# CSCI471/971
# Modern Cryptography
## Modern Symmetric-key Encryption

Rupeng Yang
SCIT UOW

# *RoadMap*

- Week 1-2:  Preliminaries
- Week 3: How to construct symmetric-key encryption

# Roadmap

Classical Ciphers

One-Time Pad

# How to Design A Block Cipher

# Confusion and Diffusion: Design Principles

Shannon (1949) introduced two concepts

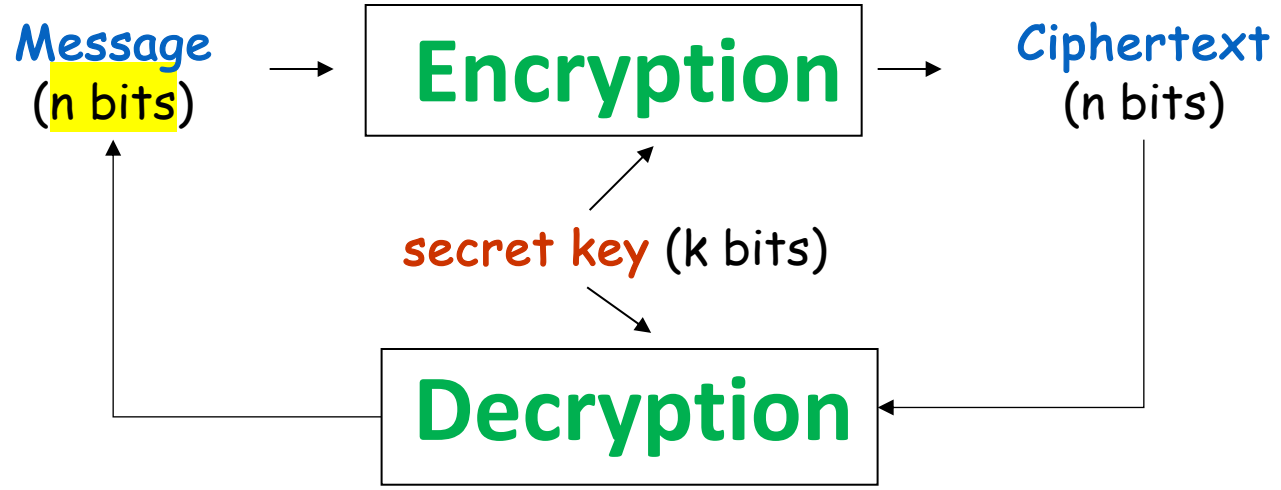
- **Confusion**: If a single bit in a key is changed, the returned ciphertext will be significantly different.
- **Diffusion:** If a single bit in a plaintext is changed, the returned ciphertext will be significantly different.

# Property: The avalanche effect

- A desirable property for encryption is that small changes in either plaintext or key should result in significant changes in the ciphertext.

- For example, a cipher might be said to have the avalanche effect if changing a single bit (in either the key or plaintext) results in half the output being different.

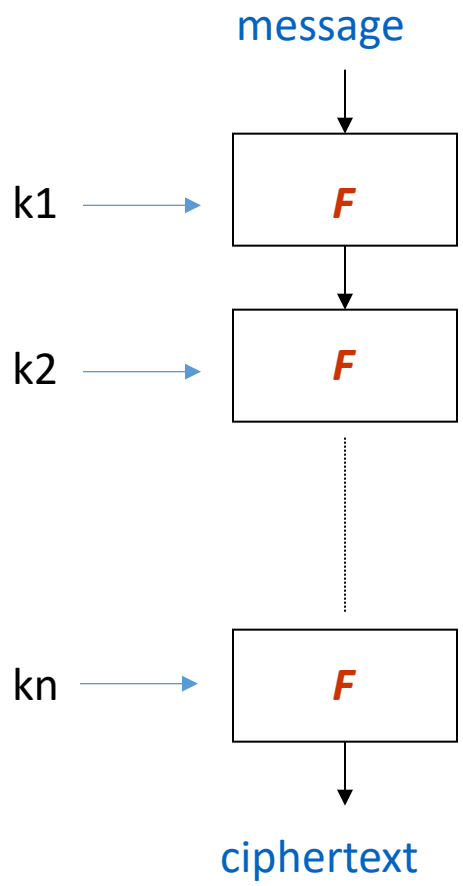
# Block cipher




- An encryption algorithm that takes a fixed length block of message letters **(plaintext)** and a **key** (*not necessarily the same length*), and produces a block of **ciphertext** of the same length as the plaintext.
- A decryption algorithm that takes the ciphertext and the key and produces the plaintext.
- The key is **reused** for different plaintext blocks
- Typical block sizes (value of n): 64 bits, 128 bits
- Key sizes (value of k): 56 bits (DES), 128/192/256 bits (AES)

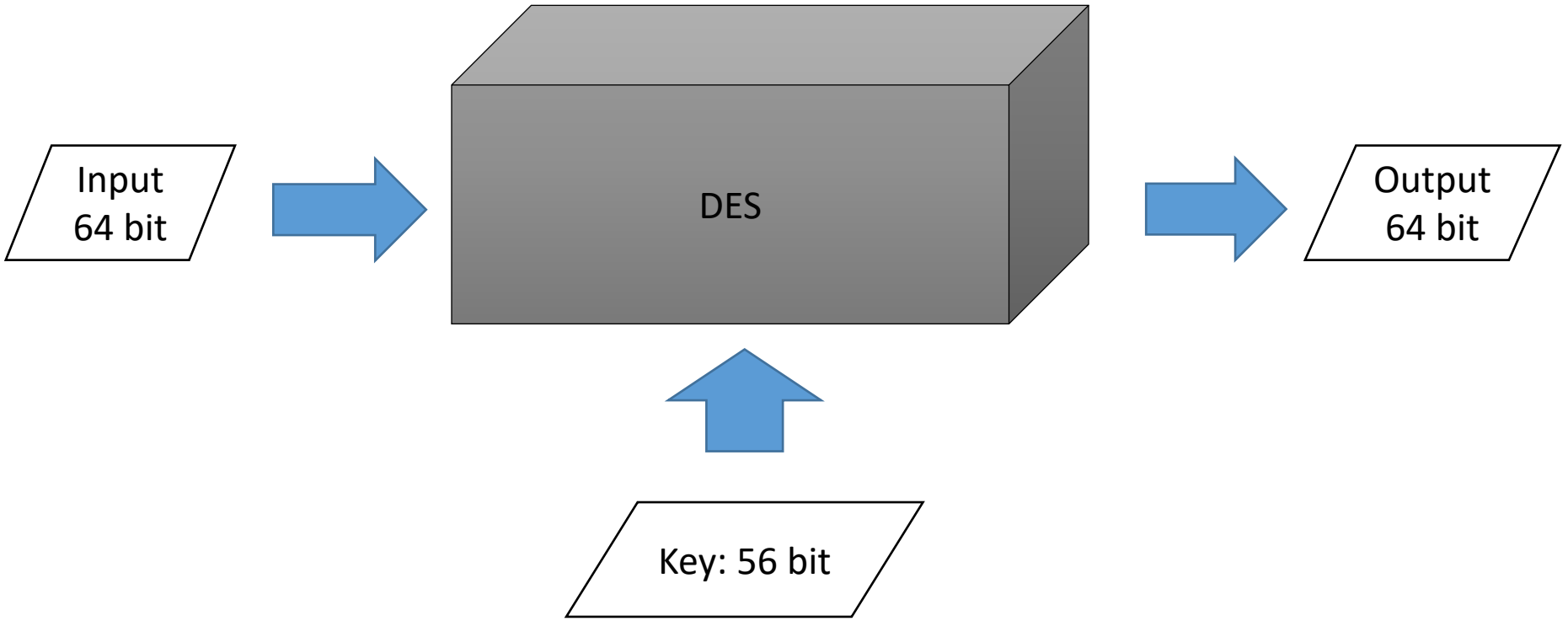
# Common Block Cipher Design approaches

- Iterated cipher
  - Each iteration is called a *round*. The output of each round is a function of the output of the previous round and a sub-key derived from the full secret key by a key-scheduling algorithm.
  - DES – 16 rounds, AES – 10 rounds
- Each Round inside DES is: Feistel structure
- Each Round inside AES is: Substitution-Permutation Networks

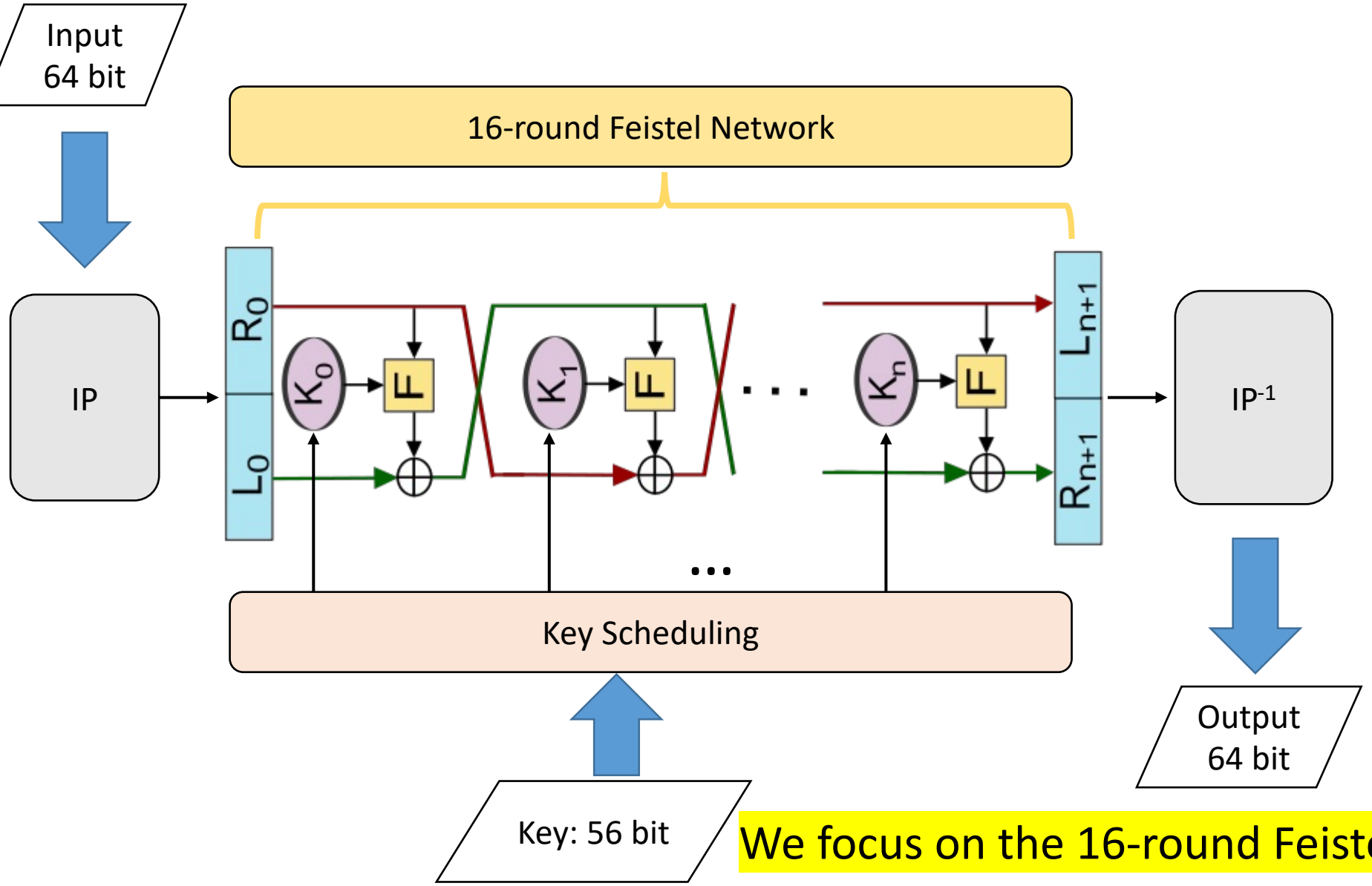


# The Development of DES

- It became clear in the 1970‘s that there was a need for a standard encryption algorithm, because:
  - Advances in information technology and the need for security to support this. Government and Commercial parties were beginning to use computers extensively.
  - Required security could not be provided by ad-hoc algorithms.
  - Different devices had to be able to exchange encrypted information.
- The standard needed to have the following properties:
  - It needed a high level of security.
  - It needed to be completely specified. In accordance with Kirchoff’s Law the security shouldn’t rely on a hidden part of the algorithm.
  - It needed to be economical to implement.
  - It needed to be adaptable to diverse application.
- In 1973 National Bureau of Standards published a solicitation for cryptosystems in the Federal Register. This lead ultimately to the development of the ***Data Encryption Standard***, or ***DES***.
- DES was developed at IBM based on an earlier cipher system known as Lucifer.


Input
64 bit
DES
Output
64 bit
Key: 56 bit

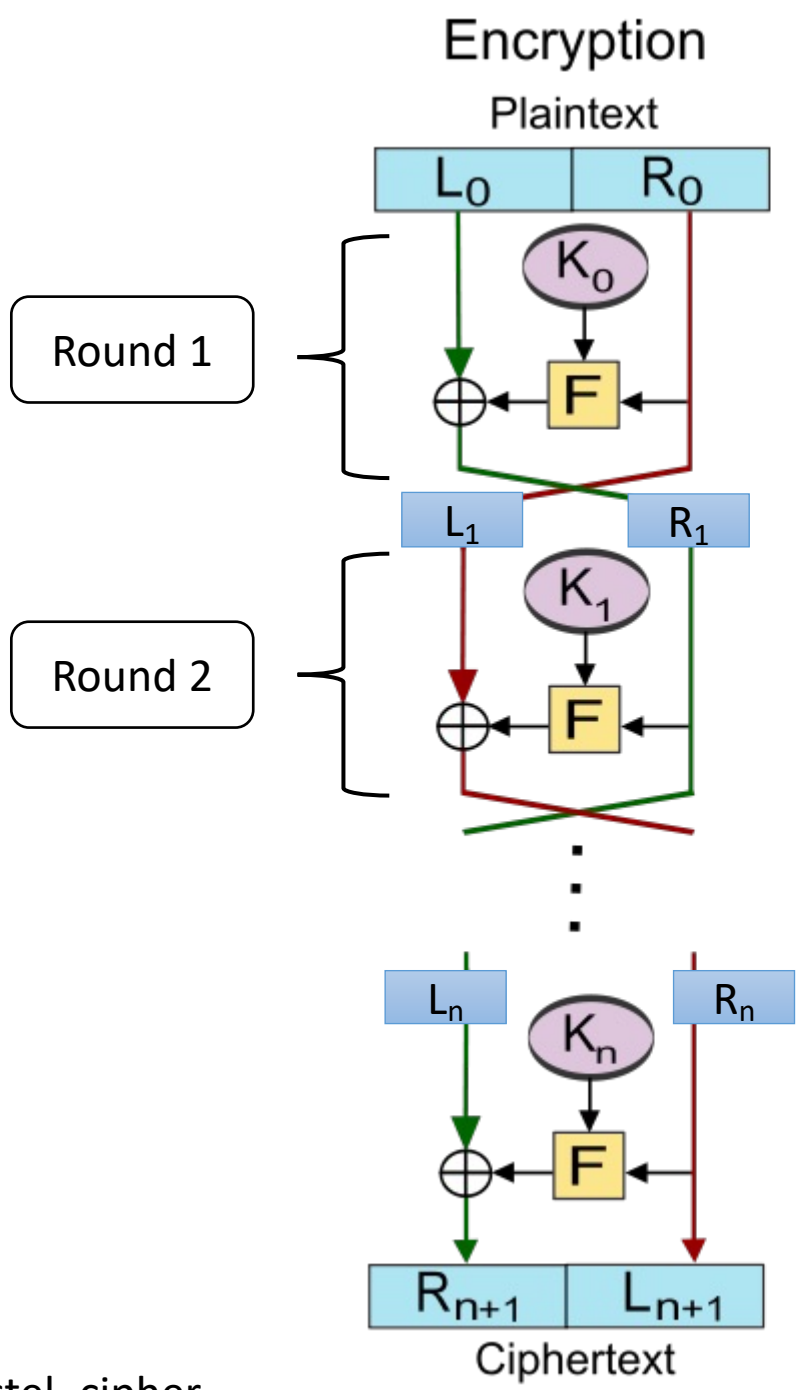
# DES




We focus on the 16-round Feistel Network!

# Feistel network (aka Feistel cipher, Feistel structure)

- Introduced by Feistel
- Many block ciphers follow this structure
- It simplifies the design of secure (block)ciphers by introducing the following design philosophy:
  - Increase security (confusion and diffusion) by having *multiple rounds*
  - Building blockcipher on a simpler function F that is *not* decryptable
  - …




$L_1 = R_0$

$R_1 = F(R_0, K_0) \oplus L_0$

$L_2 = R_1$

$R_2 = F(R_1, K_1) \oplus L_1$

.
.
.

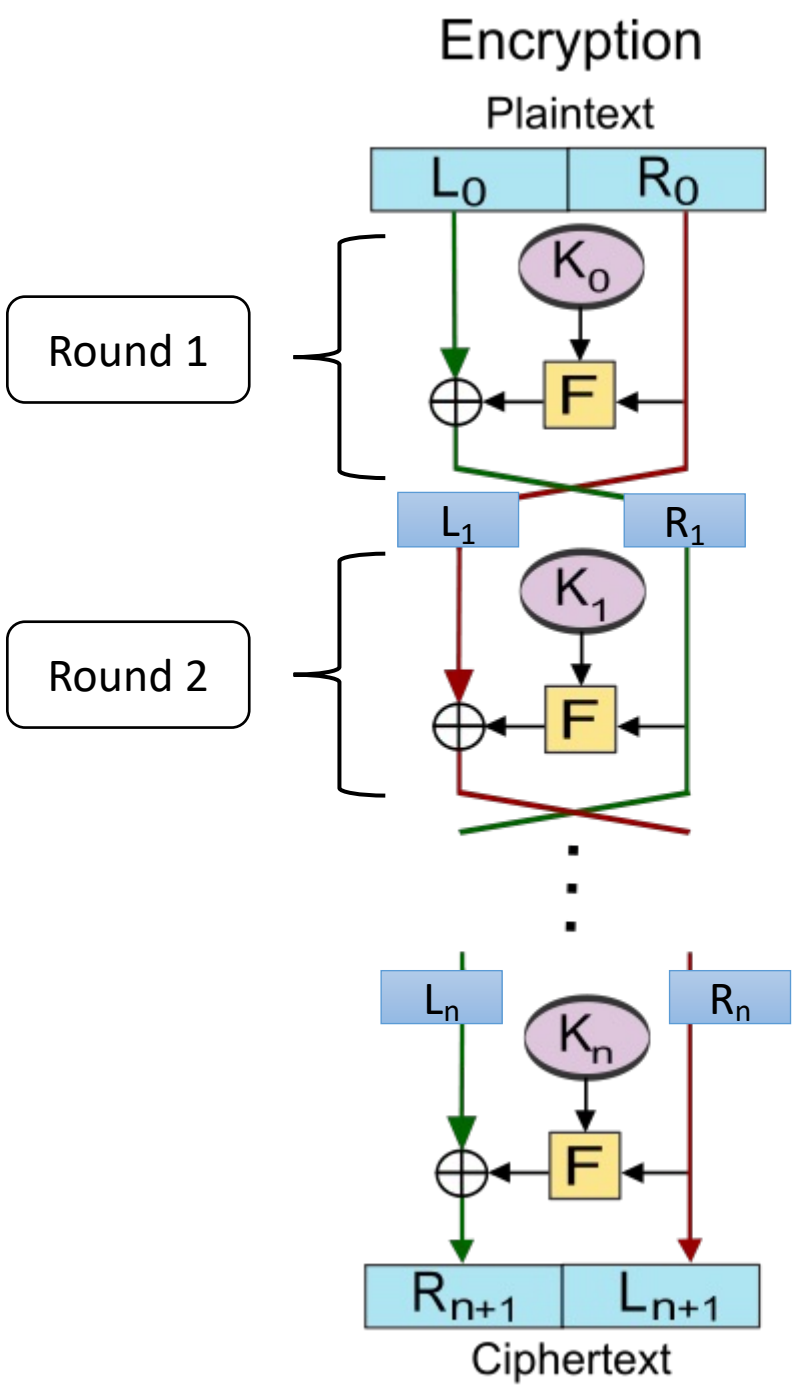
$L_{n+1} = R_n$

$R_{n+1} = F(R_n, K_n) \oplus L_n$

https://en.wikipedia.org/wiki/Feistel_cipher

# Feistel Structure

- At each round, only the right half of the message is put into the function F based on the round sub-key $K_i$
- After that, the left part of the message is XOR with the output of function F, and the right part is kept unchanged.
- Then, left part and right part exchange position at the end of each round (*except the last round*)
  - In the last round, the left part and the right part are not exchanged.




Given $L_1$ and $R_1$ and $K_0$, can we get back $L_0$ and $R_0$?

$R_0 = L_1$

$L_0 = F(L_1, K_0) \oplus R_1$

Given $L_2$ and $R_2$ and $K_1$, can we get back $L_1$ and $R_1$?

$R_1 = L_2$

$L_1 = F(L_2, K_1) \oplus R_2$

.
.
.

Given $L_{n+1}$ and $R_{n+1}$ and $K_n$, can we get back $L_n$ and $R_n$?

$R_n = L_{n+1}$

$L_n = F(L_{n+1}, K_n) \oplus R_{n+1}$

$L_1 = R_0$

$R_1 = F(R_0, K_0) \oplus L_0$

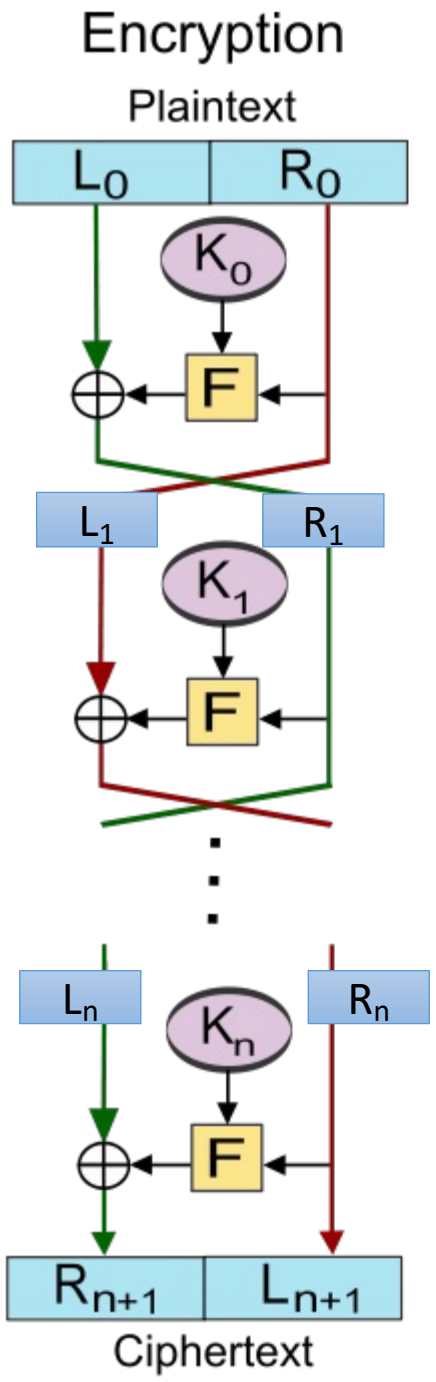
$L_2 = R_1$

$R_2 = F(R_1, K_1) \oplus L_1$

.
.
.

$L_{n+1} = R_n$

$R_{n+1} = F(R_n, K_n) \oplus L_n$




Given $L_1$ and $R_1$ and $K_0$, can we get back $L_0$ and $R_0$?

$R_0 = L_1$

$L_0 = F(L_1, K_0) \oplus R_1$

Given $L_2$ and $R_2$ and $K_1$, can we get back $L_1$ and $R_1$?

$R_1 = L_2$

$L_1 = F(L_2, K_1) \oplus R_2$

.
.
.

Given $L_{n+1}$ and $R_{n+1}$ and $K_n$, can we get back $L_n$ and $R_n$?

$R_n = L_{n+1}$

$L_n = F(L_{n+1}, K_n) \oplus R_{n+1}$

$L_1 = R_0$

$R_1 = F(R_0, K_0) \oplus L_0$

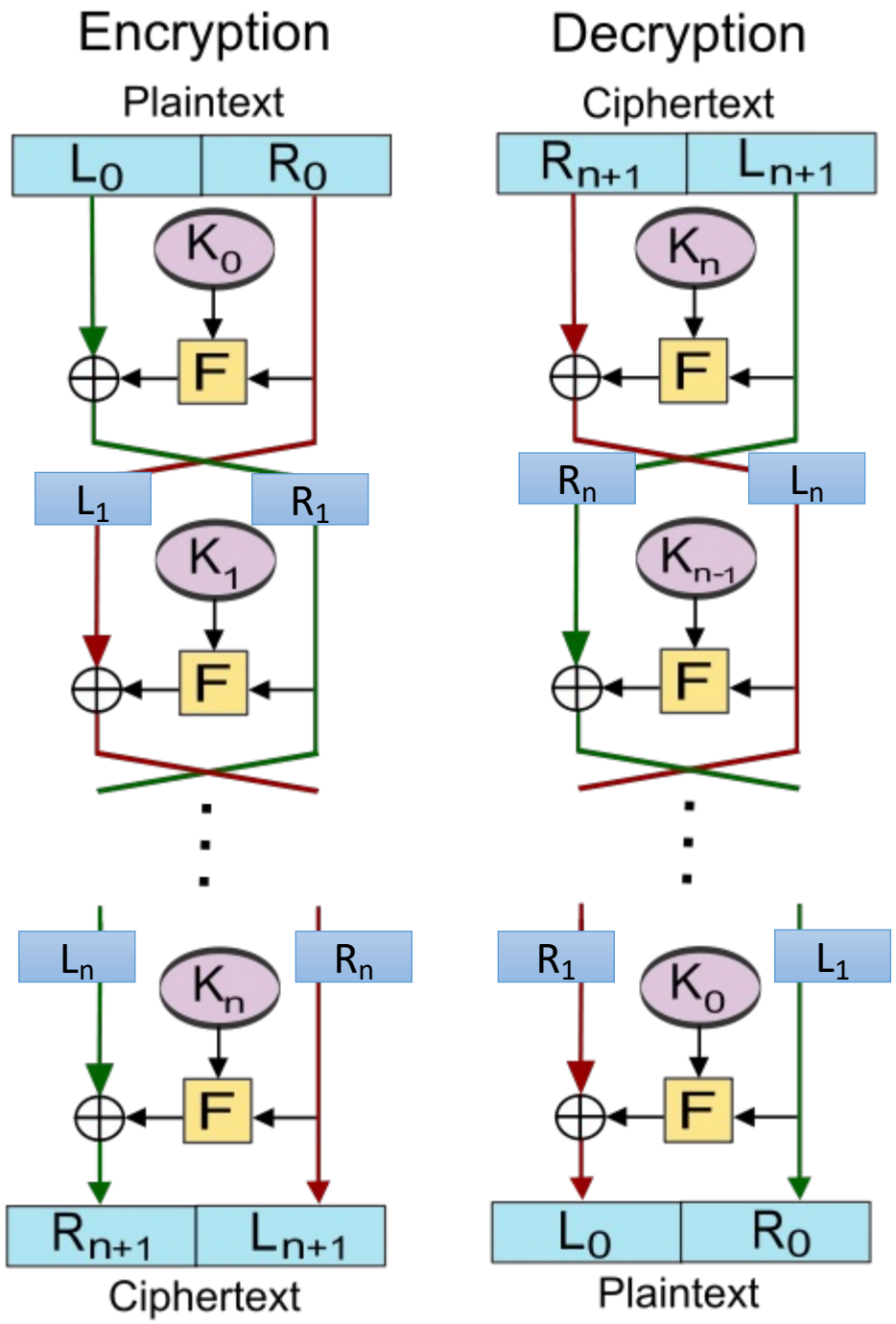
$L_2 = R_1$

$R_2 = F(R_1, K_1) \oplus L_1$

.
.
.

$L_{n+1} = R_n$

$R_{n+1} = F(R_n, K_n) \oplus L_n$




$R_n = L_{n+1}$

$L_n = F(L_{n+1}, K_n) \oplus R_{n+1}$

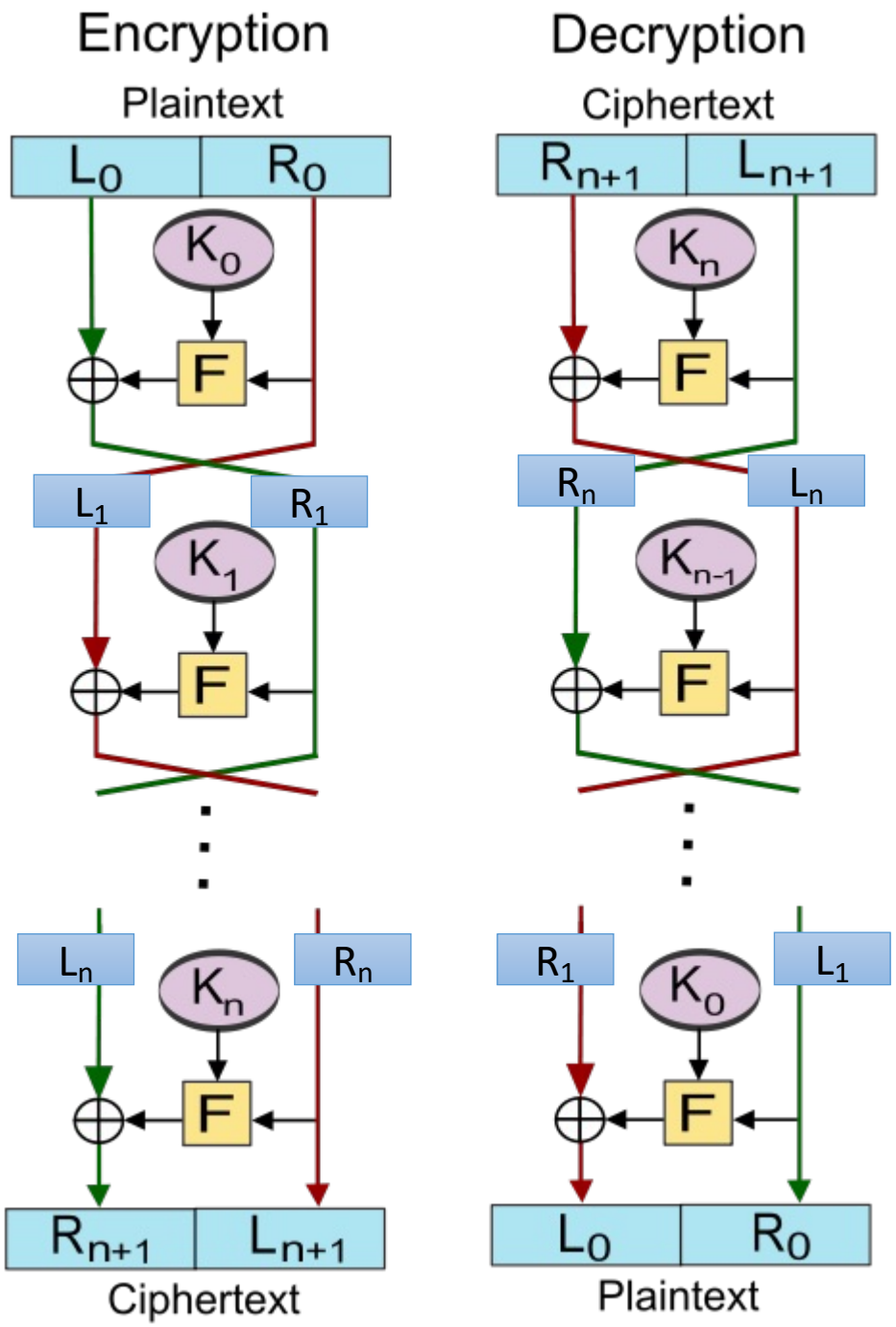
$R_{n-1} = L_n$

$L_{n-1} = F(L_n, K_{n-1}) \oplus R_n$

.
.
.

$R_0 = L_1$

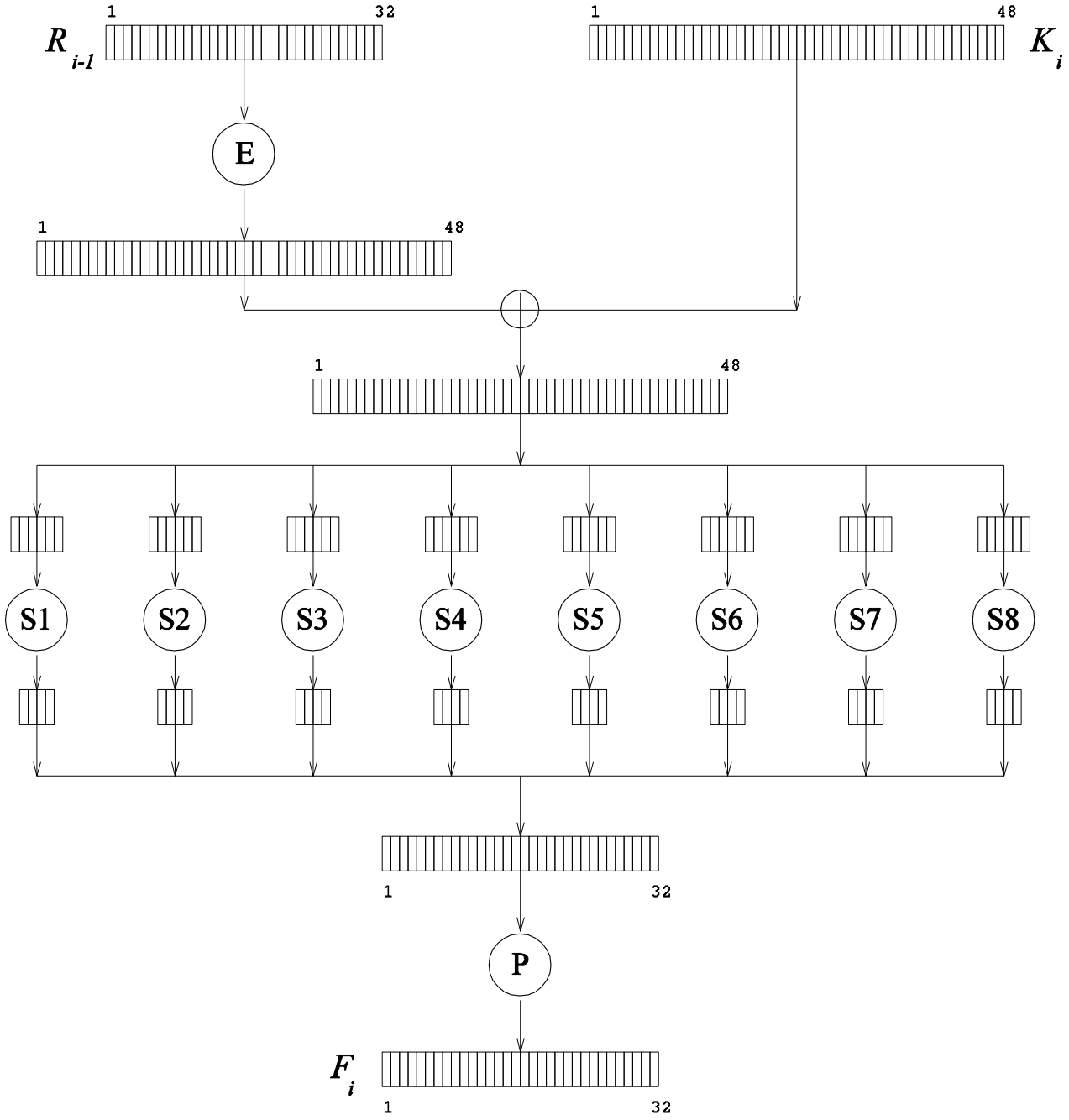
$L_0 = F(L_1, K_0) \oplus R_1$




## Feistel Structures

- Encryption and Decryption are the same (with subkeys being applied in reverse order)!
- It remains to show how to design a suitable function F.

# $F(k_i, R_{i-1})$




## The function F

- E = Extension and Permutation
- P = Permutation

- How to design S1 to S8?

# Substitution Box (S-Box)

- There are 8 S-boxes.
- Take the 6-bit input $b_1, b_2, b_3, b_4, b_5, b_6$
- Interpret $b_1\ b_6$ as a row number, between 0 and 3.
- Interpret $b_2\ b_3\ b_4\ b_5$ as a column number, 0 through 15.

| | S[0] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2. | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

# Security of DES

- DES has $2^{56} \approx 10^{17}$ keys. Given a plaintext block and the corresponding ciphertext block, we can try each key to decrypt the ciphertext and compare the result with the known plaintext.
- If we tried one key every $10^{-6}$ seconds, this would take about $10^{11}$ seconds, or about 3170 years!
- But we can **test keys in parallel**.
  - Michael Wiener (1993) gave a detailed design for a key search machine.
  - In 1997, DES was broken by the DESCHALL Project, led by Rocke Verser, Matt Curtin, and Justin Dolske, using idle cycles of thousands of computers across the Internet.
  - In 1998, a custom DES-cracker was built by the Electronic Frontier Foundation (EFF), a cyberspace civil rights group, at the cost of approximately US$250,000.

# The development of AES

- The National Institute of Standards and Technology (NIST) worked with the international cryptographic community to develop an *Advanced Encryption Standard* (AES).
- NIST selected *Rijndael* at the end of a very long and complex evaluation process:
  - January 2, 1997. NIST announced the initiation of the project.
  - September 12, 1997. NIST made a formal call for algorithms.

  NIST specified minimum requirements:
    - Implement symmetric key cryptography as a block cipher.
    - Block size of 128 bits.
    - Key sizes of 128, 192, and 256 bits.

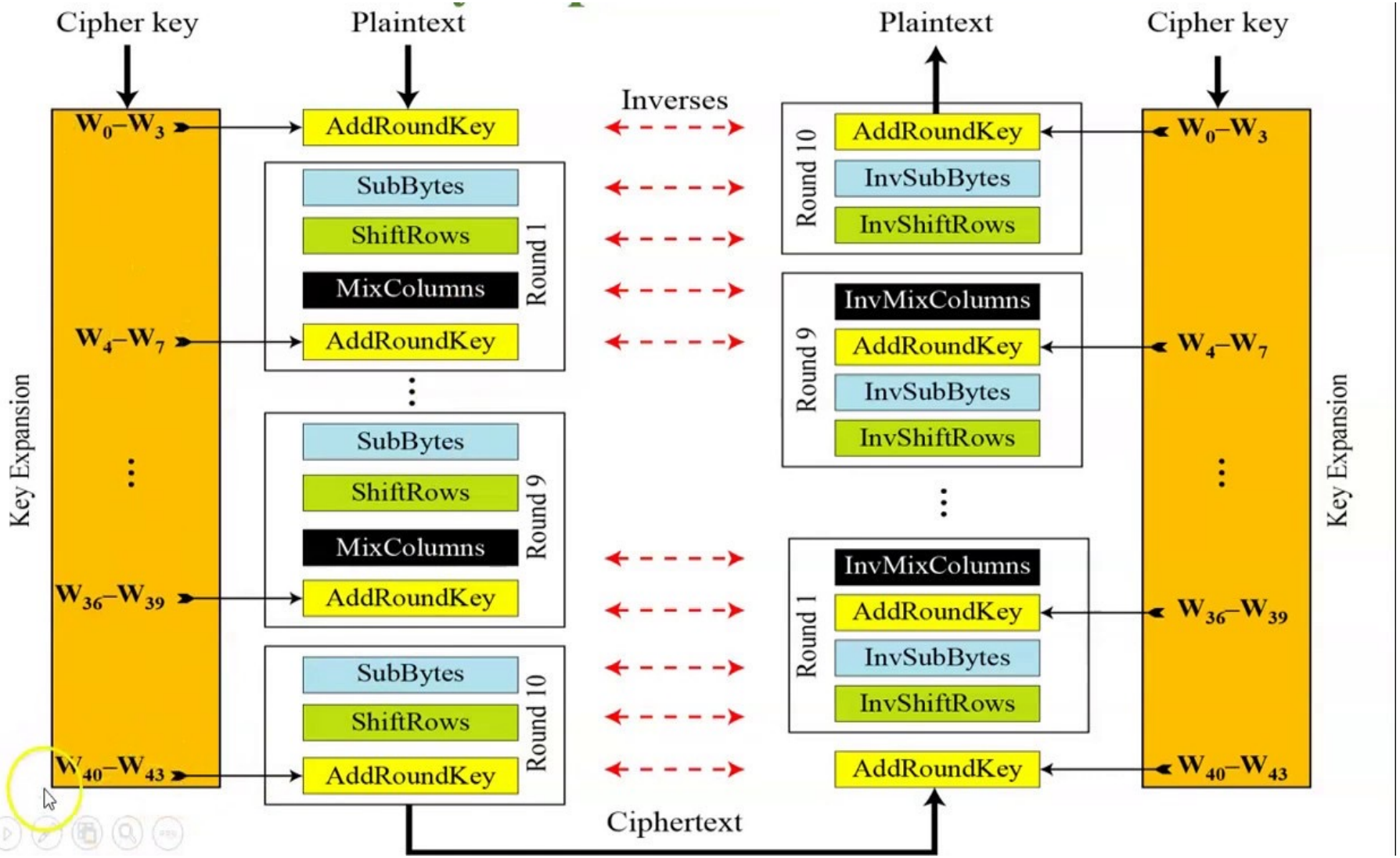
# The development of AES

- August 1998: NIST announced 15 AES candidate algorithms at the First AES Candidate Conference (AES1) and solicited public comments.
- March 1999: A Second AES Candidate Conference (AES2) was held to discuss the results of the analysis.
- August 1999: NIST announced its selection of five finalist algorithms from the fifteen candidates:
  - **MARS** (IBM; USA): Modified Feistel rounds in which one fourth of the data block is used to alter the other three fourths of the data block.
  - **RC6** (RSA Labs; USA): Feistel structure; 20 rounds.
  - **Rijndael** (Daemen, Rijmen; Belgium): Substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size.
  - **Serpent** (Anderson, Biham, Knudsen; UK, Israel, Norway): Substitution-linear transformation network consisting of 32 rounds.
  - **Twofish** (Counterpane; USA): Feistel network with 16 rounds, key-dependent S-boxes.

# Overview of Rijndael

- Both the key size and the block size may be chosen to be any of 128, 192, or 256 bits, although the AES only requires key size to be selectable in one of these lengths and the block size is fixed to be 128 bits long.
- Number of rounds is a function of the key length:
  - 10 rounds if the key is 128 bits long;
  - 12 rounds if the key is 192 bits long;
  - 14 rounds if the key is 256 bits long.

# Overview of Rijndael



# Security of AES

- The full AES is not broken.
- Reduced round versions are broken (*theoretically not practically*): There are attacks as follows.

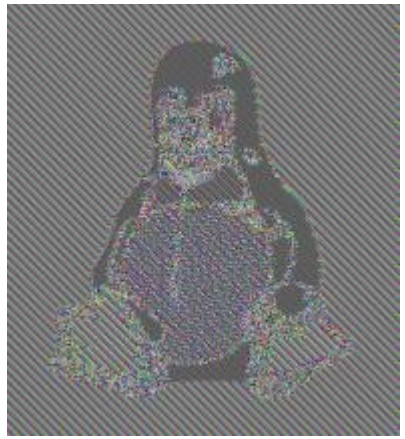
  7 rounds for 128-bit keys. (Chosen-plaintext)

  8 rounds for 192-bit keys. (Chosen-plaintext)

- The best theoretic attack breaks up to 8 rounds with over $2^{120}$ complexity for 128-bit keys and $2^{204}$ for 256-bit keys.

# How to use block ciphers
(to construct symmetric-key encryption)

# Is a blockcipher a secure symmetric-key encryption



What you will get if you use the blockciphers to encrypt each block of the image.

NO!

# Is a blockcipher a secure symmetric-key encryption

- The blockcipher is not secure since it maps the same plaintext block into the same ciphertext block.
  - This preserves the statistical property of the plaintext.
- To solve the problem, we can randomize the encryption process, i.e., mapping each plaintext block to multiple possible ciphertext blocks.
  - The problem: The size of the plaintext block is identical to the size of the ciphertext block in a blockcipher.
  - The solution: Each plaintext block can be encrypted into 2 ciphertext blocks.

# Solution:“randomize” the encryption process

Bais Encryption:

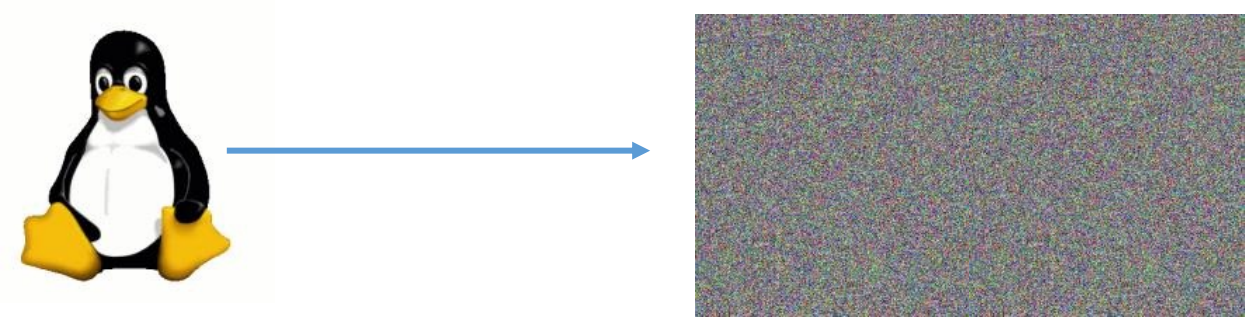
1. Run C←Enc(K, M)
2. Output  CT=C

Randomized Encryption:

1. Choose a random IV.
2. Compute M’=IV⊕ M
3. Run C←Enc(K, M’)
4. Output  CT=(IV, C)

# Is the randomized encryption secure

- Propbably yes…
- If we use the randomized encryption to encryt English texts, regarding each letter as a block:
  - The value of M’ will be different for each block, thus, no statistical property will appear in the ciphertext.
- If we use the randomized encryption to encrypt an image, we will get something like



- Can we have some more reliable evidences?
  - Yes!

# *Symmetric-Key Encryption*

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithms returns a key K

- Encrypt(K, M): Taking as input a message M and a key K, the encryption algorithm returns a ciphertext denoted by C.

$$C \leftarrow \text{Encrypt}(K,M)$$

- Decrypt(K,C): Taking as input a ciphertext C and a key K, the decryption algorithm returns a message M.

- The security parameter λ is denoted by security strength.

# *Symmetric-Key Encryption: Correctness*

- **Correctness**: For all generated K and all C←Encrypt(K,M), we have

$$\Pr[\text{Decrypt}(K,C)=M]=1$$

- **Correctness (with correctness error)**: For all generated K and all C←Encrypt(K,M), we have

$$\Pr[\text{Decrypt}(K,C)=M] \geq 1-\text{negl}(\lambda)$$

- We use negl to denote a negligible function, which is smaller than all inverse polynomial.

# Symmetric-Key Encryption: IND-CPA Security

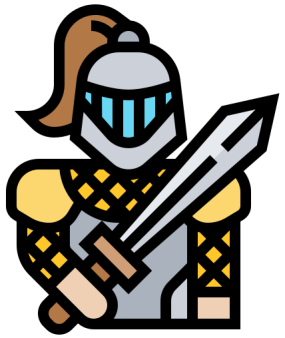
1. Run KeyGen to get k

2.1 Attacker sends m to challenger

2.2 Run Enc(k, m) to get c

2.3 Send c to the Attacker

Algorithms KeyGen, Enc, Des are public.



3.2 Randomly choose a bit b, run Enc(k, $m_b$) to get c*

3.1 Attacker sends $m_0$, $m_1$ to challenger

3.3 Send c* to the Attacker

We assume that ($|m_0|=|m_1|$)

4.2 Run Enc(k, m) to get c

4.1 Attacker sends m to challenger

4.3 Send c to the Attacker

An Encryption Scheme is Secure if ***NO*** efficient attacker can win with a probability of ½+1/poly(λ). Alternatively, that means,

Attacker wins if b = b'!

5. Attacker returns a guess bit b'

# *Security Model of Symmetric-Key Encryption (IND-CPA)*

Setup: The challenger chooses a random key K.

Phase 1: The adversary can choose any M for encryption queries and learns the encrypted result.

Challenge: The adversary can choose any two different messages M_0 and M_1. The challenger chooses a random b and computes the challenge ciphertext CT*=Enc(M_b, K), which is given to the adversary.

Phase 2: The adversary can choose any M for encryption queries.

Guess: The adversary returns the guess b' and wins if b'=b.

We say that the encryption is secure if no P.P.T adversary can win with a probability of ½+1/poly(λ).

# *Security Model of Symmetric-Key Encryption (IND-CCA)*

Setup: The challenger chooses a random key K.

Phase 1: The adversary can choose any M for encryption queries and learns the encrypted result; it can also choose any CT for decryption queries and learns the decryption result.

Challenge: The adversary can choose any two different messages M_0 and M_1. The challenger chooses a random b and computes the challenge ciphertext CT*=Enc(M_b, K), which is given to the adversary.

Phase 2: The adversary can choose any M for encryption queries and choose any CT different from CT* for decryption queries.

Guess: The adversary returns the guess c' and wins if b'=b.

We say that the encryption is secure if no P.P.T adversary can win with a probability of ½+1/poly(λ).

# A secure symmetric-key encryption scheme

Bais Encryption from blockcipher:

1. Run C←Enc(K, M)
2. Output CT=C

Randomized Encryption:

1. Choose a random IV.
2. Compute M'=IV⊕ M
3. Run C←Enc(K, M')
4. Output CT=(IV, C)

Why the solution is secure:

- The underlying blockcipher is assumed to be a **pseudorandom function (PRF)**, i.e., outputs of the blockcipher is indistinguishable from random values.
- If M' has never repeated, then the ciphertexts will just be some random values, i.e., no one could learn any information about the encrypted message from the ciphertext.
- The probability that M' will repeat is negligible since IV is randomly sampled.

Can we fix IV in every encryption?

# Another secure symmetric-key encryption scheme

Bais Encryption from blockcipher:

1. Run C←Enc(K, M)
2. Output CT=C

Randomized Encryption:

1. Choose a random IV.
2. Run W←Enc(K, IV)
3. Compute C=W⊕ M
4. Output CT=(IV, C)

Why the solution is secure:

- The underlying blockcipher is assumed to be a **pseudorandom function (PRF)**, i.e., outputs of the blockcipher is indistinguishable from random values.
- If IV never repeats, then M will be masked with random values. This will completely hide M.
  - We will show why it is secure to xor M with random values later.
- The probability that IV will repeat is negligible since it is randomly sampled.

# Secure symmetric-key encryption scheme

Bais Encryption:

1. Run C←Enc(K, M)
2. Output CT=C

Randomized Encryption I:

1. Choose a random IV.
2. Compute M'=IV⊕ M
3. Run C←Enc(K, M')
4. Output CT=(IV, C)

Randomized Encryption II:

1. Choose a random IV.
2. Run W←Enc(K, IV)
3. Compute C=W⊕ M
4. Output CT=(IV, C)

**Compare the left two encryptions.**

| | Security | Efficiency |
|---|---|---|
| **Basic** | Not Secure | Short Cipher |
| **I** | Secure | Long Cipher |
| **II** | Secure | Long Cipher |

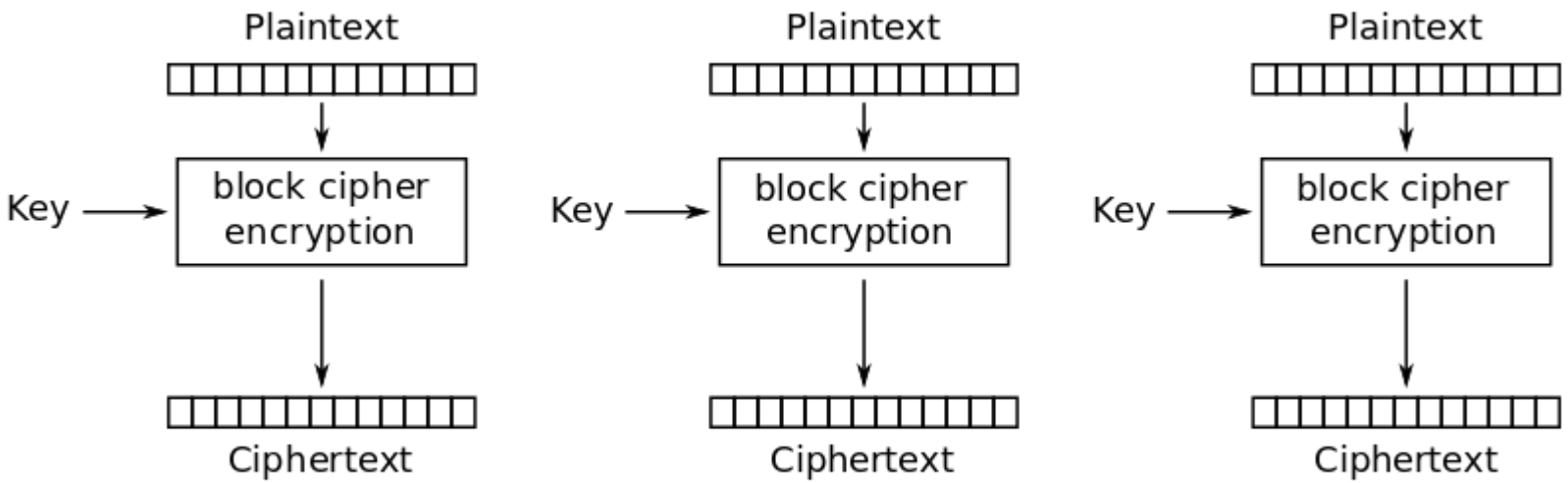
**The operation modes can reduce the cipher length**

# How to encrypt a long message using blockcipher: Securely and Efficiently (Operation Modes)

# Before you encrypts…

- First, we need to divide the long message into blocks.
- What if the length of the message is not a multiple of the block length? Then one needs to **pad** the message.
  - Padding example: Add a string 1000…0 to fill out the last block to the correct length. For decryption take the plaintext as being read back to the least significant 1.
    - Can we pad the message with 000….0?
  - may have to add a new block if the plaintext is already a multiple of the blocksize to avoid ambiguity.
- Next, we encrypt the blocks. We can choose the following different modes to encrypt the blocks.

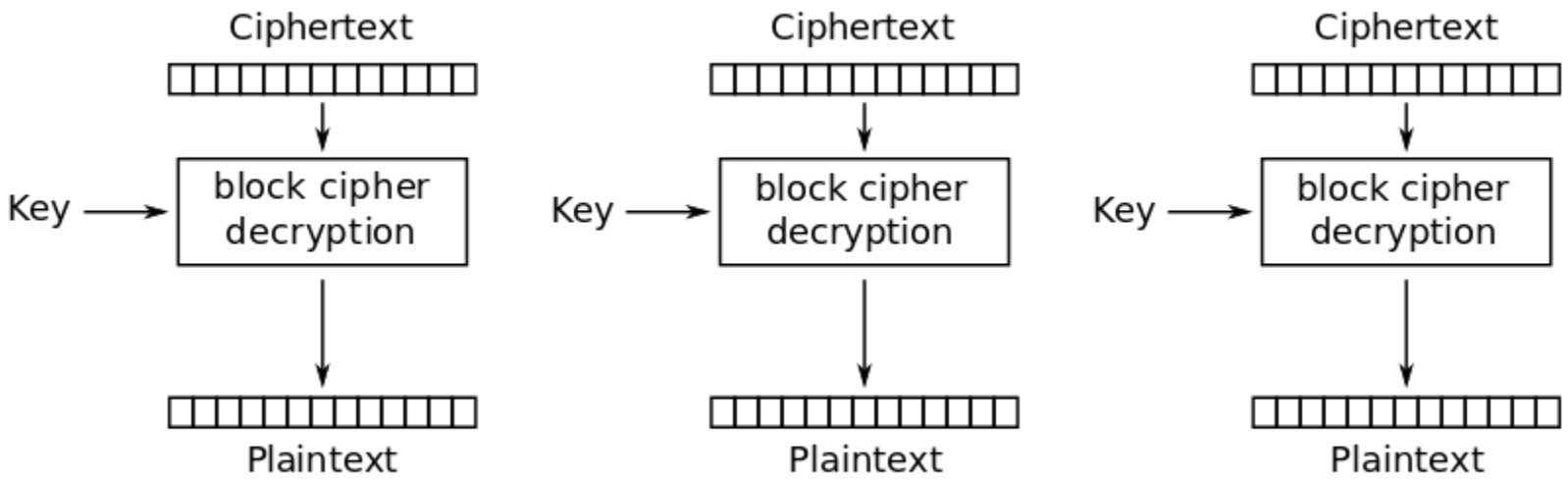
# Electronic Codebook (ECB) Mode




Electronic Codebook (ECB) mode encryption

The ECB mode is not secure!

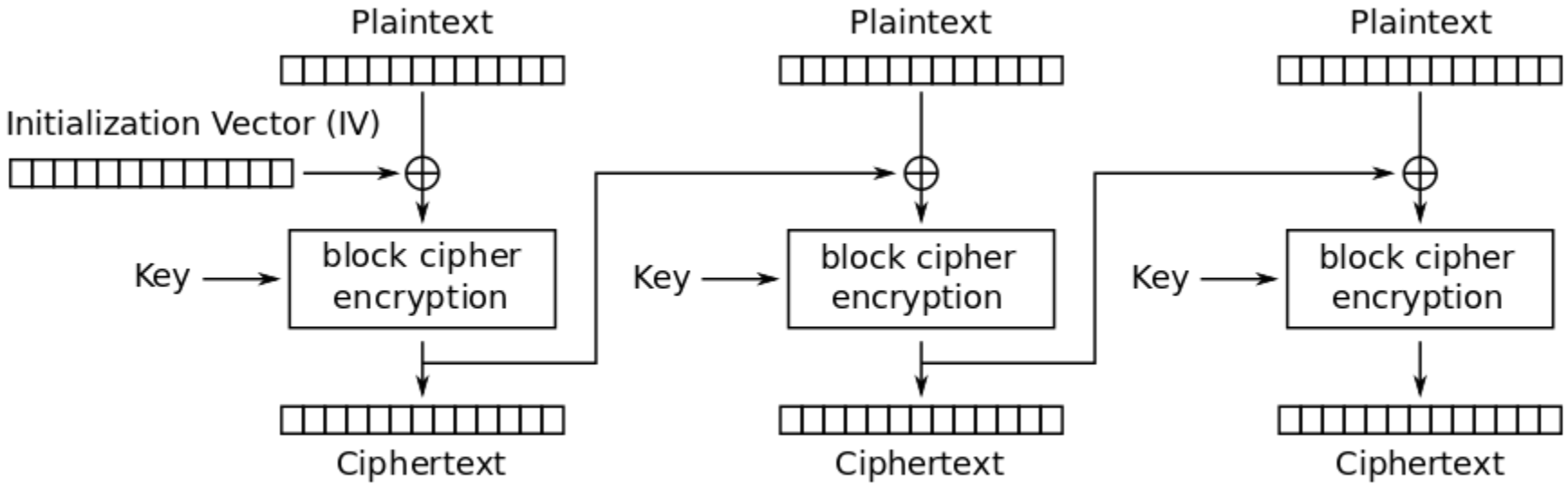
# ECB Mode Decryption




Electronic Codebook (ECB) mode decryption

The ECB mode is not secure!

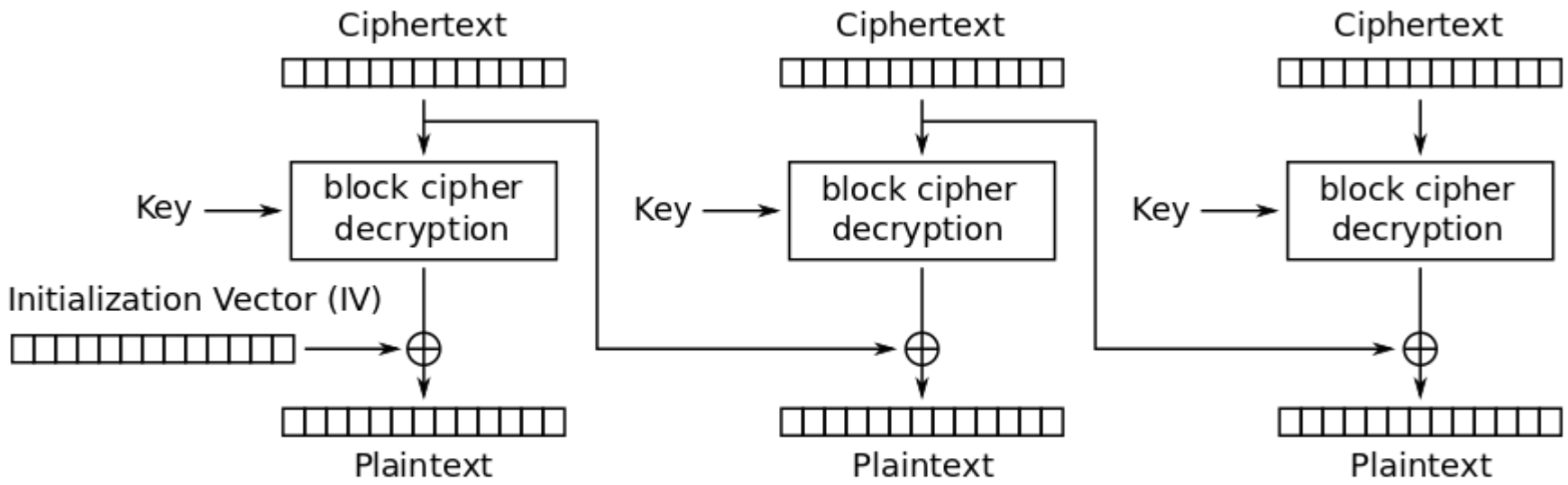
# Cipher Block Chaining (CBC) Mode




Cipher Block Chaining (CBC) mode encryption

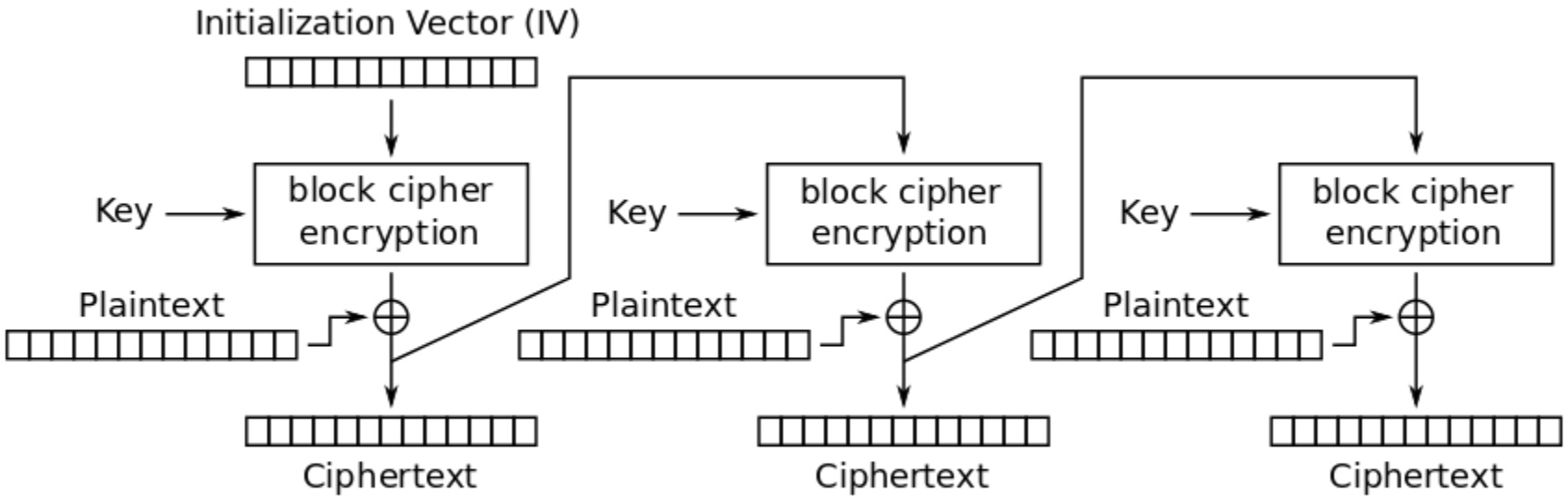
- IV is chosen uniformly at random
- This is built on the aforementioned randomized encryption I. But we use the ciphertext of each block as the IV of the next block.

# CBC Mode Decryption




Cipher Block Chaining (CBC) mode decryption

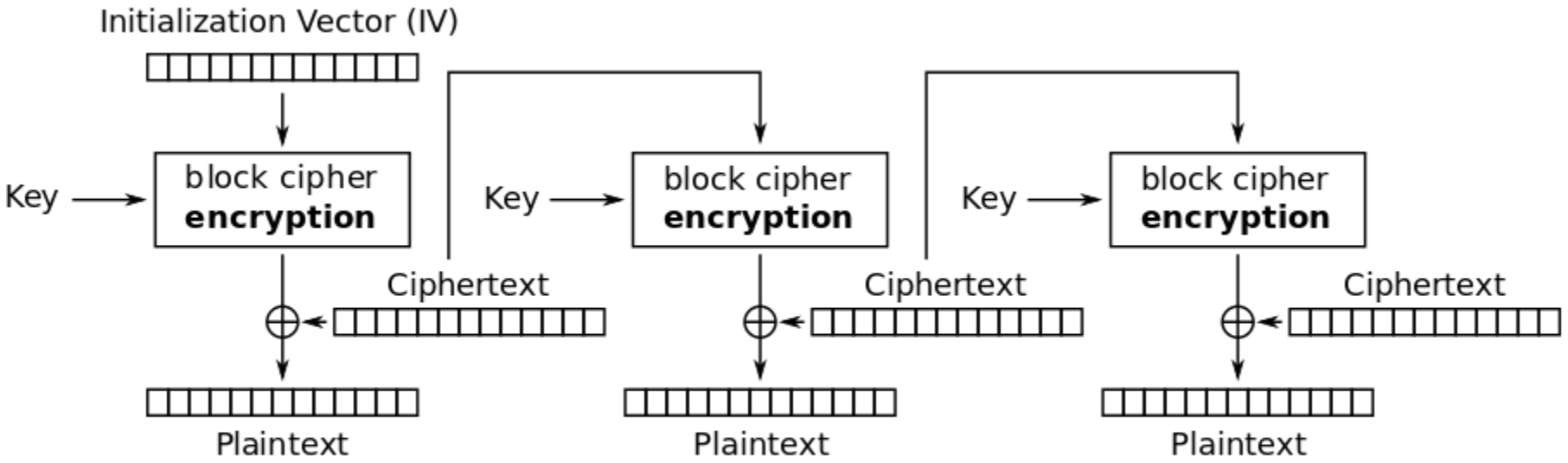
# Cipher Feedback (CFB) Mode




Cipher Feedback (CFB) mode encryption

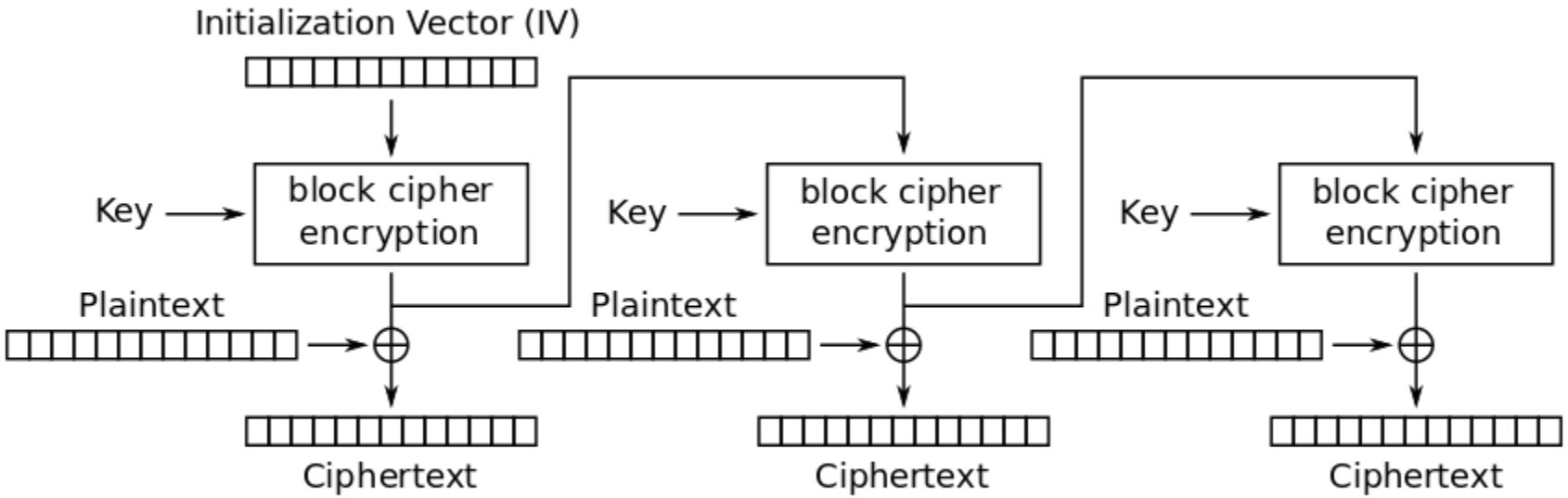
- IV is chosen uniformly at random
- This is built on the aforementioned randomized encryption II. But we use the ciphertext of each block as the IV of the next block.

# CFB Mode Decryption




Cipher Feedback (CFB) mode decryption

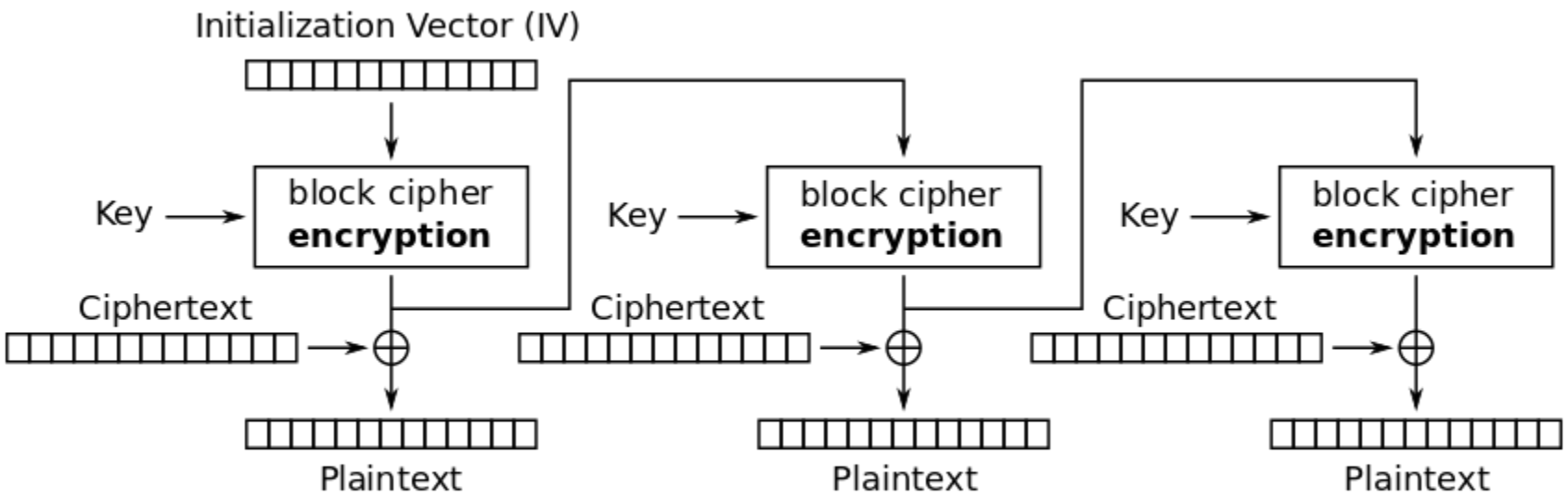
# Output Feedback (OFB) Mode




Output Feedback (OFB) mode encryption

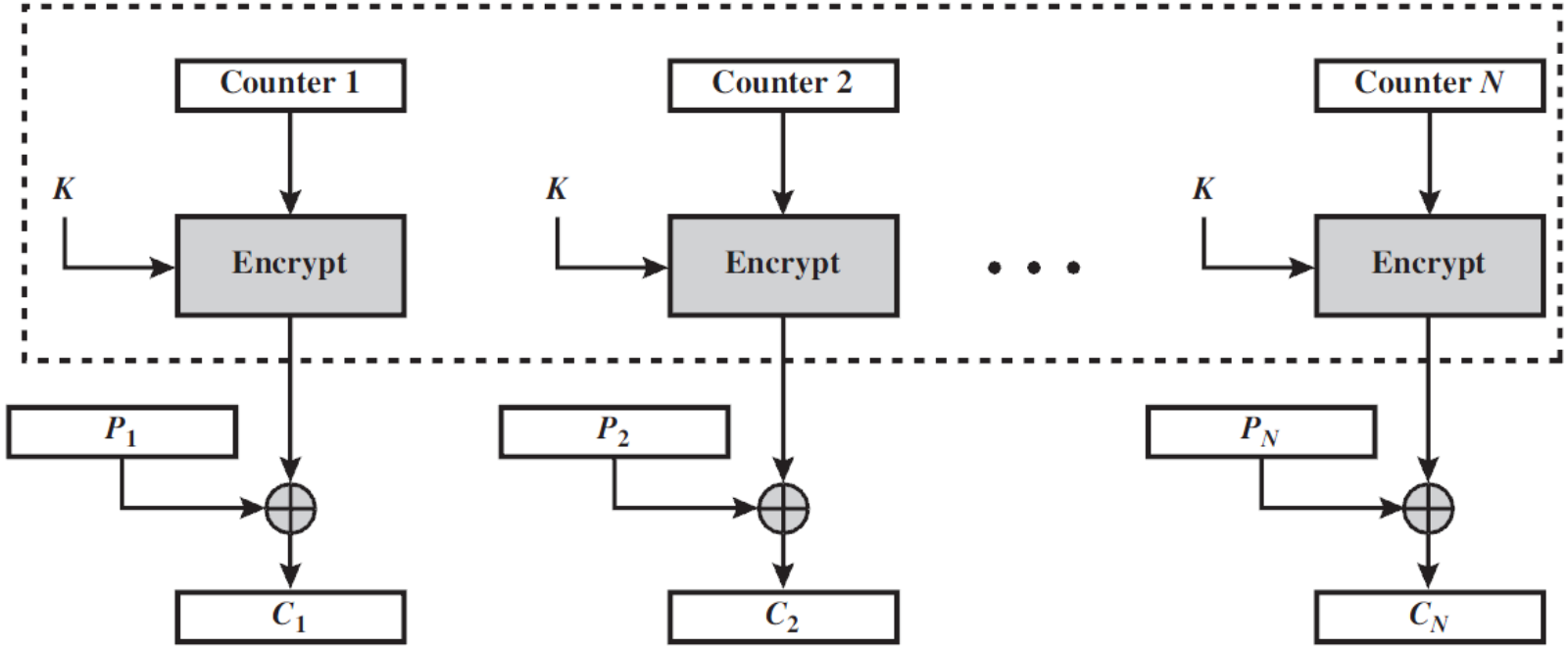
- IV is chosen uniformly at random
- This is built on the aforementioned randomized encryption II. But we use the blockcipher output of each block as the IV of the next block.
- It is possible to conduct pre-computation for encryption.

# OFB Mode Decryption



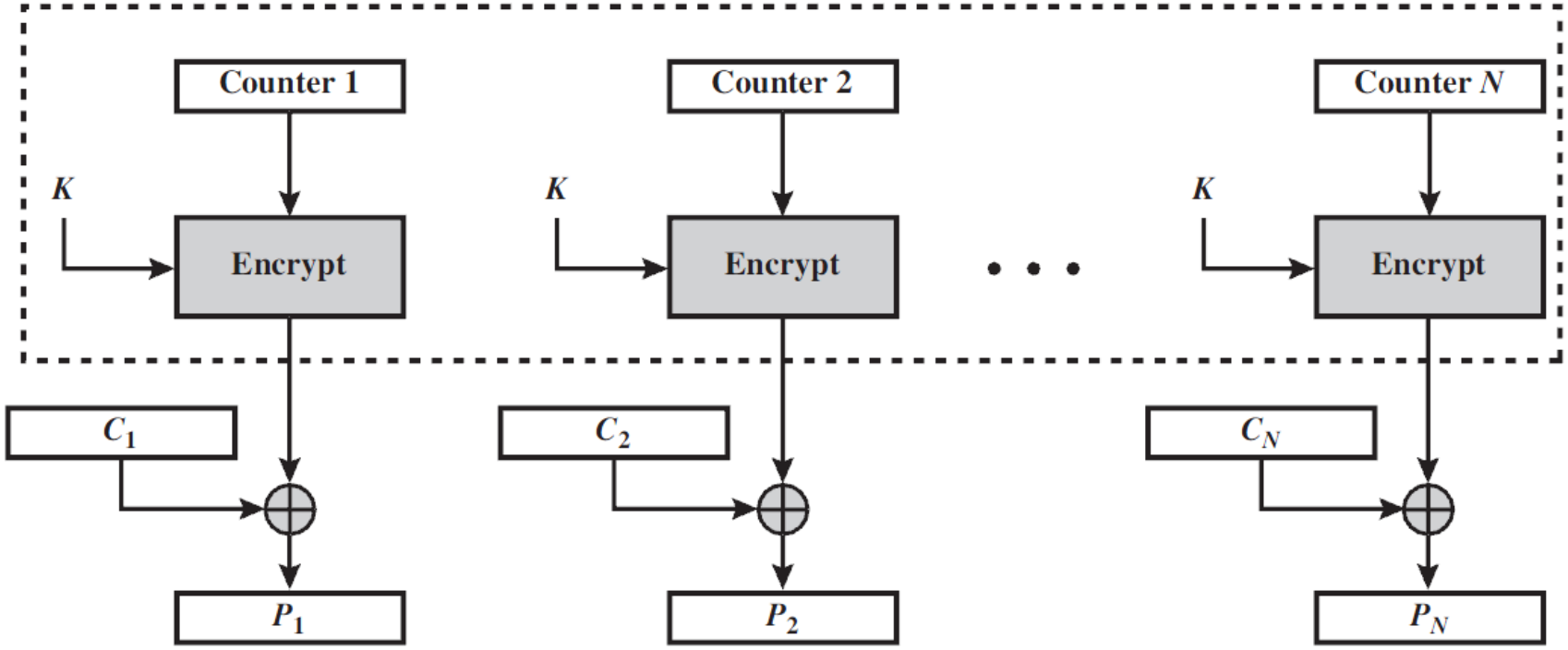

Output Feedback (OFB) mode decryption

# Counter (CTR) Mode



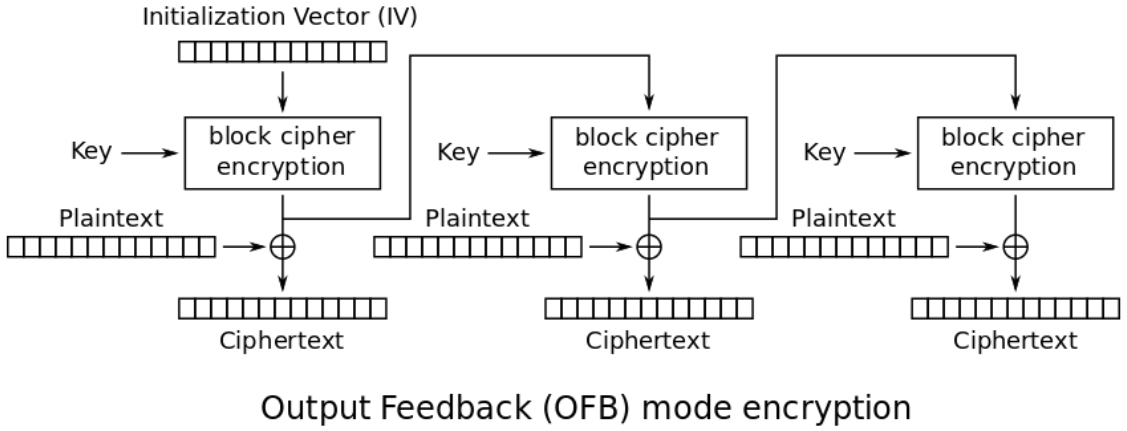

- Counter i is typically defined as IV || i, where IV is a randomly chosen initialization vector.
- This is built on the aforementioned randomized encryption II. But we use (non-uniform) and related IV for the blocks. This will not compromise security of the encryption scheme if the counters never repeat.
- Pre-computation for Encryption is possible.

# CTR Mode Decryption

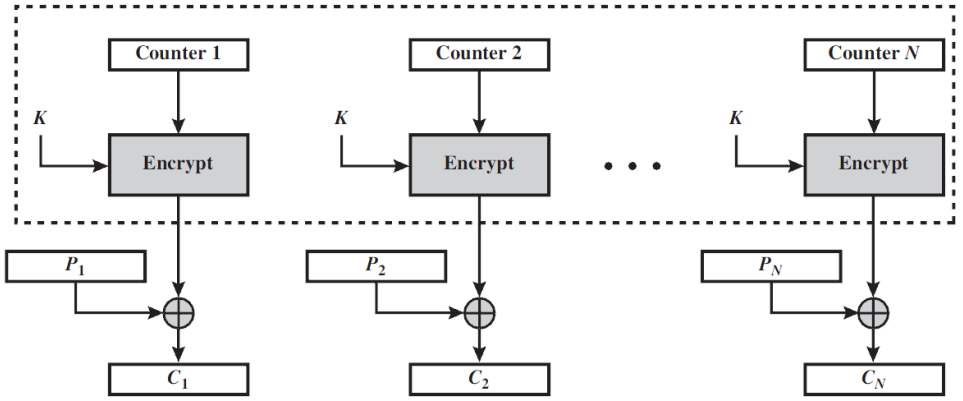



- Counter i is typically defined as IV || i, where IV is a randomly chosen initialization vector.
- This is built on the aforementioned randomized encryption II. But we use (non-uniform) and related IV for the blocks. This will not compromise security of the encryption scheme if the counters never repeat.
- Pre-computation for Encryption is possible.
- Both Encryption and Decryption can be done in parallel.
- Random access is supported (i.e., it is easy to decrypt any block without the need to decrypt other blocks).

# Stream cipher

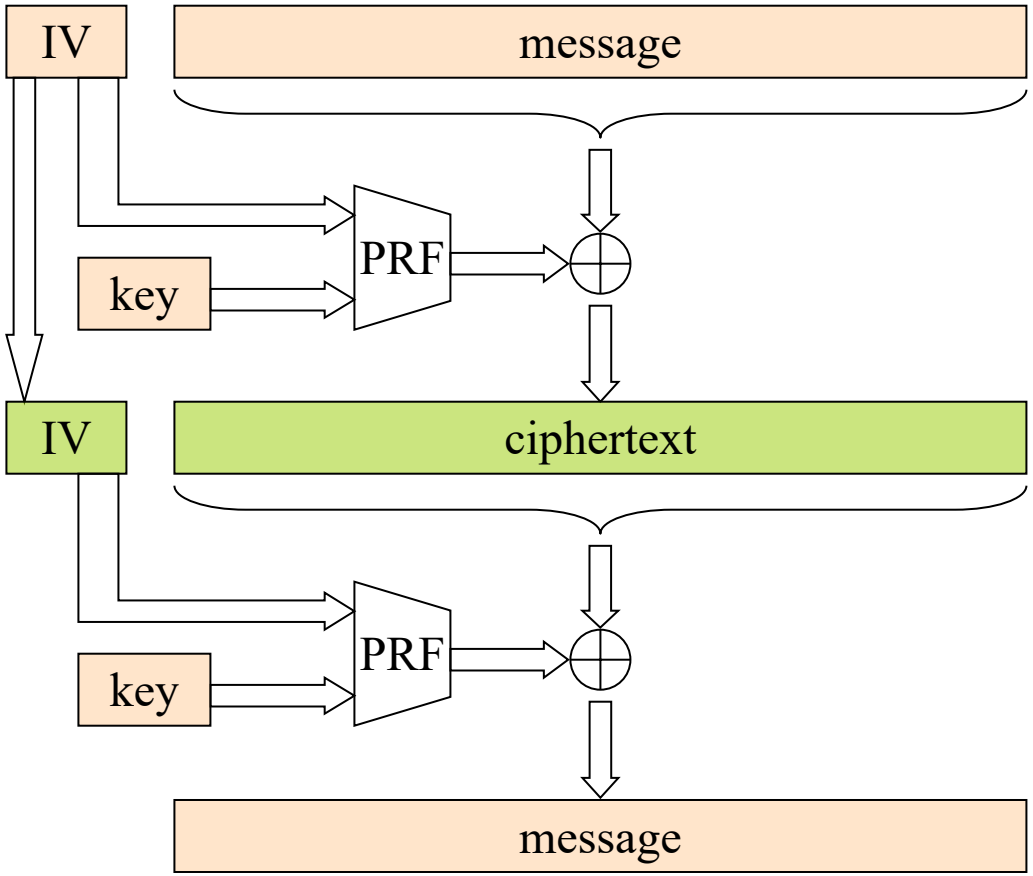


Output Feedback (OFB) mode encryption




- In the OFB mode and the CTR mode, the plaintext are xored with a key stream generated by the blockcipher, they are also called **stream ciphers**.
- A stream cipher simulates the one-time pad by xoring the message with a key stream of equal length. But the key stream is generated from a shorter secret key. In addition, the encryption is randomized since a random IV will be used in each encryption.
  - Is a stream cipher secure if we do not use the randomized IV?
- People design stream ciphers from sctratch in the beginning (e.g., RC4), but now, stream ciphers are usually constructed from a blockcipher via the OFB/CTR mode (e.g., Chacha 20).

# Stream cipher



# How to implement a SKE in practice

```
import os
from cryptography.hazmat.primitives.ciphers import
Cipher, algorithms, modes
from PIL import Image

# Read the plaintext
img = Image.open("plaintext.png")
data= img.tobytes()

# Generate the secret key
key = os.urandom(32)


# Encrypt using ECB
cipherE = Cipher(algorithms.AES(key), modes.ECB())
encryptorE = cipherE.encryptor()
ctE = encryptorE.update(data) + encryptorE.finalize()

# Write the ciphertext
imageE = Image.frombytes(img.mode, img.size, ctE)
imageE.save('ciphertextE.png')
```

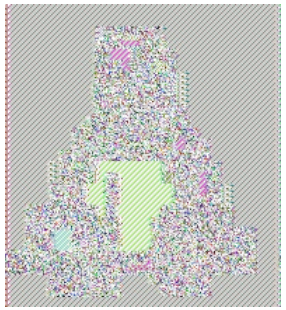
```
# Encrypt using CBC
iv = os.urandom(16)
cipherC = Cipher(algorithms.AES(key), modes.CBC(iv))
encryptorC = cipherC.encryptor()
ctC = encryptorC.update(data) + encryptorC.finalize()

# Write the ciphertext
imageC = Image.frombytes(img.mode, img.size, ctC)
imageC.save('ciphertextC.png’)

# Decrypt using ECB
decryptorE = cipherE.decryptor()
pE=decryptorE.update(ctE) + decryptorE.finalize()
imagepE = Image.frombytes(img.mode, img.size, pE)
imagepE.save('pE.png’)

# Decrypt using CBC
decryptorC = cipherC.decryptor()
pC=decryptorC.update(ctC) + decryptorC.finalize()
imagepC = Image.frombytes(img.mode, img.size, pC)
imagepC.save('pC.png')
```

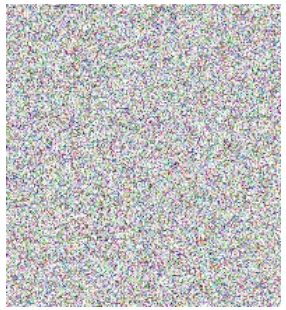
The codes are implemented using the pyca/cryptography library (https://cryptography.io/).

- This is a python library depending on OpenSSL

# How to implement a SKE in practice

```
import os
from cryptography.hazmat.primitives.ciphers import
Cipher, algorithms, modes
from PIL import Image

# Read
img = I
data= i

# Gener
key = c


# Encry
cipherE
encrypt
ctE = e

# Write
imageE = Image.frombytes(img.mode, img.size, ctE)
imageE.save('ciphertextE.png')
```



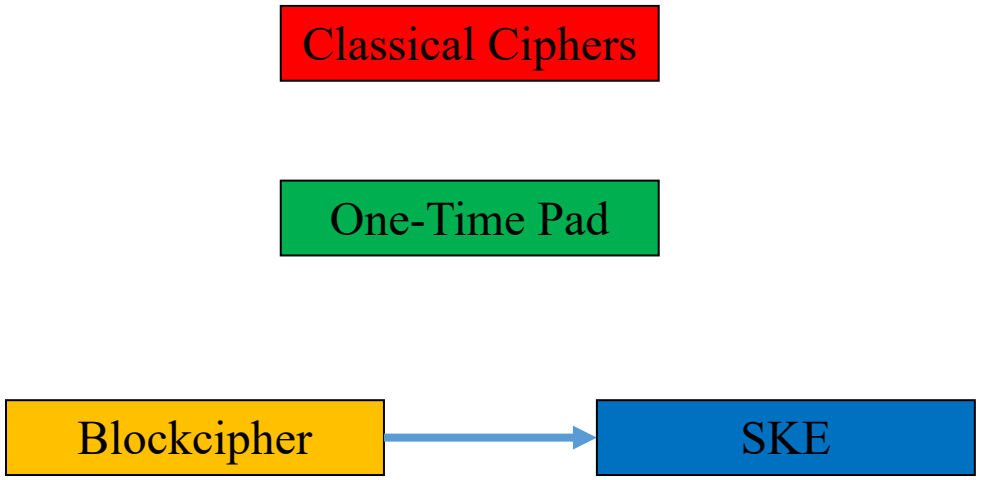


```
# Encrypt using CBC
iv = os.urandom(16)
cipherC = Cipher(algorithms.AES(key), modes.CBC(iv))
encryptorC = cipherC.encryptor()
                                        ryptorC.finalize()

                                        img.size, ctC)

                                        orE.finalize()
                                        , img.size, pE)

pC=decryptorC.update(ctC) + decryptorC.finalize()
imagepC = Image.frombytes(img.mode, img.size, pC)
imagepC.save('pC.png')
```

The codes are implemented using the pyca/cryptography library (https://cryptography.io/).

- This is a python library depending on OpenSSL

# Roadmap



# Summary

- Blockcipher
  - What is a blockcipher
  - Principles of designing blockciphers
  - DES Cipher
  - The Feistel network
    - How to encrypt
    - How to decrypt
  - Security of DES
  - AES
- Secure SKE
  - Why blockcipher is not secure
  - Definition of SKE
  - Construction of encryption scheme
    - Constructions from blockciphers
    - Why they are secure*
  - Operation Modes
    - ECB (not secure)
    - CBC
    - CFB
    - OFB
    - CTR
  - Stream Cipher