홍익대학교
HONGIK UNIVERSITY

Network Security

<CH 3>

Youn Kyu Lee
Hongik University

Security Models: types of cipher

- Random functions – hash functions
  : accepts an input string of any length and outputs a string of fixed length, say *n* bits long

- Random generators – stream ciphers
  : (reverse of hash function?) accepts a short input and produces a long output

- Random permutations – block ciphers
  : keyed-invertible function transfers a fixed-size input to a fixed size output and vice-versa

- Public key encryption(special kind of block cipher):
  : encrypt for anyone, but decrypt for only key owner

- Digital signatures
  : sign by only key owner, but checked by anyone

Random bit generation

In order to generate the keys and nonces needed in cryptographic protocols, a source of random bits unpredictable for any adversary is needed. The highly deterministic nature of computing environments makes finding secure seed values for random bit generation a non-trivial and often neglected problem.

Attack example: In 1995, Ian Goldberg and David Wagner (Berkeley University) broke the SSL encryption of the Netscape 1.1 Web browser using a weakness in its session key generation. It used a random-bit generator that was seeded from only the time of day in microseconds and two process IDs. The resulting conditional entropy for an eavesdropper was small enough to enable a successful brute-force search. `http://www.ddj.com/documents/s=965/ddj9601h/9601h.htm`

Examples for sources of randomness:

- → dedicated hardware random bit generators (amplified thermal noise from reverse-biased diode, unstable oscillators, Geiger counters)
- → high-resolution timing of user behaviour (e.g., key strokes and mouse movement)

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec/

Random bit generation

- → high-resolution timing of peripheral hardware response times (e.g., disk drives)
- → noise from analog/digital converters (e.g., sound card, camera)
- → network packet timing and content
- → high-resolution time

None of these random sources alone provides high-quality statistically unbiased random bits, but such signals can be fed into a hash function to condense their accumulated entropy into a smaller number of good random bits.

The provision of a secure source of random bits is now commonly recognised to be an essential operating system service. For example, the Linux `/dev/random` device driver uses a 4096-bit large *entropy pool* that is continuously hashed with keyboard scan codes, mouse data, inter-interrupt times, and mass storage request completion times in order to form the next entropy pool. Users can provide additional entropy by writing into `/dev/random` and can read from this device driver the output of a cryptographic pseudo random bit stream generator seeded from this entropy pool. Operating system boot and shutdown scripts preserve `/dev/random` entropy across reboots on the hard disk.

`http://www.cs.berkeley.edu/~daw/rnd/`
`http://www.ietf.org/rfc/rfc1750.txt`

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec/

Stream Cipher Example: RC4

- A self-modifying lookup table
- Table always contains a permutation of the byte values $0,1,\ldots,255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a **byte**
- Use keystream bytes like a one-time pad

RC4 algorithm

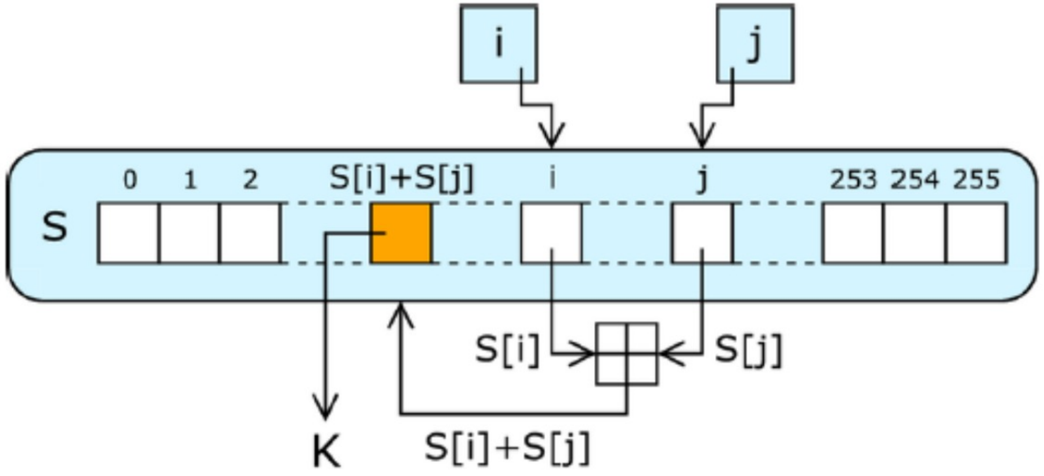
- `S[]` is permutation of `0,1,...,255`
- `key[]` contains `N` bytes of key

```
for i = 0 to 255
    S[i] = i
    K[i] = key[i (mod N)]
next i
j = 0
for i = 0 to 255
    j = (j + S[i] + K[i]) mod 256
    swap(S[i], S[j])
next i
i = j = 0
```

RC4 algorithm

- for each keystream byte, swap elements and select byte

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```



Blockchiphers

Encryption is performed today typically with blockcipher algorithms, which are key-dependent permutations, operating on bit-block alphabets. Typical alphabet sizes: 2^{64} or 2^{128}

$$E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$

$\longrightarrow$ Confusion – make relationship between key and ciphertext as complex as possible

$\longrightarrow$ Diffusion – remove statistical links between plaintext and ciphertext

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec/

Types of Attacks on Block Ciphers

- known plaintext attack
  - eg. hacker can buy a decoder for a broadcast entertainment system, watch a lot of movies and compare them with the enciphered broadcast signal
- chosen plaintext attack
  - eg. WWII Japanese intentions for an island 'AF': Midway's commander sent unencrypted message reporting problems with its fresh water condenser, then intercepted Japanese report 'AF is short of water' → Bingo!

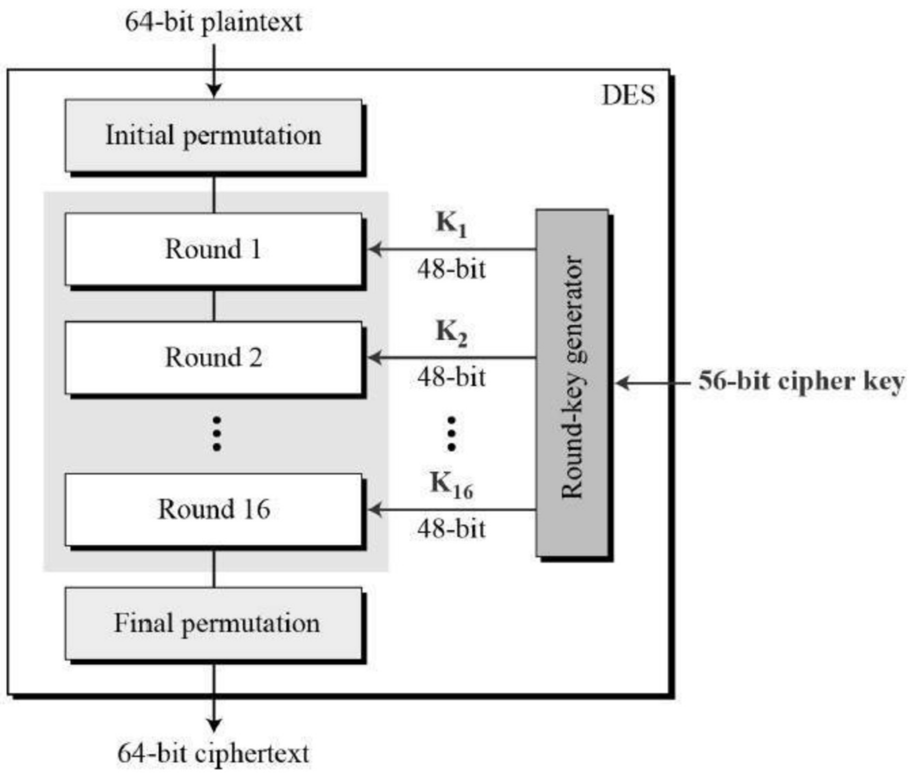
Types of Attacks on Block Ciphers

- chosen ciphertext attack
- chosen plaintext/ciphertext attack: hacker gets temporary access to a cryptographic device while its authorized user is out and tries out the full range of permitted operations for a while with data of their choice

- related key attack: hacker makes queries that will be answered using keys related to the target key K, such that K+1 and K+2 …

Data Encryption Standard

- DES developed in 1970's (IBM's Lucifer cipher)
- DES was U.S. government standard
- DES development was controversial
  - NSA secretly involved
  - Design process was secret
  - Key length reduced from 128 to 56 bits
  - Subtle changes to Lucifer algorithm
- DES is a Feistel cipher with…
  - 64 bit block length, 56 bit key length
  - 16 rounds (each round is simple for a block cipher)
  - 48 bits of key used each round (subkey)
- Security depends heavily on "S-boxes"
  - Each S-boxes maps 6 bits to 4 bits



Triple DES(3DES)

- Today, 56 bit DES key is TOO(!!!) small
  - Exhaustive key search is feasible
- But old systems still use DES, so what to do?
- **Triple DES** or **3DES** (112 bit key)
  - $C = E_{K1}(D_{K2}(E_{K1}(P)))$
  - $P = D_{K1}(E_{K2}(D_{K1}(C)))$
- Why Encrypt-Decrypt-Encrypt with 2 keys?
  - Backward compatible: $E_K(D_K(E_K(P))) = E_K(P)$
  - And 112 bits is enough(???)
- Why not $C = E_K(E_K(P))$?
  - Trick question --- it's still just 56 bit key

Double DES(2DES) ?

- Why not $C = E_{K2}(E_{K1}(P))$?
- A **known plaintext** attack: Meet-in-the-Middle
  - Pre-compute table of $E_{K1}(P)$ for every possible key K_1 (resulting table has 2^{56} entries)
  - Then for each possible K_2 compute $D_{K2}(C)$ until a match in table is found
  - When match is found, have $E_{K1}(P) = D_{K2}(C)$
  - Result gives us keys: $C = E_{K2}(E_{K1}(P))$

Advanced Encryption Standard

- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent process
  - Many strong algorithms proposed
  - *Rijndael* Algorithm ultimately selected
- Iterated block cipher, but NOT Feistel cipher

AES Overview

- Block size: 128 bits (others in Rijndael)
- Key length: 128, 192 or 256 bits (independent of block size)
- 10 to 14 rounds (depends on key length)
- Each round uses 4 functions (3 “layers”)
  - ByteSub (nonlinear layer)
  - ShiftRow (linear mixing layer)
  - MixColumn (nonlinear layer)
  - AddRoundKey (key addition layer)

Other Symmetric Block Ciphers

- IDEA, Blowfish, RC6, Serpent, SEED, …

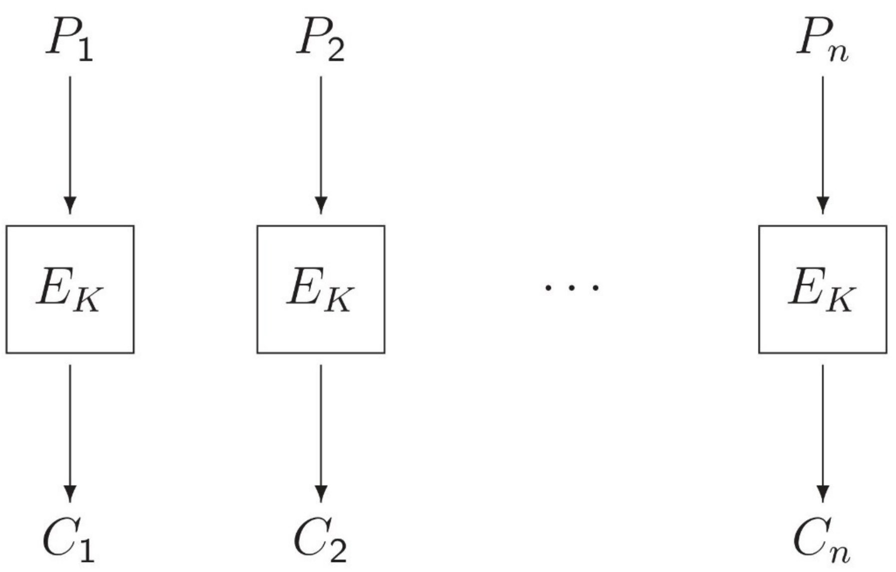
Modes of Operation for Dealing Multiple Blocks

- How to encrypt multiple blocks?
- Do we need a new key for each block?
  - As bad as (or worse than) a one-time pad!
- Encrypt each block independently?
- Make encryption depend on previous block?
  - That is, can we “chain” the blocks together?
- How to handle partial blocks?

Various Modes

- Electronic Codebook (**ECB**) mode
  - Encrypt each block independently
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
- Counter Mode (**CTR**) mode
- Cipher Feedback mode(CFB)
- Output Feedback mode(OFB)
- Galois Counter mode(GCM)
  - Parallelizable
  - (encryption + authentication) at once

Electronic Code Book (ECB)



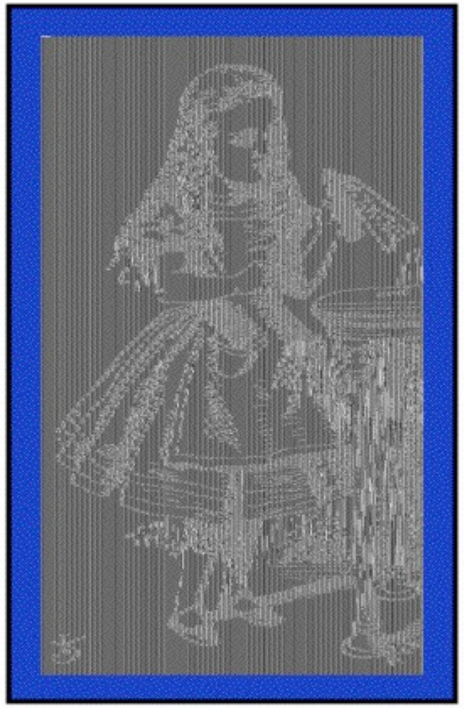
In the simplest **mode of operation** standardised for DES, AES, and other block ciphers, the message is cut into n-bit blocks, where n is the block size of the cipher, and then the cipher function is applied to each block individually.

http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec/

Alice Hates ECB Mode

- Alice’s uncompressed image, and ECB encrypted



  - Why does this happen?
  - Same plaintext yields same ciphertext!

Electronic Code Book (ECB)

The Electronic Code Book (ECB) mode has a number of problems and is therefore generally not recommended for use:

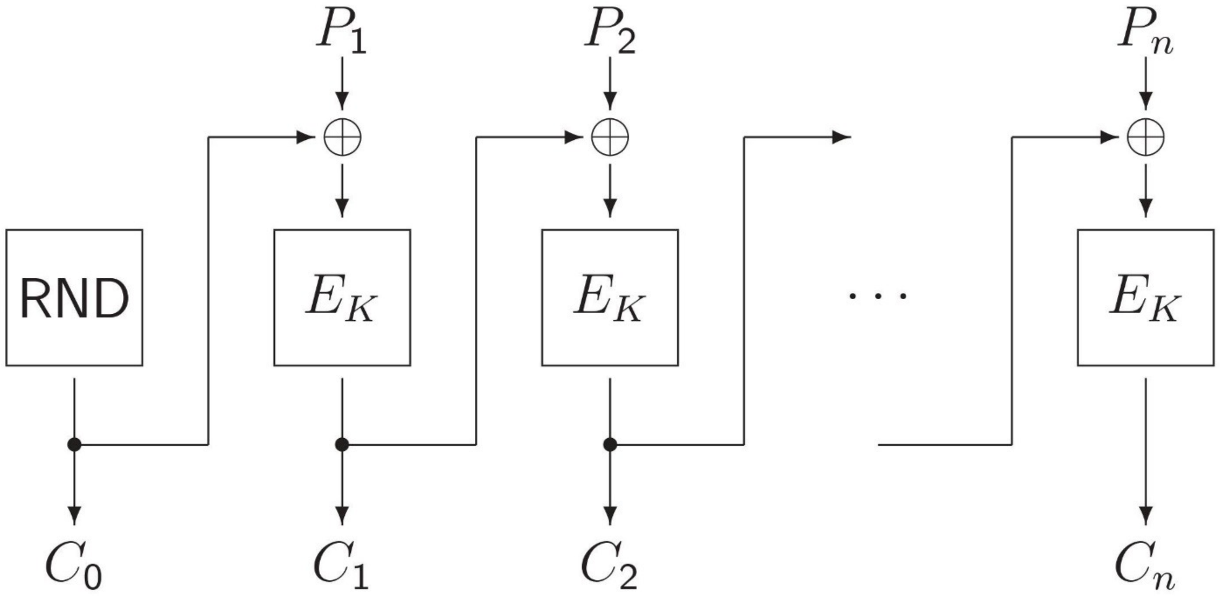
- $\longrightarrow$ Repeated plaintext messages can be recognised by the eavesdropper as repeated ciphertext. If there are only few possible messages, an eavesdropper might quickly learn the corresponding ciphertext.
- $\longrightarrow$ Plaintext block values are often not uniformly distributed, for example in ASCII encoded English text, some bits have almost fixed values. As a result, not the entire input alphabet of the block cipher is utilised, which simplifies for an eavesdropper building and using a value table of E_K.

Both problems can be solved by using other modes of operation than DES. Using a pseudo-random value as the input of the block cipher will use its entire alphabet uniformly, and independent of the plaintext content, a repetition of cipher input has to be expected only after $\sqrt{2^n} = 2^{\frac{n}{2}}$ blocks have been encrypted with the same key ($\rightarrow$ birthday paradox).

The Cipher Block Chaining mode XORs the previous ciphertext into the plaintext to achieve this, and the entire ciphertext is randomised by prefixing it with a random *initial vector* (IV).

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec/

Cipher Block Chaining (CBC)



$$C_i = E_K(P_i \oplus C_{i-1})$$

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec

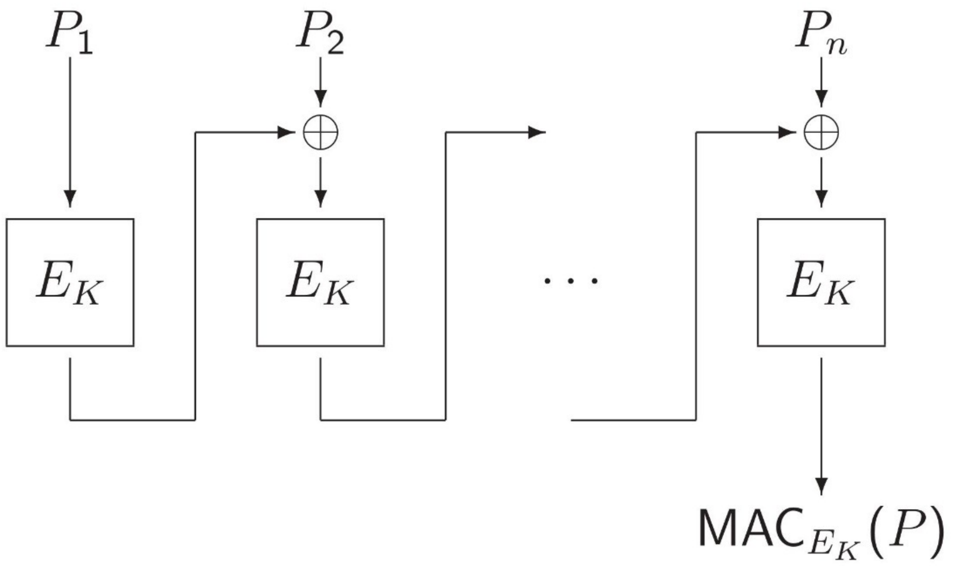
CBC Mode

- Identical plaintext blocks yield different ciphertext blocks — this is good!
- If C_1 is garbled to, say, G then

$$P_1 \neq C_0 \oplus D_K(G),\ P_2 \neq G \oplus D_K(C_2)$$

- But $P_3 = C_2 \oplus D_K(C_3),\ P_4 = C_3 \oplus D_K(C_4),\ \ldots$
- Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

Message Authentication Code (MAC)



A modification of CBC provides integrity protection for data. The initial vector is set to a fixed value (e.g., 0), and C_n of the CBC calculation is attached to the transmitted plaintext. Anyone who shares the secret key K with the sender can recalculate the MAC over the received message and compare the result. A MAC is the cryptographically secure equivalent of a checksum, which only those who know K can generate and verify.

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec

Various Modes

Counter Mode (CTR)

This mode obtains the pseudo-random bit stream by encrypting an easy to generate sequence of mutually different blocks, such as the natural numbers plus some offset O encoded as n-bit binary values:

$$R_i = E_K(i + O), \quad C_i = P_i \oplus R_i$$

It is useful for instance when encrypting files for which fast random access decryption is needed. The offset O can be chosen randomly and transmitted/stored with each encrypted message like an initial vector.

Cipher Feedback Mode (CFB)

$$C_i = P_i \oplus E_K(C_{i-1})$$

As in CBC, C_0 is a randomly selected initial vector, whose entropy will propagate through the entire ciphertext.

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec

Various Modes

Output Feedback Mode (OFM)

In this mode of encryption, the plaintext is simply XORed with the output of the above pseudo-random bit stream generator:

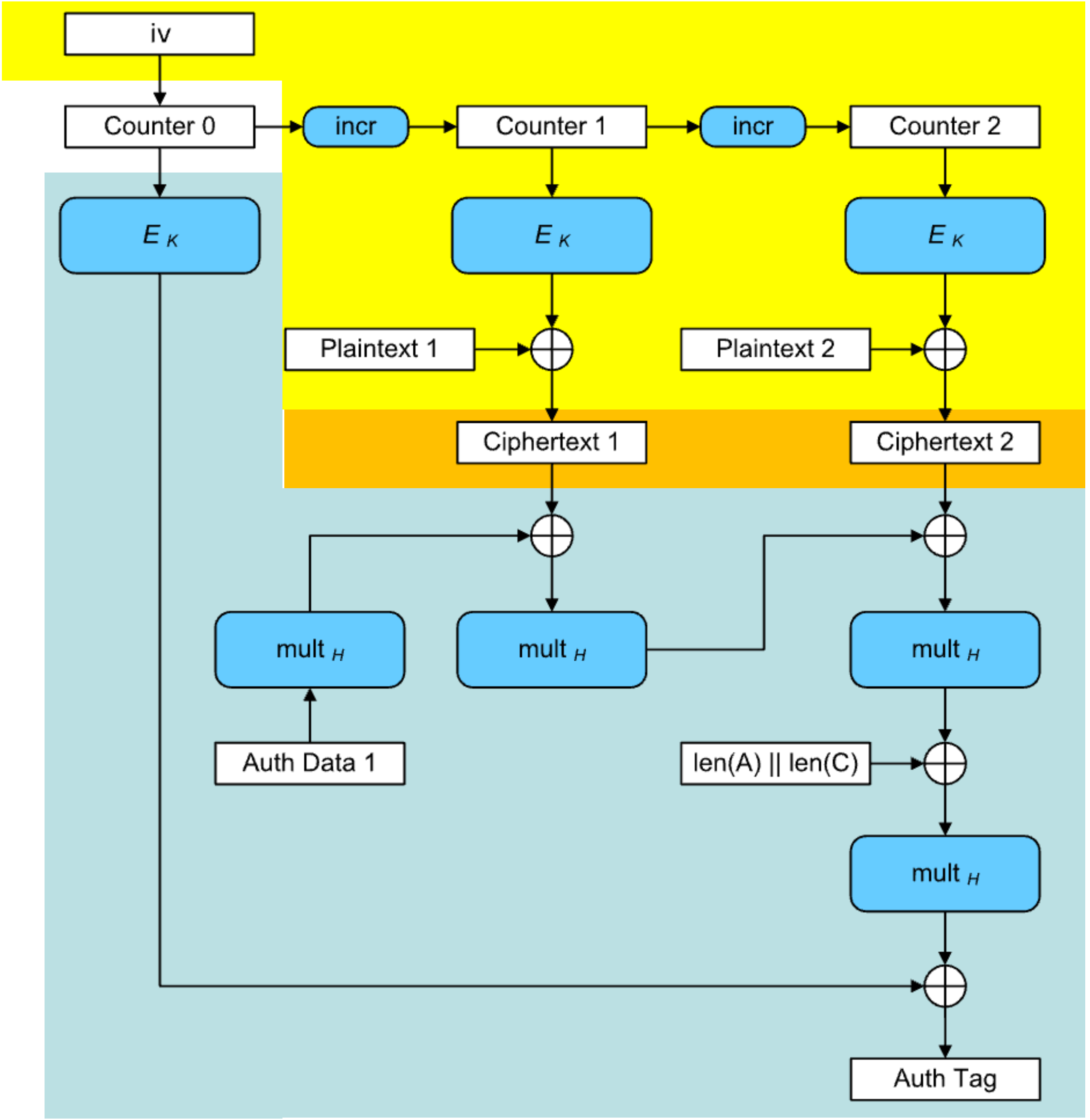
$$R_0 = 0, \quad R_i = E_K(R_{i-1}), \quad C_i = P_i \oplus R_i$$

from Introduction to Security by Markus Kuhn
http://www.cl.cam.ac.uk/teaching/0708/IntroSec

Galois Counter Mode(GCM)

- Authenticated Encryption with Associated Data(AEAD)
- CBC not efficient for bulk encryption when protecting confidentiality AND integrity
- New mode for authenticated encryption
- One invocation of block cipher per text block
- Parallelizable for high throughput encryption / decryption
- Basic Idea
  - encryption in a variant of counter mode
  - authentication tag(for integrity) computed using the ciphertexts as coefficients of a polynomial over a Galois field $GF(2^{128})$

Galois Counter Mode(GCM)



Q & A

aiclasshongik@gmail.com