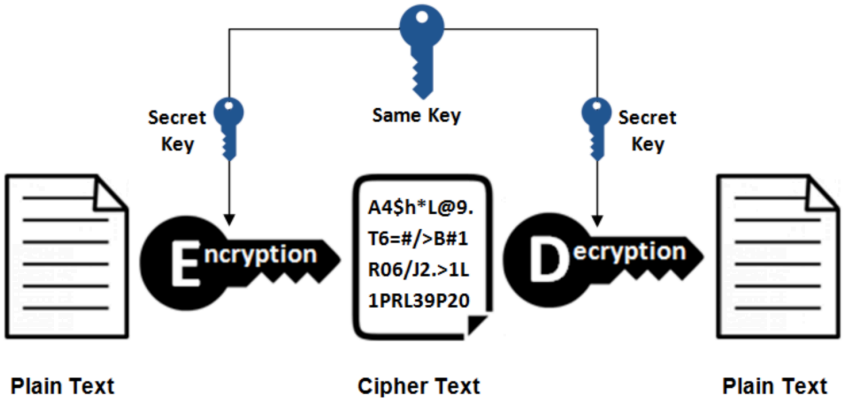
Secret Key Cryptography



Tecnologie Internet

a.a. 2022/2023

Summary

- Cryptography
- Cryptanalysis
- Computational and unconditional security
- Classical encryption techniques
- Block and stream ciphers
- Feistel cipher
- DES
- Double DES
- 3DES
- AES
- Encrypting large messages
- Main security uses of secret key cryptography
- Disadvantages of secret key cryptography

Cryptography

Greek: kryptós+graphein: hidden/secret + writing

Study of mathematical techniques related to information security in the presence of third party adversaries.

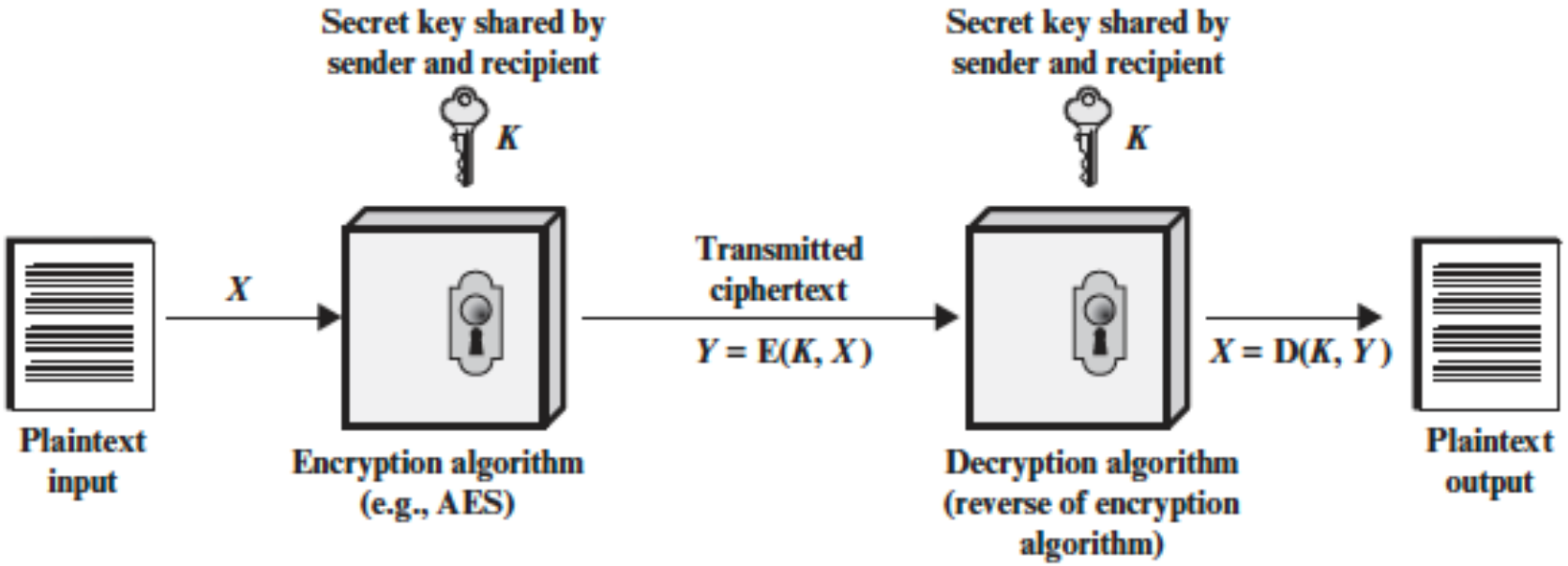
An original message is known as the **plaintext**, while the coded message is called the **ciphertext**.

The process of converting from plaintext to ciphertext is known as **enciphering** or **encryption**; restoring the plaintext from the ciphertext is **deciphering** or **decryption**.

A scheme used for encryption is known as a **cryptographic system** or a **cipher**.

Cryptography

Example: **secret key cryptography (symmetric encryption)**



The **secret key** is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time.

Cryptography

Cryptographic systems are characterized by three independent dimensions:

1) the **type of reversible operations used** for transforming plaintext to ciphertext (substitutions and transpositions)

2) the **number of keys used** (secret key cryptography vs. public key cryptography)

3) the **way in which the plaintext is processed** (block cipher vs. stream cipher)

Note: we do not have to keep the algorithm secret, we need only to keep the key secret.

Cryptanalysis

Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**.

Attacks **exploit the characteristics of the algorithm** to attempt to deduce a specific plaintext or to deduce the key being used.

In a **brute force attack** the attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained.

A **side channel attack** is any attack based on information gained from the physical implementation of a cryptosystem, rather than theoretical weaknesses in the algorithms.

Ciphertext only attack

The opponent has seen (and presumably stored) some ciphertext that can be analyzed, and he/she should be able to recognize when he/she has succeeded (often called **recognizable plaintext attack**).

It is the hardest attack to carry on, as it is necessary to have enough ciphertext and the opponent has the least amount of information to work with.

Known plaintext attack

The opponent knows a <plaintext, ciphertext> pair.
From that pairs, the attacker can try to figure out the mapping of some fraction of the text.

How is it possible to obtain the plaintext?

- the secret data does not remain secret forever
- the opponent may have knowledge of what is in the message
    - certain key words (e.g. in the header of the message)
    - expected patterns (e.g. PostScript, etc.)
    - probable-word attack

Chosen plaintext/ciphertext attack

The opponent can choose any plaintext and get the corresponding ciphertext from the system (or the contrary).

E.g., there is a transmission service that encrypts and transmits messages; the attacker can ask the transmission service to transmit any plaintext he/she wants.

Computational and unconditional security

An encryption scheme is said to be **computationally secure** if either of the following two criteria are met.

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available.

Classical encryption techniques

Substitution techniques:
- Caesar cipher
- Monoalphabetic ciphers
- Multiletter ciphers
- Polyalphabetic ciphers
- One-time pad

Transposition techniques:
- Row transposition ciphers

Product ciphers

Caesar cipher

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

plaintext: meet me after the toga party
ciphertext: PHHW PH DIWHU WKH WRJD SDUWB

A **shift** may be of any amount, so the general algorithm is:

$$C = E(k,p) = (p+k) \bmod 26$$
$$p = D(k,C) = (C-k) \bmod 26$$

where *26* is the number of characters in the English alphabet, and *k* (the shift, i.e., the key) takes values in *{1,…,25}*.

Caesar cipher

Cryptanalysis of the Caesar cipher is simple.

1. The encryption and decription algorithms are known.
2. There are only 25 keys to try (brute force attack is feasible).
3. The language of the plaintext is known and easily recognizable.

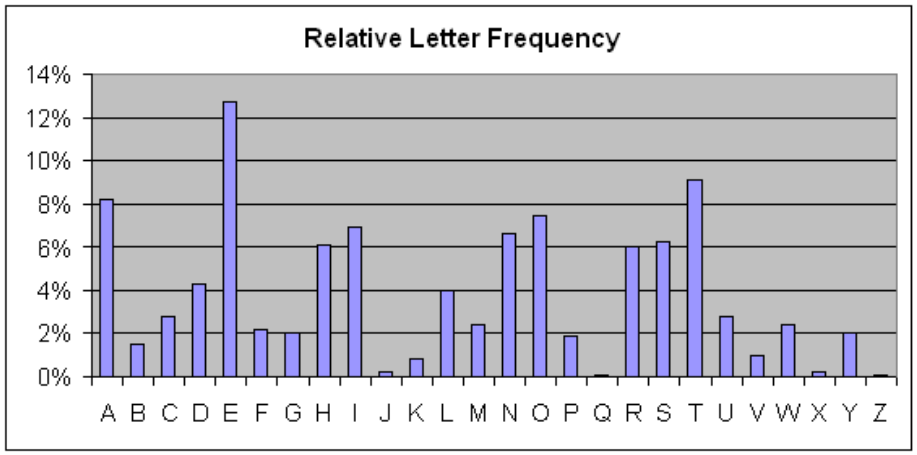
Monoalphabetic ciphers

Based on **permutations** of the *26* alphabetic characters.

There are $26! > 4 \times 10^{26}$ possible keys.

However, the opponent has another line of attack: exploiting the regularities of the language.

- Letter frequency
- Frequency of two-letter combinations (digrams)



Multiletter ciphers

The best-known multiletter cipher is the **Playfair**, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

The Playfair algorithm is based on the use of a *5x5* matrix of letters constructed using a keyword.

Encryption rules

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as X, so that **balloon** would be treated as **BA LX LO ON**.
2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, **ar** is encrypted as **RM**.
3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, **mu** is encrypted as **CM**.
4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, **hs** becomes **BP** and **ea** becomes **IM** (or JM, as the encipherer wishes).

Polyalphabetic ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message.

All polyalphabetic techniques have the following features in common:

1. a set of related monoalphabetic substitution rules is used

2. a key determines which particular rule is chosen for a given transformation

Polyalphabetic ciphers

The best-known polyalphabetic cipher is the **Vigenère cipher**.

key: $k = \{k_0,k_1,\ldots,k_{m-1}\}$
plaintext: $p = \{p_0,p_1,\ldots,p_{n-1}\}$
ciphertext: $C = \{C_0,C_1,\ldots,C_{n-1}\}$

$$C_i = (p_i + k_{i \bmod m}) \bmod 26$$
$$p_i = (C_i - k_{i \bmod m}) \bmod 26$$

Example:

key: deceptive ($m = 9$)
plaintext: wearediscoveredsaveyourself
ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Polyalphabetic ciphers

With the Vigenère cipher, the letter frequency information is obscured, but not all knowledge of the plaintext is lost.

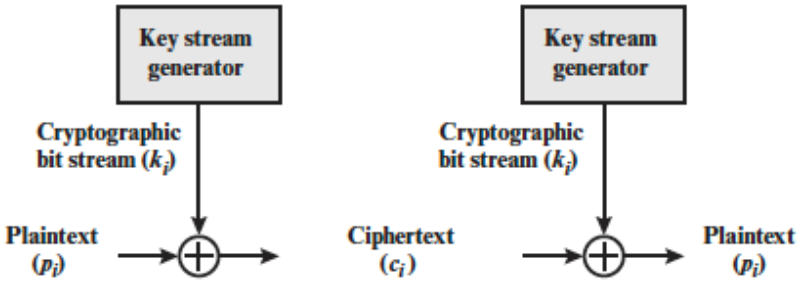
The length of the key can be determined using the following insights.

If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length, they will generate identical ciphertext sequences.

Moreover, if the key length is *m*, then the cipher, in effect, consists of *m* monoalphabetic substitution ciphers. For example, with the keyword "deceptive", the letters in positions 1, 10, 19, and so on are all encrypted with the same monoalphabetic cipher. Thus, we can use the known frequency characteristics of the plaintext language to attack each of the monoalphabetic ciphers separately.

One-time pad

- **Vernam cipher**
  - the key $k=\{k_0,k_1,\ldots,k_{n-1}\}$ is as long as the plaintext
  - ciphertext contains no statistical relationship to the plaintext
  - In case of alphabet {0,1}:
    - $C_i = p_i \oplus k_i$
    - $p_i = C_i \oplus k_i$



- If a new truly random key k is used for each message we obtain the so called **One-Time Pad (OTP)** cipher
  - no statistical relationship between distinct ciphertexts
  - the cipher will be unconditionally secure (unbreakable)

- disadvantages:
  - huge plaintext needs huge key
  - safe distribution of key is complicated

Row transposition ciphers

Write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of the columns then becomes the key to the algorithm.

For example,

```
Key:          4 3 1 2 5 6 7
Plaintext:    a t t a c k p
              o s t p o n e
              d u n t i l t
              w o a m x y z
Ciphertext:   TTNAAPTMTSUOAODWCOIXKNLYPETZ
```

The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed.

Product ciphers

Ciphers using substitutions or transpositions may be not sufficiently secure.

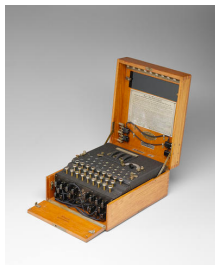
Hence consider using several ciphers in succession to make harder:

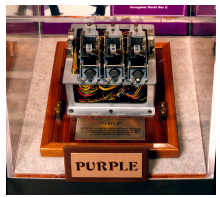
- two substitutions make a more complex substitution
- two transpositions make more complex transposition
- substitution followed by a transposition makes a new much harder cipher

This is the bridge from classical to modern ciphers.

Before modern ciphers, **rotor machines** were the most common product ciphers. During World War 2:

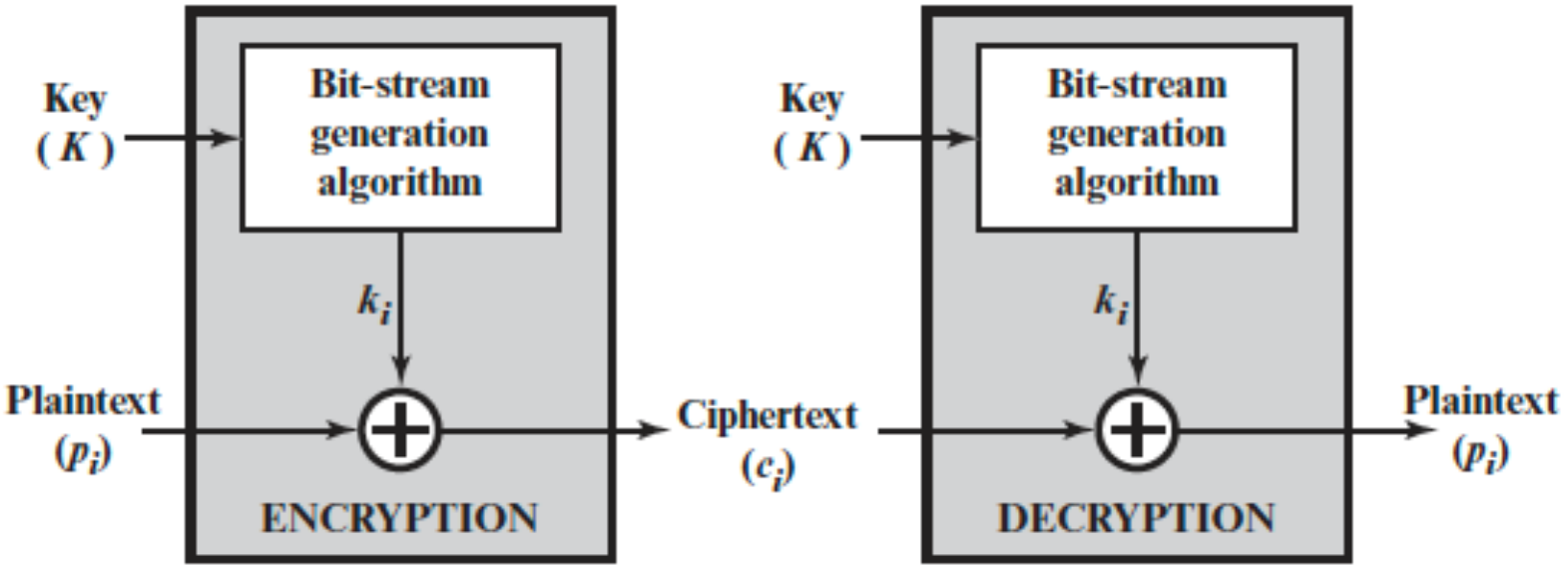
- Enigma (Germany)
- Hagelin (Alliance)
- Purple (Japan)





Block and stream ciphers

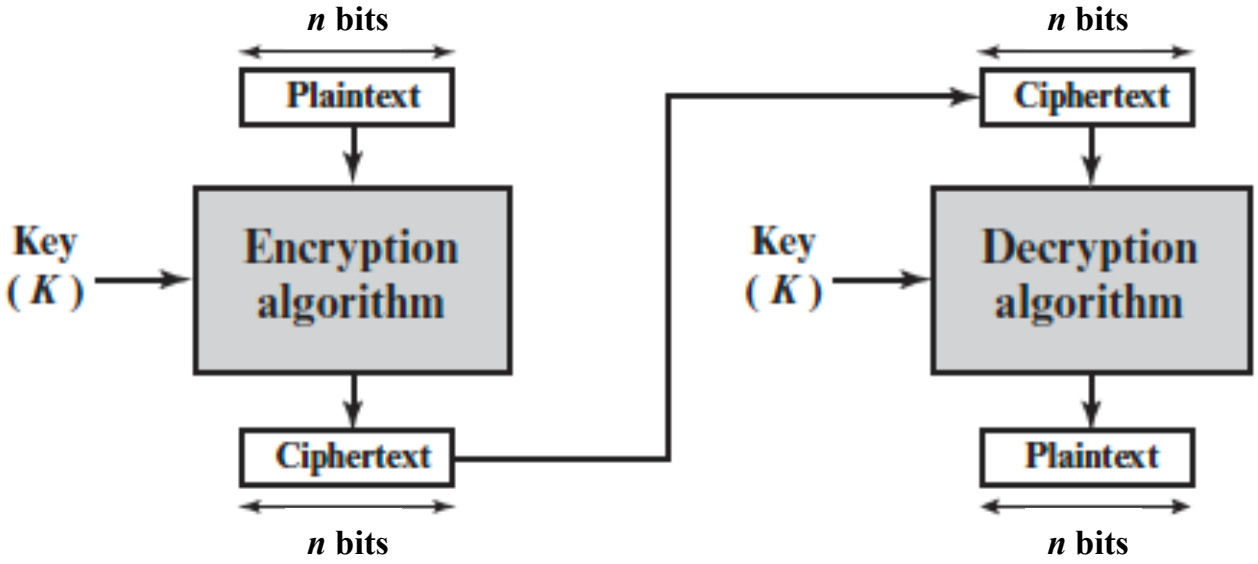
A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time.



Block and stream ciphers

A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

The vast majority of network-based symmetric cryptographic applications make use of block ciphers.



Ideal block cipher

A block cipher operates on a plaintext block of *n* bits to produce a ciphertext block of *n* bits.

There are 2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block.

If we limit ourselves to reversible mappings, the number of different transformations is $2^n!$ (recall permutations).

Reversible Mapping	
Plaintext	**Ciphertext**
00	11
01	10
10	00
11	01

Irreversible Mapping	
Plaintext	**Ciphertext**
00	11
01	10
10	01
11	01

Ideal block cipher

An arbitrary reversible substitution cipher (the **ideal block cipher**) for a large block size is not practical, from an implementation and performance point of view.

For such a transformation, the mapping itself constitutes the key.

In general, for an *n*-bit ideal block cipher, the length of the key defined in this fashion is $n2^n$ bits.

For a *64*-bit block, which is a desirable length to thwart statistical attacks, the required key length is $64*2^{64} = 2^{70} \approx 10^{21}$ bits.

Feistel cipher

Feistel proposed to approximate the ideal block cipher by utilizing the concept of a **product cipher**, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers.

The essence of the approach is to develop a block cipher with a **key length of *k* bits** and a **block length of *n* bits**, allowing a total of 2^k possible transformations, rather than the $2^n!$ transformations available with the ideal block cipher.

Feistel cipher

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:

- **Substitution**: Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.

- **Permutation**: A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

Feistel cipher

In fact, Feistel's is a practical application of a proposal by **Claude Shannon** to develop a product cipher that alternates diffusion and confusion functions.

In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits.

For example, message $M = m_1,m_2,m_3,\ldots$ could be encrypted with an averaging operation: $y_i = (m_{i+1} + m_{i+2} + \ldots + m_{i+k}) \bmod 26$

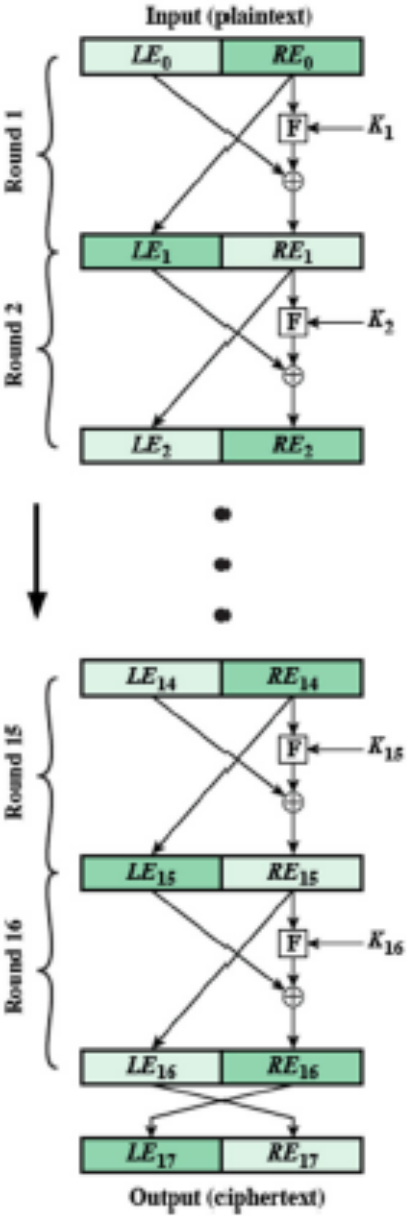
On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, by means of a complex substitution algorithm.

Feistel cipher

The inputs to the **encryption algorithm** are a plaintext block of length $n=2w$ bits and a key K.

The plaintext block is divided into two halves, LE_0 and RE_0. The two halves of the data pass through several rounds of processing (all rounds have the same structure: substitution + permutation) and then combine to produce the ciphertext block.

Each round i has as inputs LE_{i-1} and RE_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys K_i are different from K and from each other.

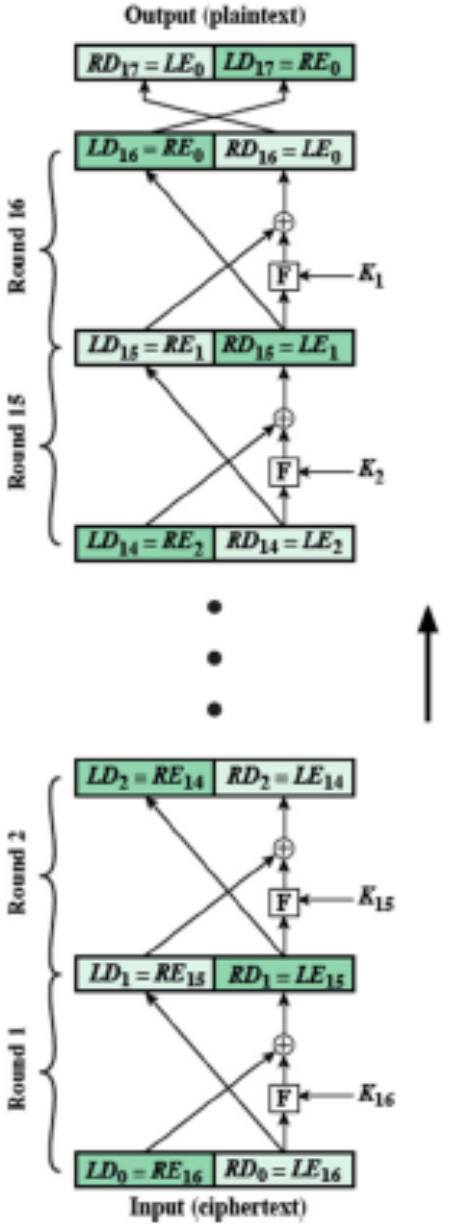


Feistel cipher

The process of **decryption** with a Feistel cipher is essentially the same as the encryption process.

The rule is as follows: use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order.

This is a nice feature, because it means we need not implement two different algorithms, one for encryption and one for decryption.
Instead, one algorithm implementation is enough.



Feistel cipher

Parameters and design features:

- a larger **block size** means greater security; typical values are 64 and 128 bits
- larger **key size** means greater security but may decrease encryption/decryption speed; 128 bits is a common size
- the greater the **number of rounds**, the more difficult it is to perform cryptanalysis; a typical size is 16
- greater complexity in the **subkey generation algorithm** should lead to greater difficulty in cryptanalysis
- greater complexity in the **round function F** means greater resistance to cryptanalysis

DES

DES (Data Encryption Standard) was issued in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST).

The algorithm itself is denoted as **DEA (Data Encryption Algorithm)**, which is a Feistel cipher with:

- 64-bit blocks
- 56-bit keys

In 1999, NIST released a new version of the DES document indicating **TDEA (Triple DEA)** as the algorithm to be used.

Note: TDEA is mostly known as **3DES**, thus we will use 3DES.

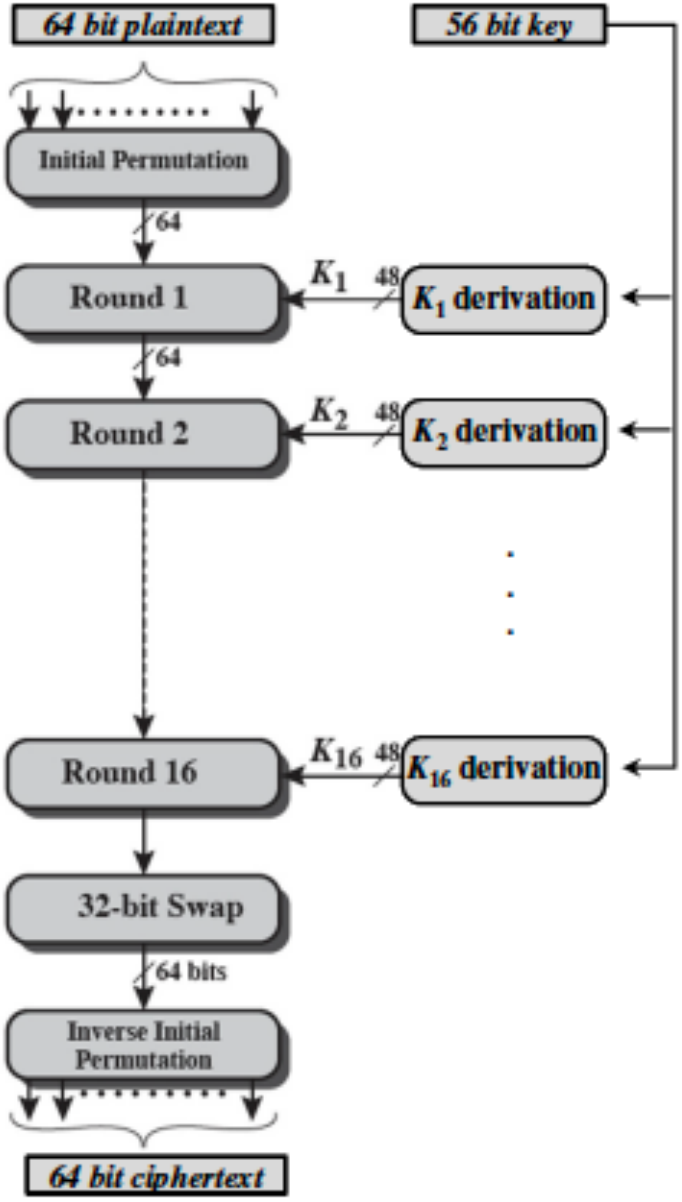
DES

Encryption

- Key transformation
  - the 56-bit key is transformed into 16 48-bit subkeys (one per round)
- An initial permutation (IP)
- 16 identical "rounds" of operation where the data is confused and diffused with the key
- A final permutation (IP^{-1})

Avalanche effect: a small change in either the plaintext or the key produces a significant change in the ciphertext.

Decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.



DES

DES strengths:

- 56-bit keys make a brute-force attack impractical (but not impossible, with modern GPU-based HPC systems; for this reason, 3DES or AES with 128-bit keys are better)
- No one has so far succeeded in discovering fatal weaknesses in the encryption/decryption algorithm
- Robust to **timing attacks**, i.e., those in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts

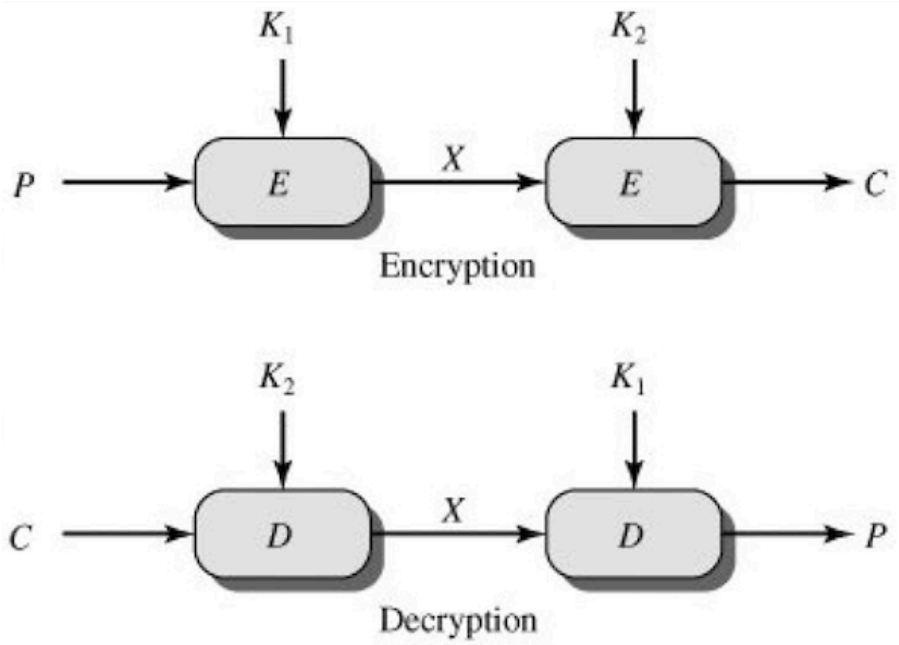
Double DES

Given a plaintext P and two keys K_1 and K_2, ciphertext C is generated as

$C = E(K_2, E(K_1, P))$

Decryption requires that the keys be applied in reverse order:

$P = D(K_1, D(K_2, C))$



Double DES

Consider that encryption with DES is a mapping of 64-bit blocks to 64-bit blocks. In fact, the mapping can be viewed as a **permutation**.

With 2^{64} possible inputs, there are $(2^{64})!$ different mappings that generate a permutation of the input block.

On the other hand, DES defines one mapping for each different key, for a total of 2^{56} mappings.

Therefore, it is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that are not defined by a single application of DES.

In other words, **it is unlikely to find K_3 such that**

$$\mathbf{E}(K_2,\mathbf{E}(K_1,P)) = \mathbf{E}(K_3,P)$$

Double DES

However, there is a way to attack Double DES. The algorithm, known as **meet-in-the-middle attack**, is based on the observation that, if we have $C = E(K_2,E(K_1,P))$, then

$$X = E(K_1,P) = D(K_2,C)$$

Given a known (P,C) pair:

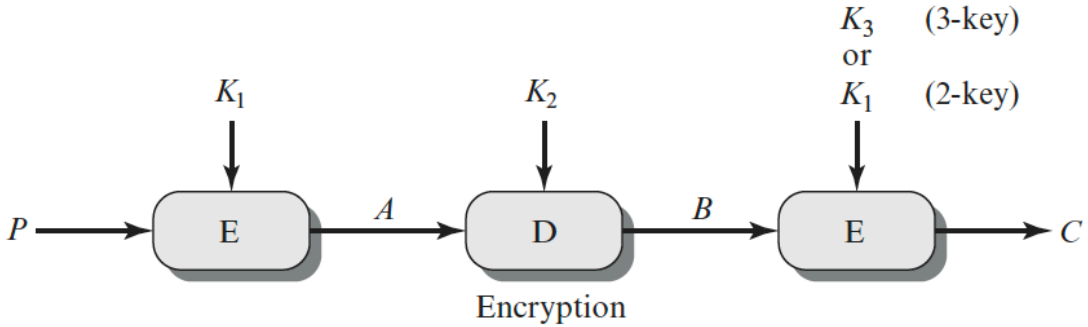
1) encrypt P for all possible values of K_1; store the results in a table and sort them by X
2) decrypt C by using all possible values of K_2; check each result against the table for a match; if a match occurs, test the two keys against a new known (P,C) pair; if they work, then stop

Computational cost of the attack: $\boldsymbol{O(2^{57})}$ **E and D operations**
Not much more than the computational cost of the brute force attack against single DES.

3DES

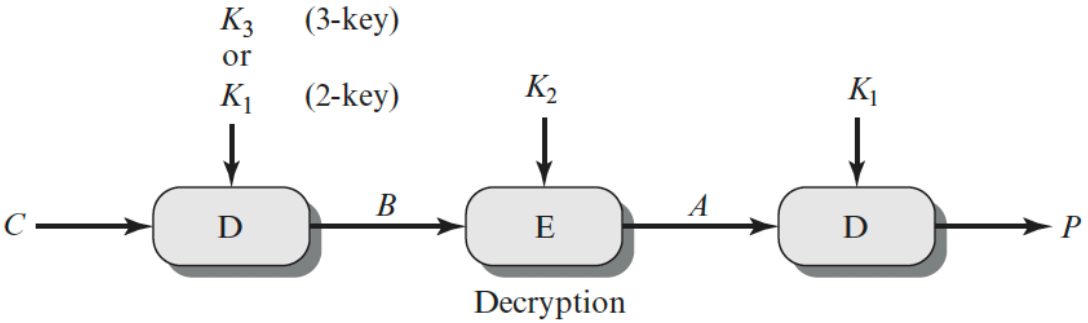
With two keys:

$$C = E(K_1, D(K_2, E(K_1, P)))$$
$$P = D(K_1, E(K_2, D(K_1, C)))$$



With three keys:

$$C = E(K_3, D(K_2, E(K_1, P)))$$
$$P = D(K_1, E(K_2, D(K_3, C)))$$



In SP 800-57, Part 1 (Recommendation for Key Management—Part 1: General, July 2012) NIST recommends that 2-key 3DES be retired and replaced with 3-key 3DES.

AES

AES (Advanced Encryption Standard) was published by NIST in 2001.

It is also known as Rijndael block cipher
- original name of the algorithm submitted to AES selection process
- developed by Joan Daemen and Vincent Rijmen (Belgium)

Adopted as an encryption standard by the U.S. government.

Currently, one of the most popular algorithms used in symmetric key cryptography.

Its structure is quite complex.

General structure

Plaintext block size: 128 bits.

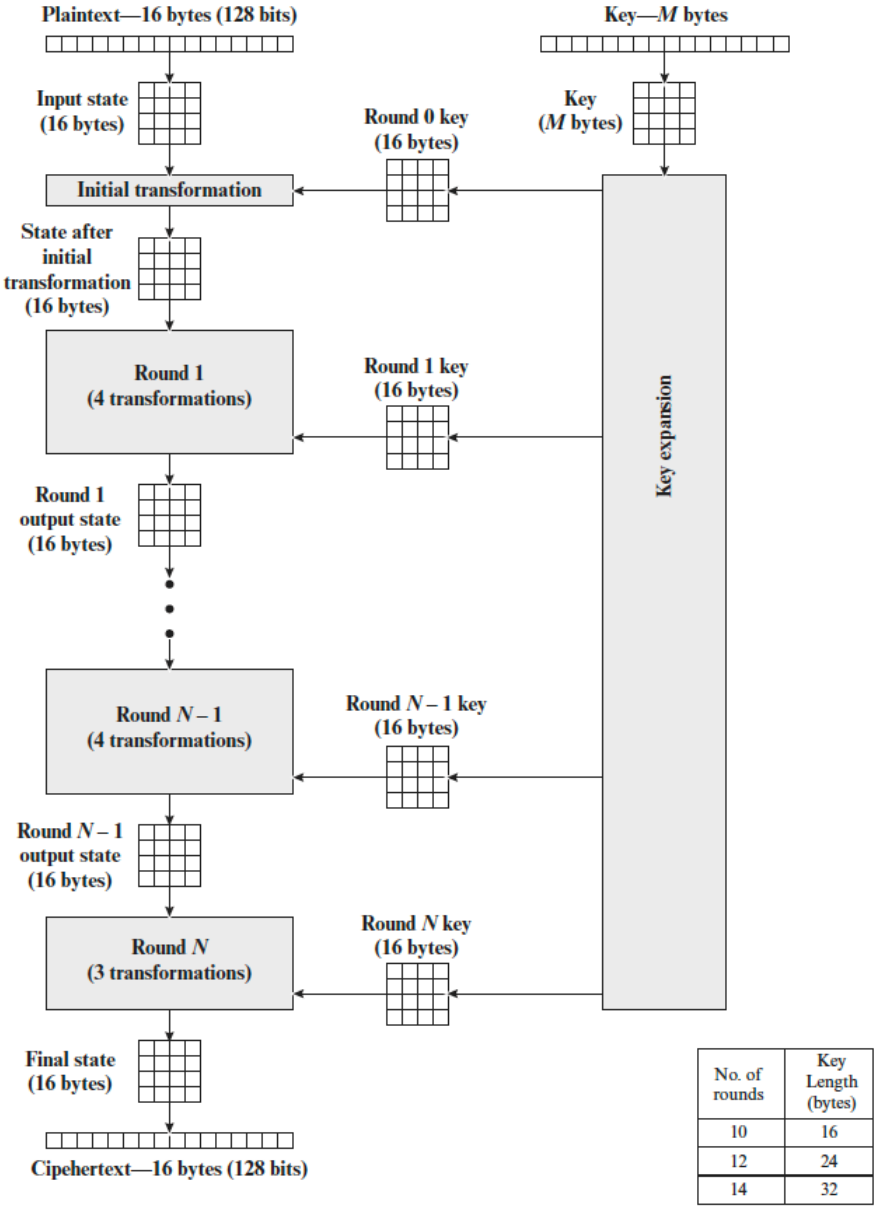
Block depicted as 4x4 matrix of bytes.

Block copied to **State** array, which is modified at each stage of encryption or decryption.

Key length: 128, 192, or 256 bits (AES-128, AES-192, **AES-256**).

recommended

The key is expanded into an array of words, each one being 4 bytes (44 words for the 128-bit key).



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

General structure

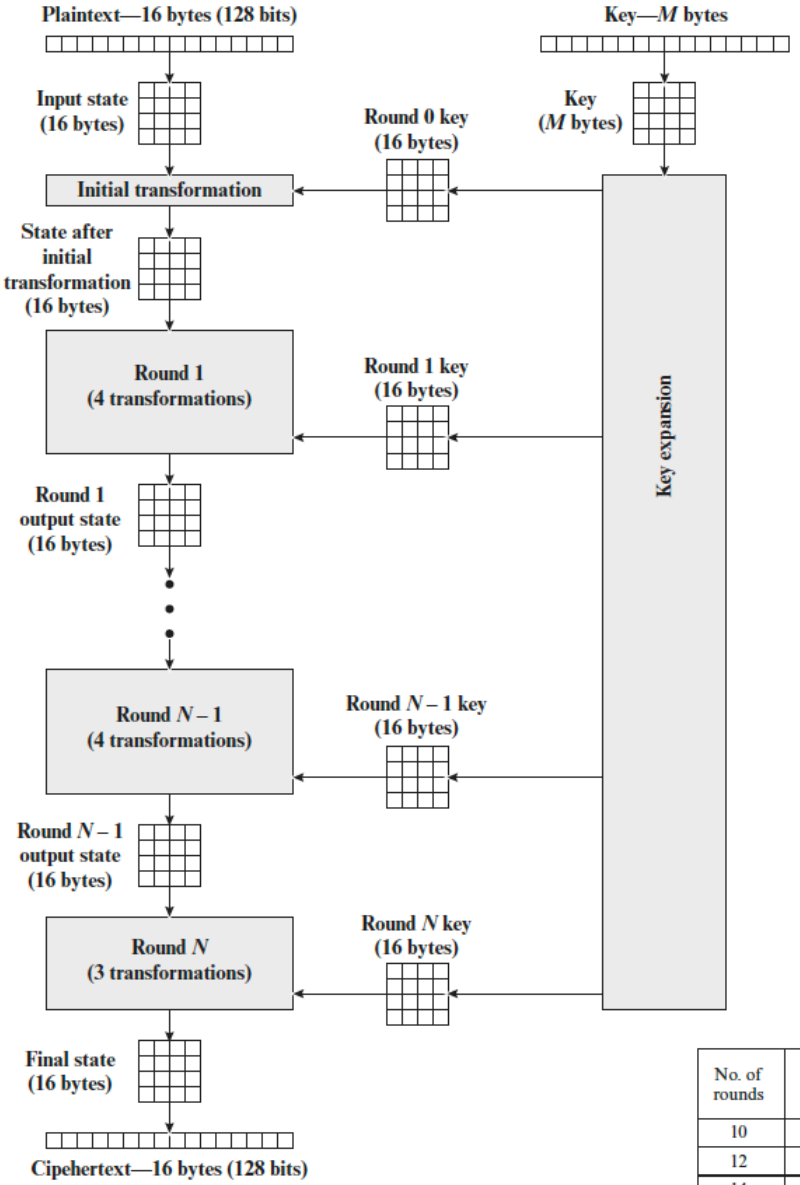
The cipher consists of *N* rounds.

N depends on the key length:
- 10 rounds for 16-byte key
- 12 rounds for 24-byte key
- 14 rounds for 32-byte key

First *N*-1 rounds: 4 transformations
- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

Initial transformation: AddRoundKey

N-th round: 3 transformations



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Detailed structure

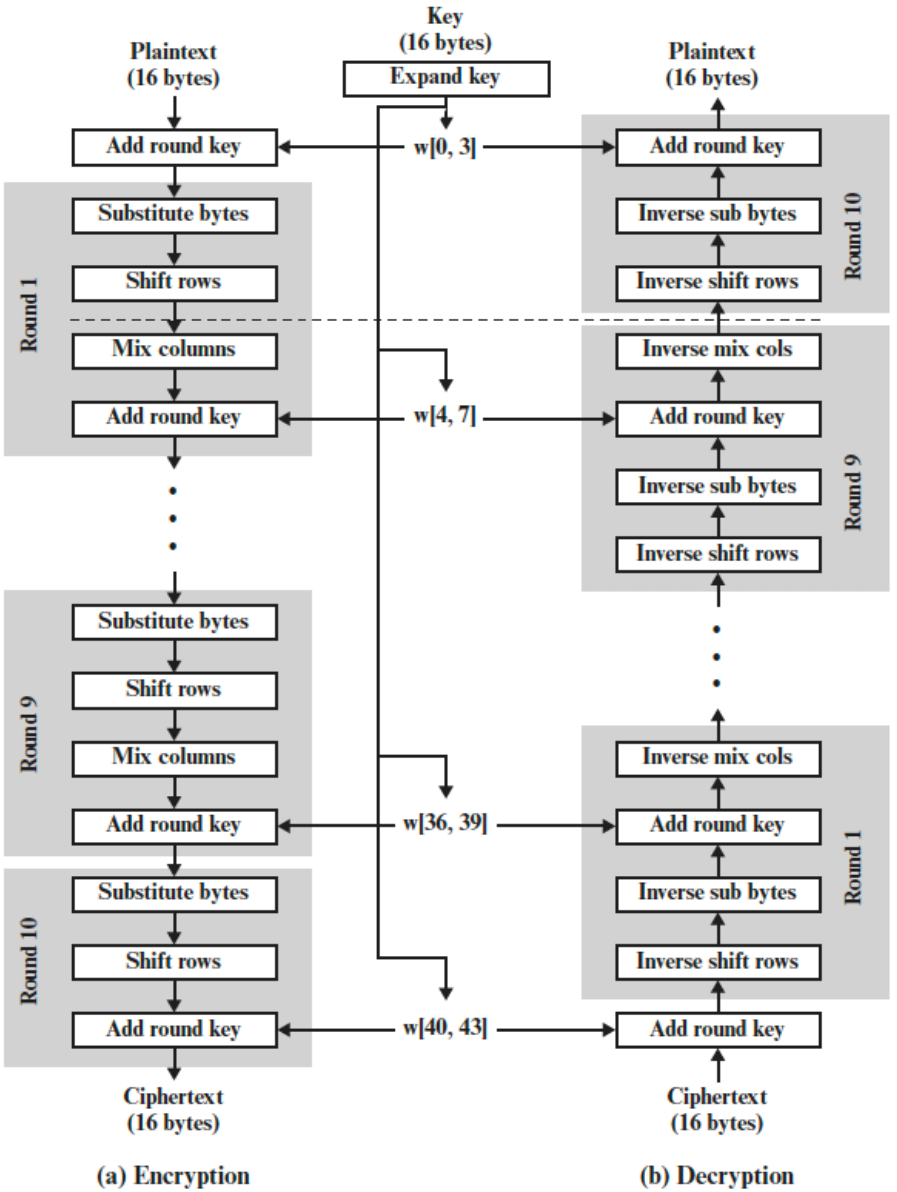
Unlike Feistel ciphers, AES processes the entire data block as a single matrix during each round.

SubBytes: uses an S-box to perform a byte-by-byte substitution of the block

ShiftRows: a simple permutation

MixColumns: a substitution that makes use of arithmetic over GF(2^8)

AddRoundKey: bitwise XOR of current block with a portion of the expanded key



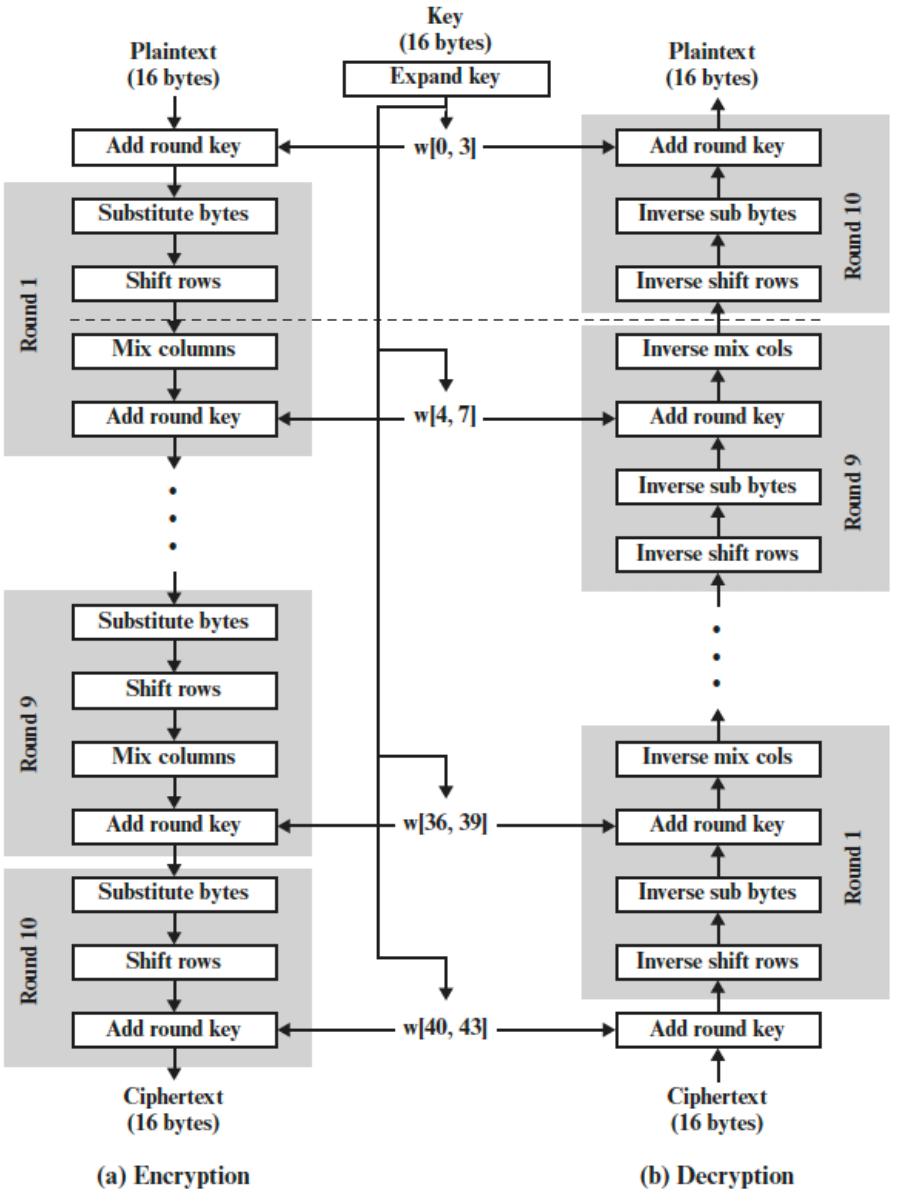
Detailed structure

Encryption and decription have the same simple structure, but the decryption algorithm is not identical to the encryption algorithm.

Only AddRoundKey makes use of the key. It is a form of Vernam cipher. The other stages provide:

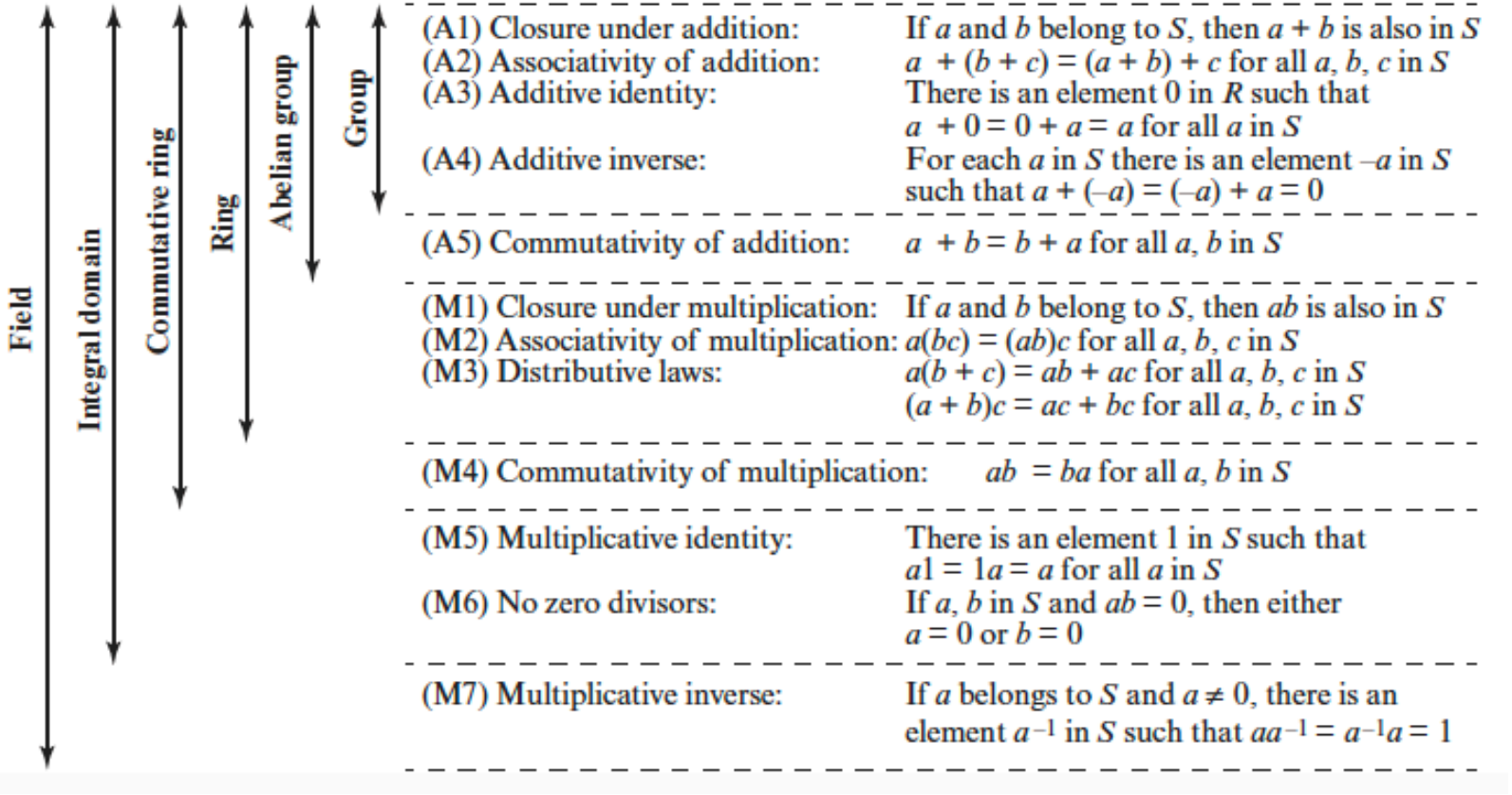
- confusion
- diffusion
- nonlinearity

All four stages are easily reversible. At each horizontal point in the figure, **State** is the same for both encryption and decryption.



Finite field arithmetic

A field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set.



Finite field arithmetic

A **finite field** (or Galois field) is a field with a finite number of elements (the **order** of the field).

A finite field with order m only exists if and only if $m = p^n$, for some positive integer n and prime integer p.

The finite field with order p^n is denoted as **GF(p^n)**, GF standing for Galois field.

Its elements may be represented as polynomials of degree $n-1$ with coefficients in {0,1,..,p-1} (i.e., polynomials of degree $n-1$ over GF(p)):

$$A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$

In the following, we consider **GF(2^n)**, thus the coefficients of the polynomials are binary.

Finite field arithmetic

In GF(2^n), addition is performed by taking the bitwise XOR of the two n-bit strings of coefficients:

$$A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$
$$B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_1x + b_0$$

$$C = A + B = (c_{n-1},c_{n-2},\ldots,c_1,c_0) \text{ where } c_i = a_i \oplus b_i$$

Multiplication is performed modulo an **irreducible polynomial** over GF(2):

$$C(x) = A(x)B(x) \bmod P(x)$$

A polynomial f(x) over a field GF is called irreducible if and only if f(x) cannot be expressed as a product of two polynomials, both over GF, and both of degree lower than that of f(x).

The inverse of a nonzero element is defined as:

$$A^{-1}(x) \text{ s.t. } A^{-1}(x)A(x) \bmod P(x) = 1$$

$A^{-1}(x)$ can be calculated by means of the Extended Euclid Algorithm

Finite field arithmetic

AES operates on 8-bit bytes.

All operations are performed in **GF(2^8)** with the irreducible polynomial

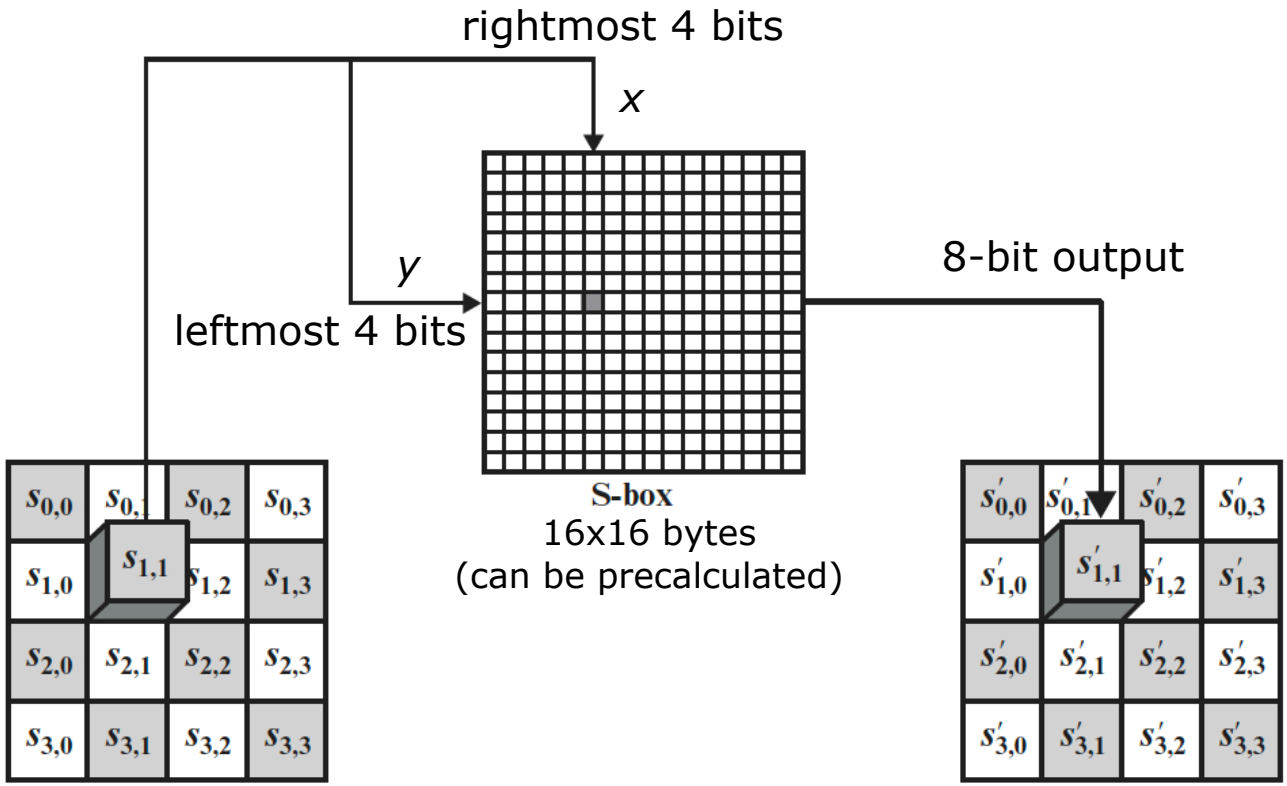
$$m(x) = x^8 + x^4 + x^3 + x + 1$$

There are 30 possible irreducible polynomials of degree 8. Apparently, this one was chosen because it is the first one on the list given in the following book:

Lidl, R., and Niederreiter, H. Introduction to Finite Fields and Their Applications. Cambridge: Cambridge University Press, 1994.

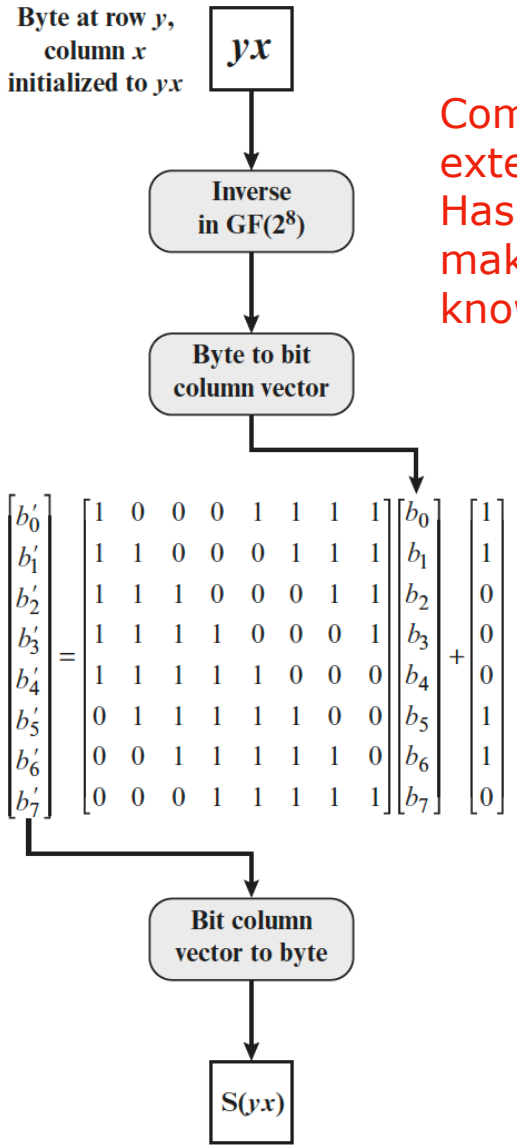
SubBytes

The **forward substitute byte transformation** (SubBytes) is a simple table lookup. Each byte of **State** is mapped into a new byte.



SubBytes

Construction of the S-box:



$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Computed by means of the extended Euclidean algorithm. Has good nonlinearity properties, making the S-box resistant to known cryptanalytic attacks.

Attention! Each element in the product vector is the bitwise XOR of products of elements of one row and one column. Also, the final addition is a bitwise XOR.

Example: {95} —> {2A}

InvSubBytes

The **inverse substitute byte transformation** (InvSubBytes) is a table lookup that makes use of the inverse S-box (IS-box).

Example: {2A} —> {95}

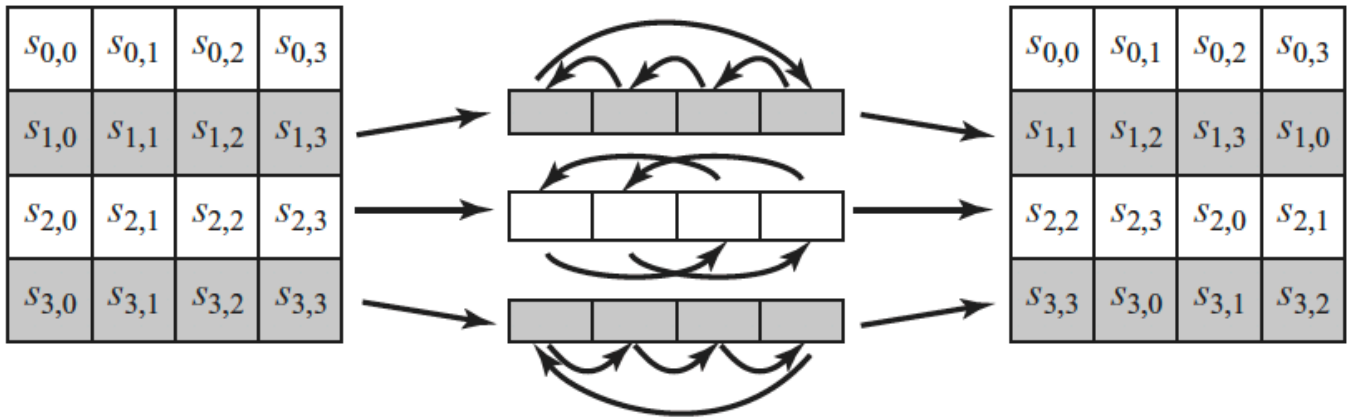
Note:

- IS-box[S-box(*a*)] = *a*
- S-box(*a*) ≠ IS-box(*a*)

Example:
S-box({95}) = {2A}
IS-box({95}) = {AD}

ShiftRows

The **forward shift row transformation** (ShiftRows) performs different circular shifts on the second, third and fourth row of **State**.



InvShiftRows

The **inverse shift row transformation** (InvShiftRows) performs the circular shifts in the opposite direction for each of the last three rows.

Since

- the plaintext is copied to **State** in order to fill it column by column
- AddRoundKey operates on **State** column by column

it follows that the shift row transformation is quite substantial, as it ensures that the 4 bytes of one column are spread out to four different columns.

MixColumns

The **forward mix column transformation** (MixColumns) operates on each column individually.

Each byte of a column is mapped into a new value that is a function of all four bytes in that column.

The transformation is defined by the following matrix multiplication on **State**:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Additions and multiplications are performed in GF(2^8).

InvMixColumns

The **inverse mix column transformation** (InvMixColumns) is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

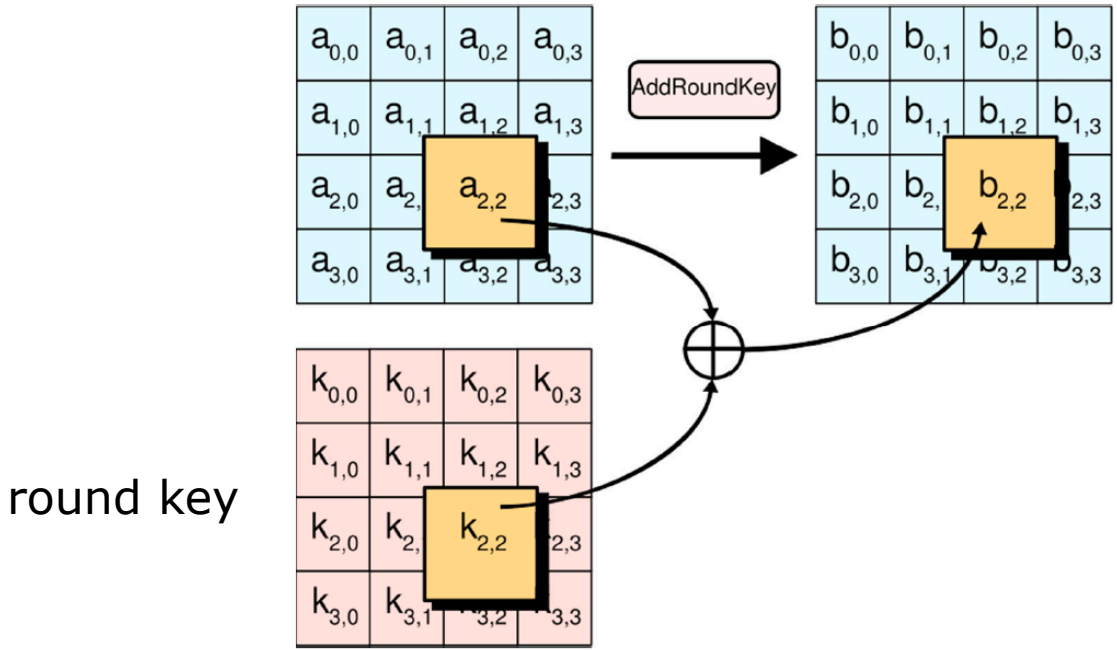
Additions and multiplications are performed in GF(2^8).

The coefficients of the MixColumns matrix are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column.

AddRoundKey

In the **forward add round key transformation** (AddRoundKey), the 128 bits of **State** are bitwise XORed with the 128 bits of the round key.

The **inverse add round key transformation** is identical to the forward one, because the XOR operation is its own inverse.



AES key expansion

In AES-128, the key expansion algorithm takes as input the 128-bit (16-byte) key and produces a linear array of 44 words:

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++) w[i] = (key[4*i], key[4*i+1],
                                    key[4*i+2],
                                    key[4*i+3]);
    for (i = 4; i < 44; i++)
    {
    temp = w[i - 1];
    if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                            ⊕ Rcon[i/4];
    w[i] = w[i-4] ⊕ temp
    }
}
```

AES key expansion

1. **RotWord** performs a one-byte circular left shift on a word.

2. **SubWord** performs a byte substitution on each byte of its input word, using the S-box.

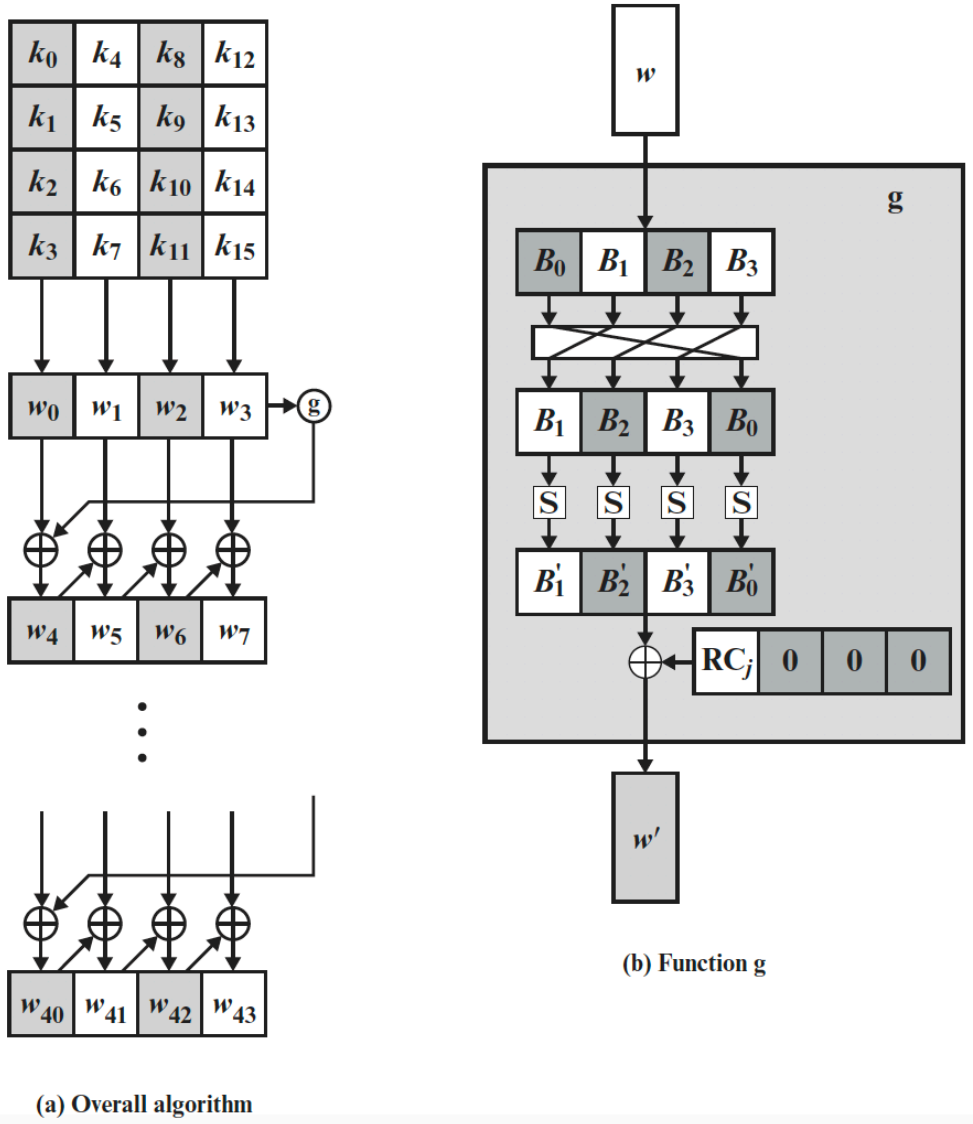
3. The result of steps 1 and 2 is XORed with a round constant, **Rcon**[j].

The round constant is different for each round:

Rcon[j] = (RC[j],0,0,0) with RC[1] = 1, RC[j] = 2RC[j-1]

where the multiplication is defined in GF(2^8).

AES key expansion



(a) Overall algorithm

(b) Function g

AES security

Brute force attack:

- 128-bit key —> $2^{128} = 3.4 \times 10^{38}$ possible keys
- 192-bit key —> $2^{192} = 3.4 \times 10^{57}$ possible keys
- 256-bit key —> $2^{256} = 3.4 \times 10^{77}$ possible keys

Comparing to DES, if you could crack a DES key in one second (i.e., try 2^{56} keys per second), it would take 149 trillion years to crack a 128-bit AES key by brute force at the same speed.

In June 2003, the US Government announced that AES may be used also for classified information.

In 2011 the first key-recovery attacks on full AES were published [Bogdanov2011], improving on brute force by a factor of about four.

Encrypting large messages

Usually we have an arbitrary amount of data to encrypt (longer than the block size *b*).

In 2011, NIST has published five modes:

- **Electronic Code Book (ECB)**
- **Cipher Block Chaining (CBC)**
- **Output Feedback (OFB)**
- **Cipher Feedback (CFB)**
- **Counter Mode (CM)**

Encrypting large messages

If the encryption mode requires that the length of the whole message has to be a multiple of the block size, a **padding** operation has to be performed first.

Bit padding (e.g., ISO/IEC 7816-4): symbol 1 is added, and then as many 0 as required are added.

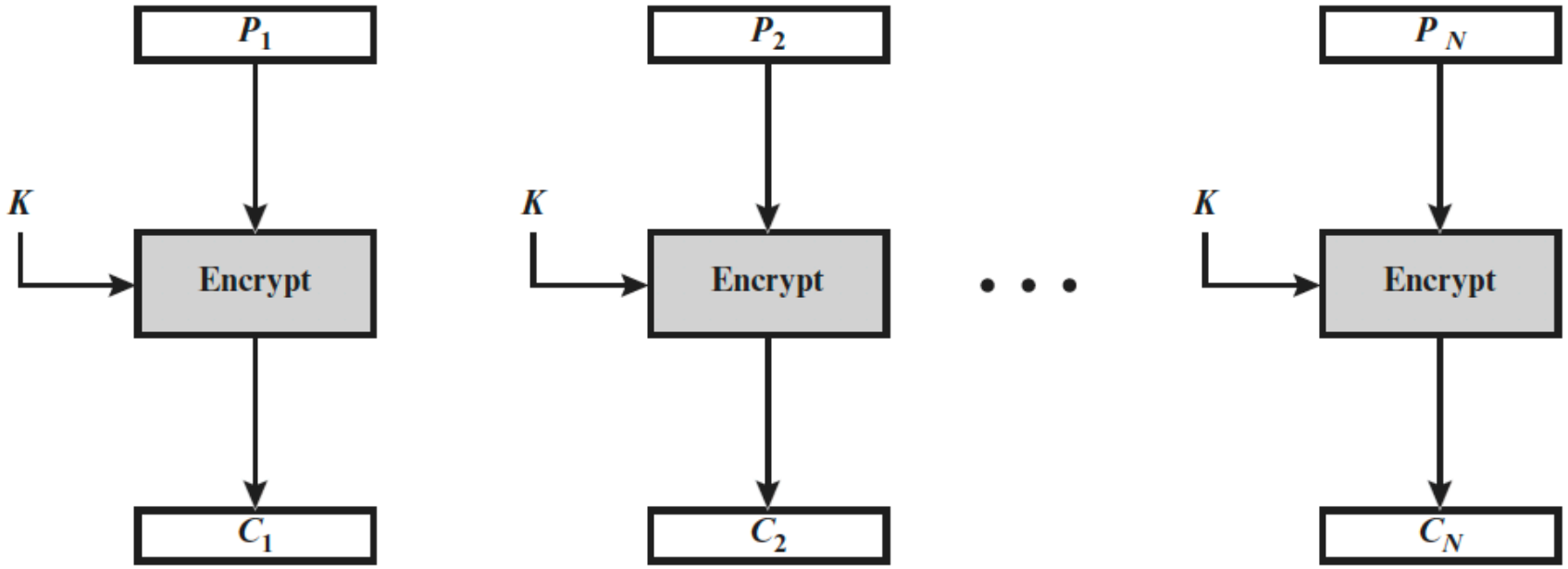
ANSI X.923: the last byte defines the number of padding bytes; the remaining bytes are filled with zeros.
Example:
[b1 b2 b3 00 00 00 00 05] (3 data bytes, then 5 bytes pad+count)

RFC 5652: the value of each added byte is the number of bytes that are added. Example: [b1 b2 b3 05 05 05 05 05]

Electronic Code Book (ECB)

The message is broken into *n* blocks of size *b* with padding for the last one.

Each block is independently encrypted with the secret key K.



Electronic Code Book (ECB)

Advantages:

- ECB is very simple and does not introduce extra operation (except for padding)

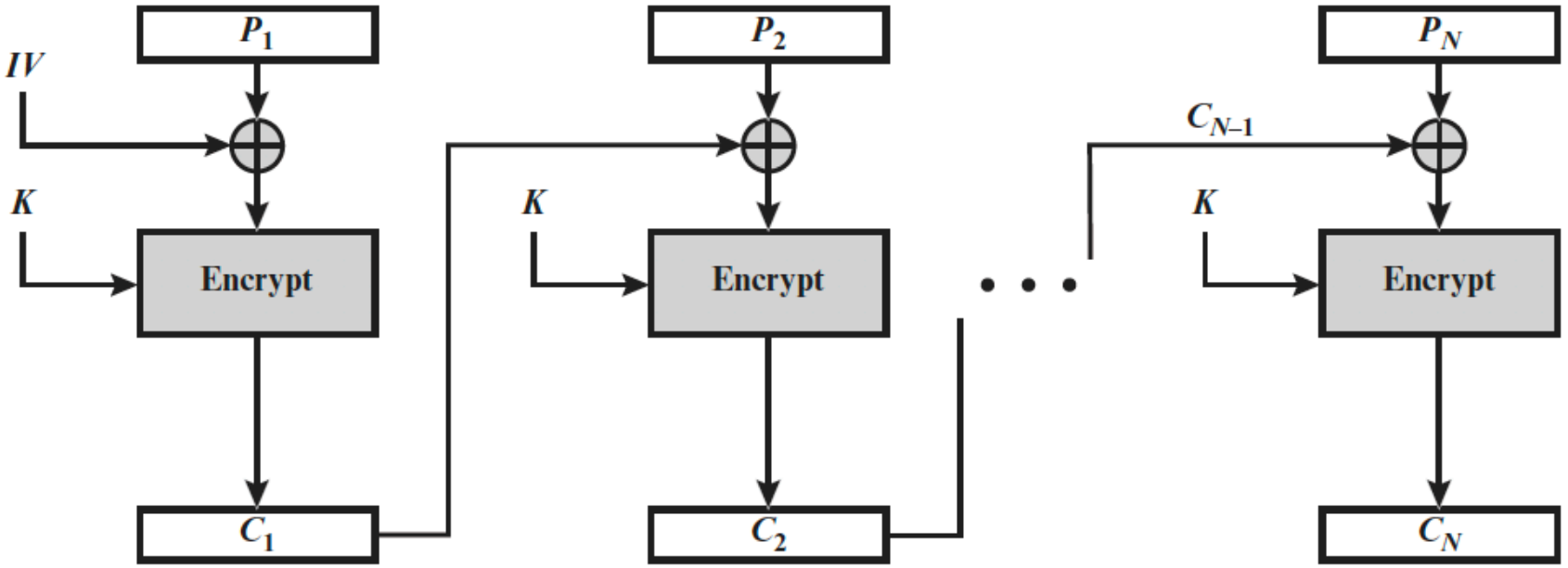
Limitations:

- repetitions in message may show in ciphertext if aligned with message block
- by comparing two ciphertexts it is possible to discover similarities in the plaintexts
- (partially) knowing the plaintext, is possible to rearrange the ciphertext blocks in order to obtain a new (known) plaintext

Therefore, ECB is rarely used. Its main use is for sending a few blocks of data (e.g., a secret key).

Cipher Block Chaining (CBC)

If the same block repeats in the plain text, it will not cause repeats of ciphertext.



IV = Initialization Vector (b-bit pseudorandom number)

Cipher Block Chaining (CBC)

Advantages:

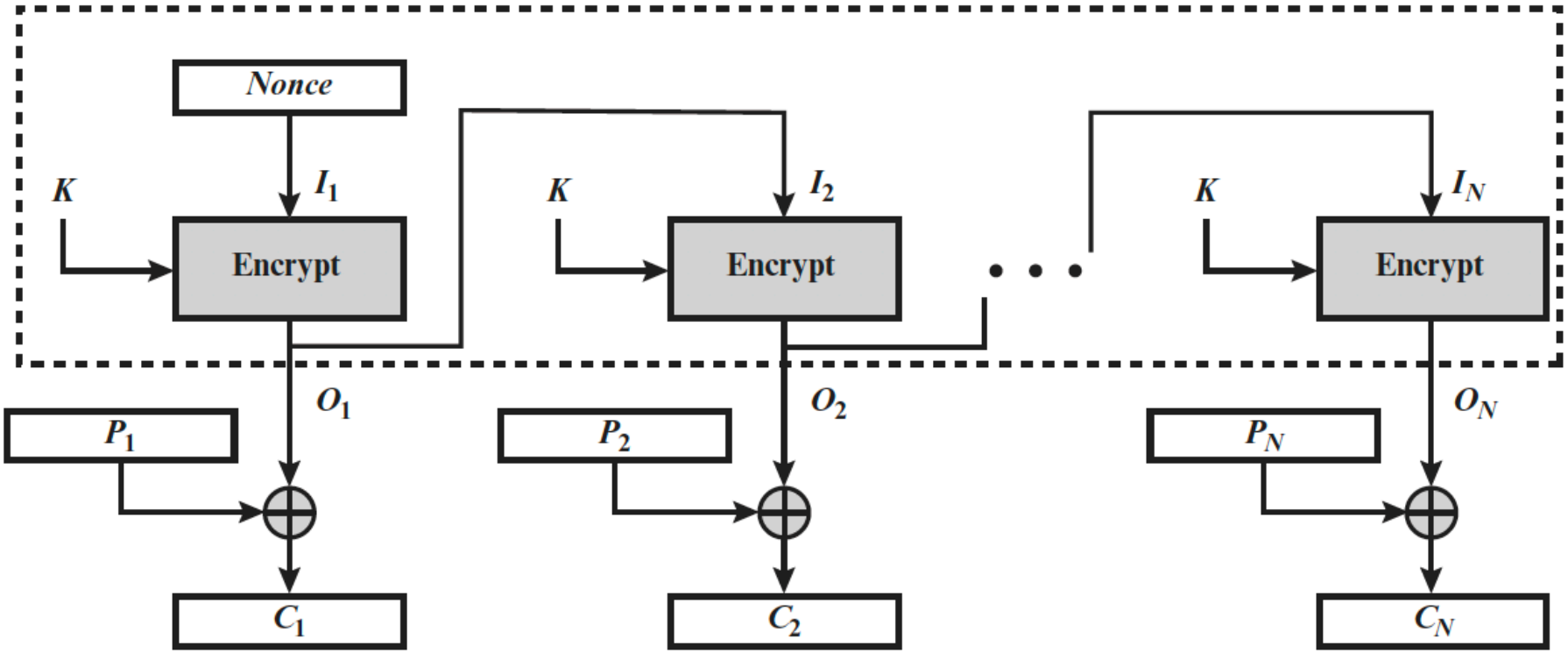
- decryption is simple because XOR is reversible
- each ciphertext block depends on all previous message blocks
- Has the same performance of ECB, except for the cost of generating and transmitting the IV using ECB (and the cost of one XOR)

Limitations:

- using a constant value as IV may lead to some problems (if the value is known, attackers may supply chosen plaintext)

Output Feedback (OFB)

As with CBC, the ciphertext of any plaintext unit is a function of the preceding plaintext.



Nonce = *b*-bit pseudorandom number (like IV in CBC)

Output Feedback (OFB)

Advantages:

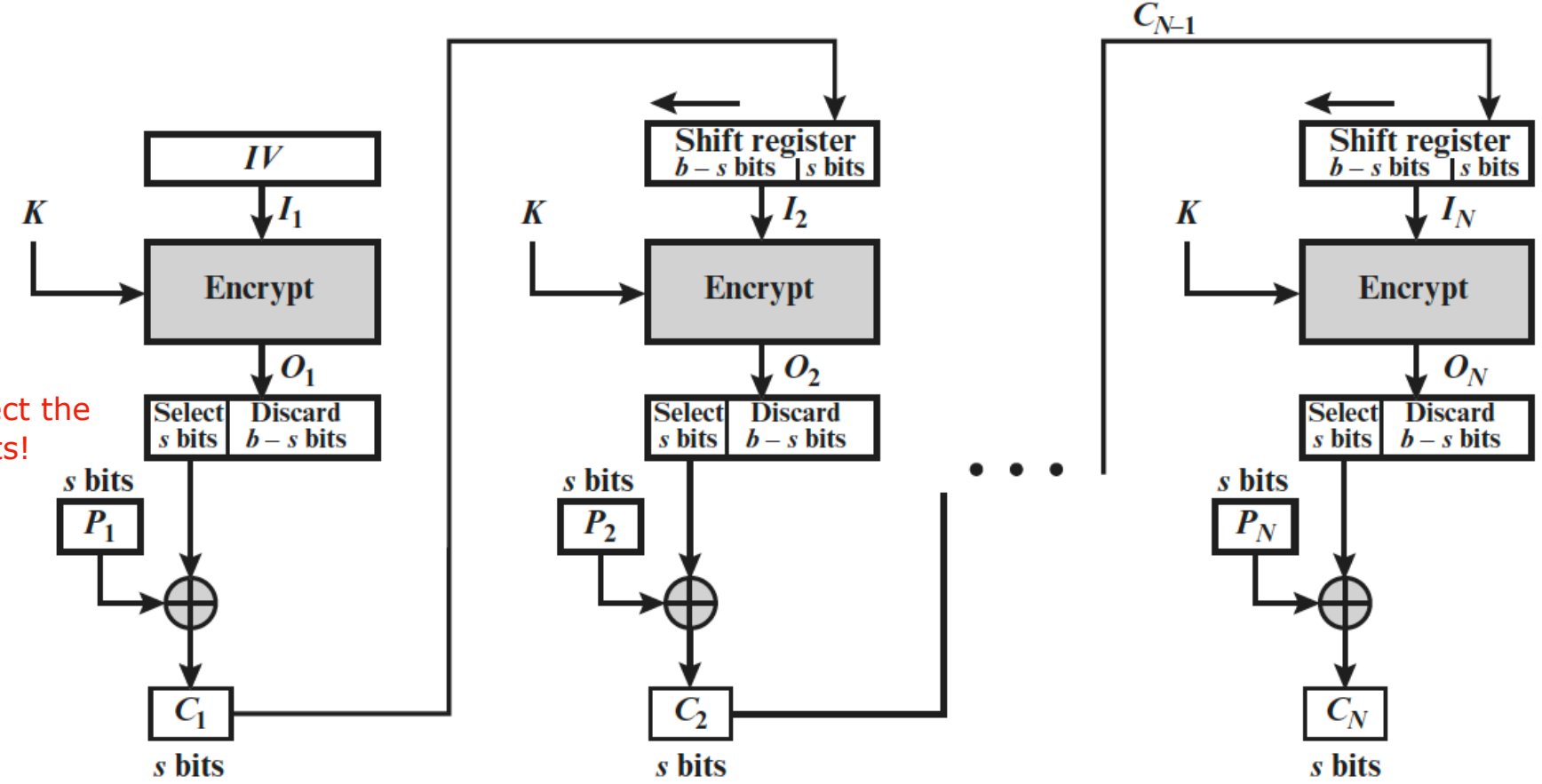
- the long pseudorandom sequence deriving from the nonce can be generated in advance

Limitations:

- we must never reuse the same pseudorandom sequence (it is like a one-time pad!)
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs

Cipher Feedback (CFB)

In this case, rather than blocks of *b* bits, the plaintext is divided into segments of *s* bits.



always select the leftmost bits!

Cipher Feedback (CFB)

Advantages:

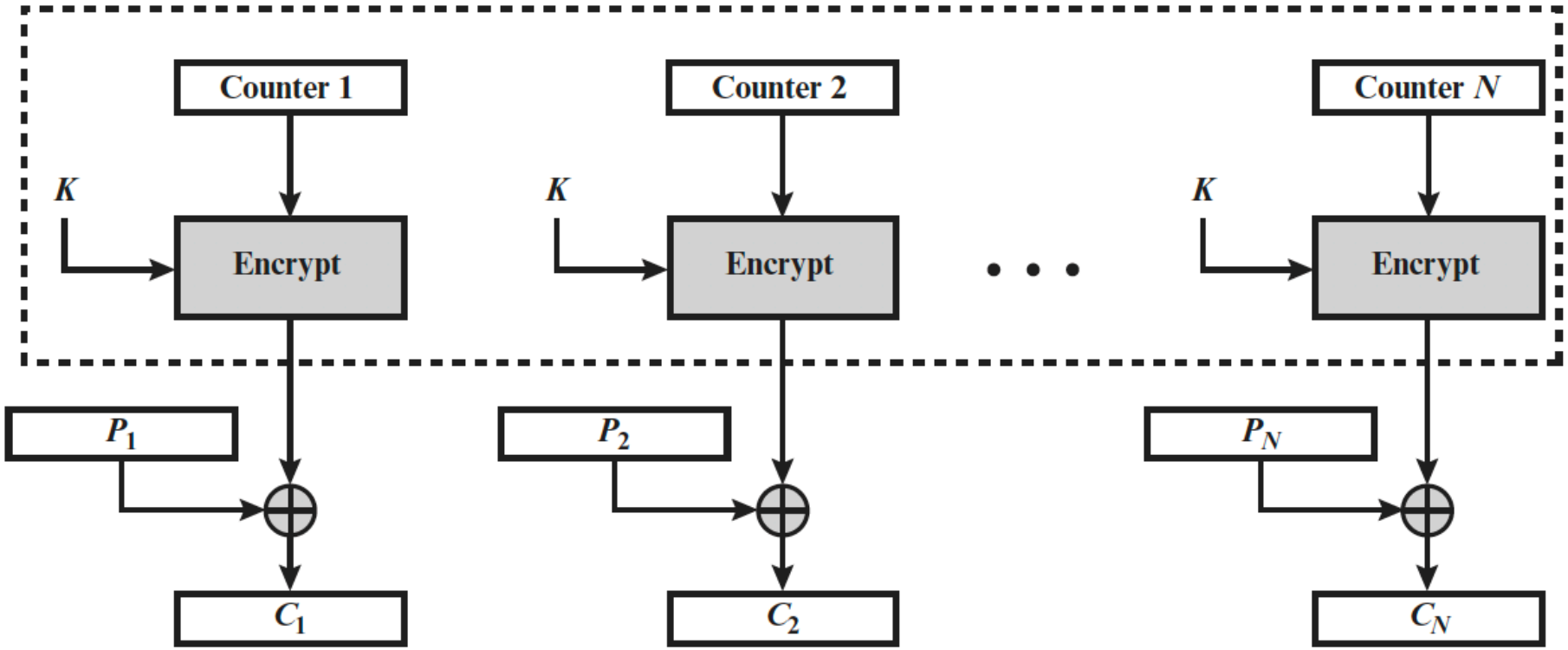
- like OFB, but the long pseudorandom sequence deriving from IV cannot be generated in advance

Limitations:

- Like OFB

Counter Mode (CM)

Similar to OFB.



Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b).

Counter Mode (CM)

Advantages:

- the forward cipher functions can be applied to the counters prior to the availability of the plaintext or ciphertext data
- the cipher functions can be performed in parallel
- the plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks if the corresponding counter block can be determined

Limitations:

- we must never reuse the same counter sequence across messages
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs

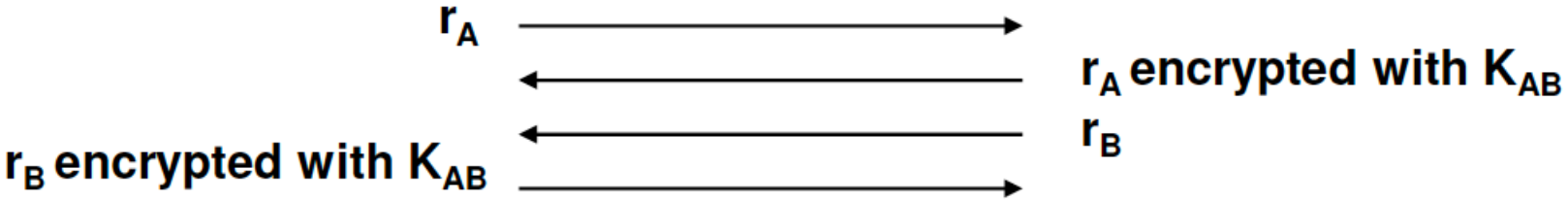
Main security uses of secret key cryptography

Confidentiality

- Transmitting over an insecure channel: the two parties agree on a shared secret key and use secret key cryptography to send messages
- Secure storage on insecure media: the user uses a secret key to store and retrieve data on an (insecure) media

Authentication

- Strong authentication through a challenge-response mechanism



Disadvantages of secret key cryptography

- Requires a secure method for key exchange

- In case of pre-shared permanent keys, the total number of secret keys for *N* users is:

$$N(N-1)/2$$

—> too many keys to handle

References

William Stallings, ***Cryptography and Network Security - Principles and Practice***, 7th edition, Pearson 2017

Horst Feistel, ***Cryptography and Computer Privacy***, Scientific American, May 1973

Claude Shannon, ***Communication Theory of Secrecy Systems***, Bell Systems Technical Journal, No. 4, 1949

Andrey Bogdanov *et al.*, ***Biclique Cryptanalysis of the Full AES***, 2011