 bits of S
by 1 mod 232, and the remaining bits are unchanged

$A_m = (X_1 \cdot H^m) \oplus (X_2 \cdot H^{m-1}) \oplus \ldots\ldots \oplus (X_{m-1} \cdot H^2) \oplus (X_m \cdot H)$

The value $H_2, H_3, \ldots$ can be precalculated one time for use with each message to be authenticated. Then, the blocks of data ($x_1, x_2, \ldots, x_{n-1}, x_n^*$) can be processed in parallel.


Cntr0
Incr
Cntr1
Incr
Cntr2
. . . . . .
Cntrn
K
E
X1
X2
Xn*
C1
C2
Cn*
A1
A2
A2*
·H
A0
len(A)||len(C)
Tag