Table of contents

Symmetric Cryptography

Stream Ciphers (Stream Encryption)

Introduction to Stream Ciphers

Definition and Principle

Stream ciphers are a family of encryption systems characterized by:

- **Unit block size**: each encrypted block = 1 bit
- **Two-phase architecture**:

1. **Keystream generation**: production of the key sequence
2. **Substitution**: operation on plaintext bits based on the keystream

Classic example: the *one-time pad*

- Generation: (pseudo-)random generator
- Substitution: XOR operation ($\oplus$) with the keystream

General Characteristics

Advantages:

- **Speed**: encryption at register level, ideal for real-time *streaming* (video)
- **Lightweight**: work on systems with limited CPU resources
- **Low memory**: little or no buffering needed
- **Non-propagated errors**: retransmission of defective packets is sufficient (suitable for wireless transmissions - WiFi)

Disadvantages:

- **Dependency on keystream quality**: randomness determines robustness
- **Dangerous reuse**: keystream reuse allows easy cryptanalysis

> Original text
>
> #### Stream Ciphers
>
> - **Stream ciphers** constitute a **family of encryption systems** where the **size of the encrypted block is equal to 1 bit**.
> - Stream ciphers are generally composed of **two phases**:
>   - A **generation phase** of the sequence of elements forming the key (the **keystream**).
>   - A **substitution phase** where the *plaintext* bits undergo a specific operation dependent on the keystream.
> - An obvious example of a stream cipher is the **one-time pad** with:
>   - A keystream generation phase performed by a **(pseudo-)random generator**.
>   - A substitution phase consisting of performing a **xor** ($\oplus$) with the keystream.
>
> #### Stream Ciphers: Characteristics
>
> - **Speed**: Encryption is done directly at the register level. Ideal for applications requiring *"on the fly"* encryption like **video streaming**.
> - **Ease**: Operations can be performed by systems with **limited CPU resources**.

- **No (or little...) need for memory/buffering**.
- **Limited or absent error propagation**: retransmission of faulty packets is normally sufficient (suitable for applications where packet loss is frequent like **wireless transmissions (WiFi)**).
- **Disadvantages**:
    - The **quality in terms of randomness** of the generated keystream determines the **system's robustness**.
    - **Keystream reuse** allows **easy cryptanalysis** (cf. the one-time pad).

Quick revision

Stream Ciphers = encryption bit by bit in 2 phases (keystream generation + substitution).
Advantages: fast, lightweight, no error propagation.
Disadvantages: keystream quality critical, reuse = vulnerability.

Synchronous Stream Ciphers

Operating Principle

In a **synchronous stream cipher**, the keystream depends **only on the key**, independent of the plaintext and ciphertext.

Process equations:

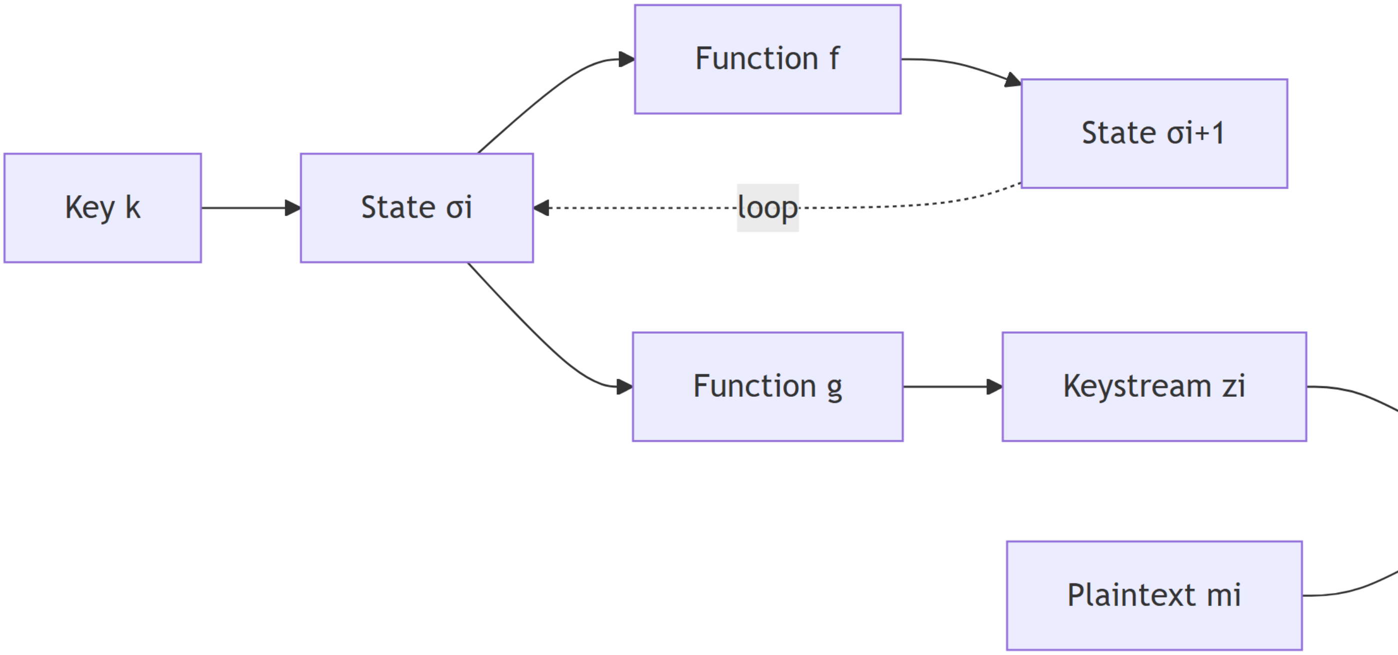
$$\sigma_{i+1} = f(\sigma_i, k)$$

$$z_i = g(\sigma_i, k)$$

$$c_i = h(z_i, m_i)$$

Where:

- σ_i: state at time i (initial state σ_0 may depend on k)
- k: secret key
- f: state transition function
- g: keystream production function producing z_i
- h: output function producing ciphertext c_i from plaintext m_i



Characteristics

Synchronization requirement:

- Transmitter and receiver must share the same key k **AND** the same state σ_i
- Loss of synchronization = need for external mechanisms (markers, redundancy analysis)

Properties:

- **No error propagation**: modification of ciphertext does not affect subsequent sequences
- **Attention**: deletion of a ciphertext = receiver desynchronization

Vulnerabilities to active attacks:

- Detection: insertion, elimination, replay of fragments
- Bit modification: adversary can modify bits and analyze impact on plaintext
- **Solution**: additional authentication mechanisms necessary

Special case: Additive Stream Cipher

The most frequent case where:

- Functions f and g replaced by a random generator
- Function h = modulo 2 addition (XOR: $\oplus$)

Formula: $c_i = z_i \oplus m_i$

Original text

Synchronous Stream Ciphers

- The **generated keystream depends only on the key** and not on the plaintext nor the ciphertext.
- The encryption process of a **synchronous stream cipher** is described by the following equations:

$$\sigma_{i+1} = f(\sigma_i, k)$$
$$z_i = g(\sigma_i, k)$$
$$c_i = h(z_i, m_i)$$

with σ_i the **initial state** which may depend on the key k, f the **function determining the next state**, g the **function producing the keystream** z_i and h the **output function** producing the ciphertext c_i from the plaintext m_i.

Synchronous Stream Ciphers: Characteristics

- **Require synchronization** of the transmitter and receiver: In addition to using the same key k, both must be in the **same state** for the process to work. If synchronization is lost, **external mechanisms** are needed to recover it (special markers, plaintext redundancy analysis, etc.)
- **No error propagation**. Modification of the ciphertext during transmission does not cause disturbances in subsequent ciphertext sequences (however, the **deletion** of a ciphertext would cause **desynchronization** of the receiver).
- **Active attacks**: Insertion, elimination or replay of parts of ciphertext are **detected** by the receiver. However, an adversary could **modify certain bits** of the ciphertext and analyze the impact on the corresponding plaintext. Additional **origin authentication mechanisms** are necessary to detect these attacks.
- **Most frequent case** of Synchronous Stream Ciphers: the **additive stream cipher** (cf. the one-time pad) where the functions f and g generating the keystream are replaced by a **random generator** and the function h is a **modulo 2 addition (xor)**.

Quick revision

Synchronous: keystream = f(key only). Equations: $\sigma_{i+1} = f(\sigma_i, k)$, $z_i = g(\sigma_i, k)$, $c_i = h(z_i, m_i)$.
Requires synchronization transmitter/receiver. No error propagation but vulnerable to bit modifications.

> **Frequent case**: additive cipher with XOR.

Asynchronous Stream Ciphers

Operating Principle

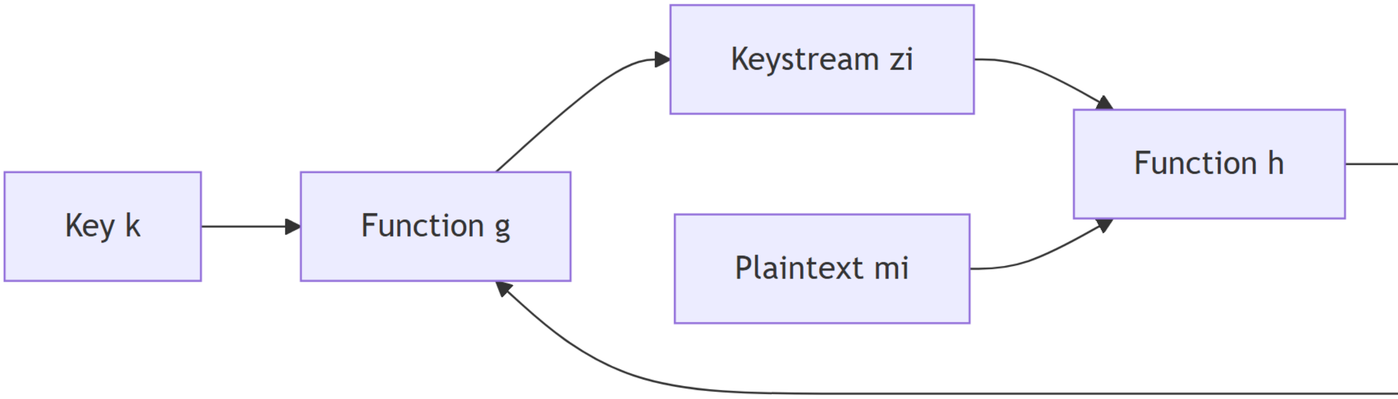
Also called **self-synchronizing ciphers**.

The keystream depends on the key **AND** a fixed number of previous ciphertexts.

Process equations:

$$\sigma_i = (c_{i-t}, c_{i-t+1}, \dots, c_{i-1})$$
$$z_i = g(\sigma_i, k)$$
$$c_i = h(z_i, m_i)$$

Where σ_i represents a buffer of the last t ciphertexts.



Characteristics

Self-synchronization:

- In case of insertion/elimination of ciphertexts, the receiver **automatically resynchronizes**
- Mechanism: memorization (buffer) of the last ciphertexts

Limited error propagation:

- Error propagates only over the **buffer size** (t bits)
- After buffer exhaustion, correct decryption resumes

Security against active attacks:

- **Better detection**: modifications detected thanks to error propagation
- **Attention**: self-synchronization allows receiver to continue even after insertions/deletions
- **Solution**: verification of integrity and authenticity of entire stream necessary

Diffusion of plaintext statistics:

- Each plaintext bit influences **all subsequent ciphertexts**
- **Result**: better dispersion of statistics vs. synchronous case
- **Application**: use for low entropy or highly redundant plaintexts

> ℹ Original text
>
> **Asynchronous Stream Ciphers**
>
> - Also called **self-synchronizing ciphers**.
> - The **generated keystream depends on the key as well as a fixed number of previous ciphertexts**.
> - The encryption process of an **asynchronous stream cipher** is described by the following equations:
>
> $$\sigma_i = (c_{i-t}, c_{i-t+1}, \dots, c_{i-1})$$
>
> $$z_i = g(\sigma_i, k)$$
>
> $$c_i = h(z_i, m_i)$$
>
> with σ_i, g and h as for the synchronous case.
>
> **Asynchronous Stream Ciphers: Characteristics**
>
> - **Self-synchronization**: In case of elimination or insertion of ciphertexts along the way, the receiver is capable of **re-synchronizing with the transmitter** thanks to the **memorization (buffer)** of a number of previous ciphertexts.
> - **Limited error propagation**: Error propagation extends only to the **number of ciphertext bits memorized** (buffer size). Afterwards, decryption proceeds correctly again.
> - **Active attacks**: Modification of ciphertext fragments will be **more easily detected** than in the synchronous case because of error propagation. However, since the receiver is capable of self-synchronizing with the transmitter, even if ciphertexts are eliminated or inserted along the way, it is necessary to **verify the integrity and authenticity of the entire stream**.

- **Diffusion of plaintext statistics**: The fact that **each plaintext bit will influence all subsequent ciphertexts** results in a **greater dispersion of statistics** compared to the synchronous case…
- … It is therefore advisable to use **asynchronous stream ciphers when the entropy of plaintexts is limited** and could allow targeted attacks on highly redundant plaintexts.

💡 Quick revision

Asynchronous (self-synchronizing): keystream $= f$(key + last ciphertexts). State $\sigma_i =$ buffer of t previous ciphertexts.
Automatic self-synchronization. Limited error propagation to buffer.
Better diffusion of statistics → ideal for redundant/low entropy plaintexts.

Keystream Generators: LSFR

Context and Necessity

Problem: generate a keystream of length m from a secret key of length l with $l \ll m$.

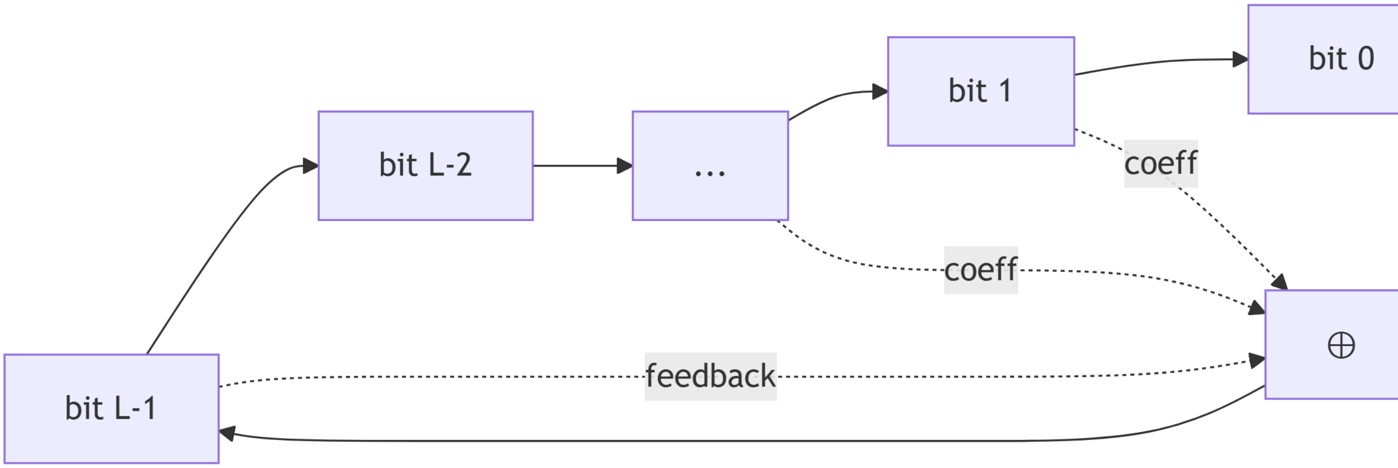
Solution: Linear Feedback Shift Register (**LSFR** or **LFSR**)

LSFR Characteristics

Advantages:

- **Optimal hardware implementation**: very efficient circuits
- **Long periods**: sequences of great length
- **Good random quality**: notable randomness
- **Mathematical basis**: algebraic properties of linear combinations

Generic structure: LSFR of length L



Important Remarks on LSFR

History and Usage:

- Very widespread construction in cryptography and coding theory
- Many military stream ciphers based on LSFR

Security Limits:

- **Insufficient security level** compared to modern block ciphers
- **Vulnerability**: the Berlekamp-Massey algorithm allows to:
  - Determine the **linear complexity** of an LSFR
  - Calculate an arbitrary number of generated sequences

Metric: Linear complexity (*linear complexity*)

Improvement Solution:

Replace the linear combination with a **non-linear function** f

→ **Non Linear Feedback Shift Registers** (NLFSR)

> Original text
>
> **Stream Ciphers: Keystream Generators**
>
> - When it is necessary to **generate a keystream of length** m from a **secret key of length** l with $l \ll m$, we call upon **keystream generators**.
> - The most common of these generators is the **Linear Feedback Shift Register (LSFR)**.

- An LSFR has the following characteristics:
  - **Adapts very well to hardware implementations**.
  - Produces sequences of **long periods** and with **notable random quality** (quite strong randomness)
  - Based on the **algebraic properties of linear combinations**.

LSFRs: Some Remarks

- LSFRs are **very widespread constructions** in cryptography and coding theory.
- A **large number of stream ciphers** based on LSFRs (especially in the **military sphere**) were developed in the past.
- Unfortunately, the **security level offered by these systems is deemed insufficient** nowadays (compared to that of block ciphers…)
- The **metric** allowing analysis of an LFSR is its **linear complexity**. The **Berlekamp-Massey algorithm** allows determining the linear complexity of an LSFR and thus calculating an arbitrarily large number of sequences generated by an LSFR.
- A solution to **increase complexity** is to substitute the linear combination of ciphertext bits with a **non-linear function** f. These are the **Non Linear Feedback Shift Registers**.

Quick revision

LSFR: long keystream generator (m) from short key (l). Base = linear combinations.
Advantages: efficient hardware, long periods.
Problem: insufficient security, vulnerable to Berlekamp-Massey (linear complexity calculation).
Solution: NLFSR (non-linear function).

RC4: Software Stream Cipher

General Presentation

RC4™ (*Rivest Cipher 4*) developed in 1987 by Ron Rivest for RSA Security.

Main characteristics:

- **Variable key**: flexible length
- **Extremely fast**: 10× faster than DES

- **Synchronous mode**: keystream independent of plaintext/ciphertext

History:

- 1987-1994: patented, details confidential (NDA contract required)
- 1994: unofficial publication in a newsgroup
- Since then: intensive analysis by cryptographic community

Architecture

Key components:

- **S-box**: 8×8 substitution box (256 entries)
  - Content: permutation of numbers 0 to 255
  - Depends on the main key of variable length: $0 < len(k) \leq 255$
- **Combinations**: linear and non-linear
- **Final encryption**: XOR between keystream and plaintext

Applications and Security

Commercial uses (numerous):

- Lotus Notes
- Oracle SQL
- Microsoft Windows
- SSL/TLS
- And many more…

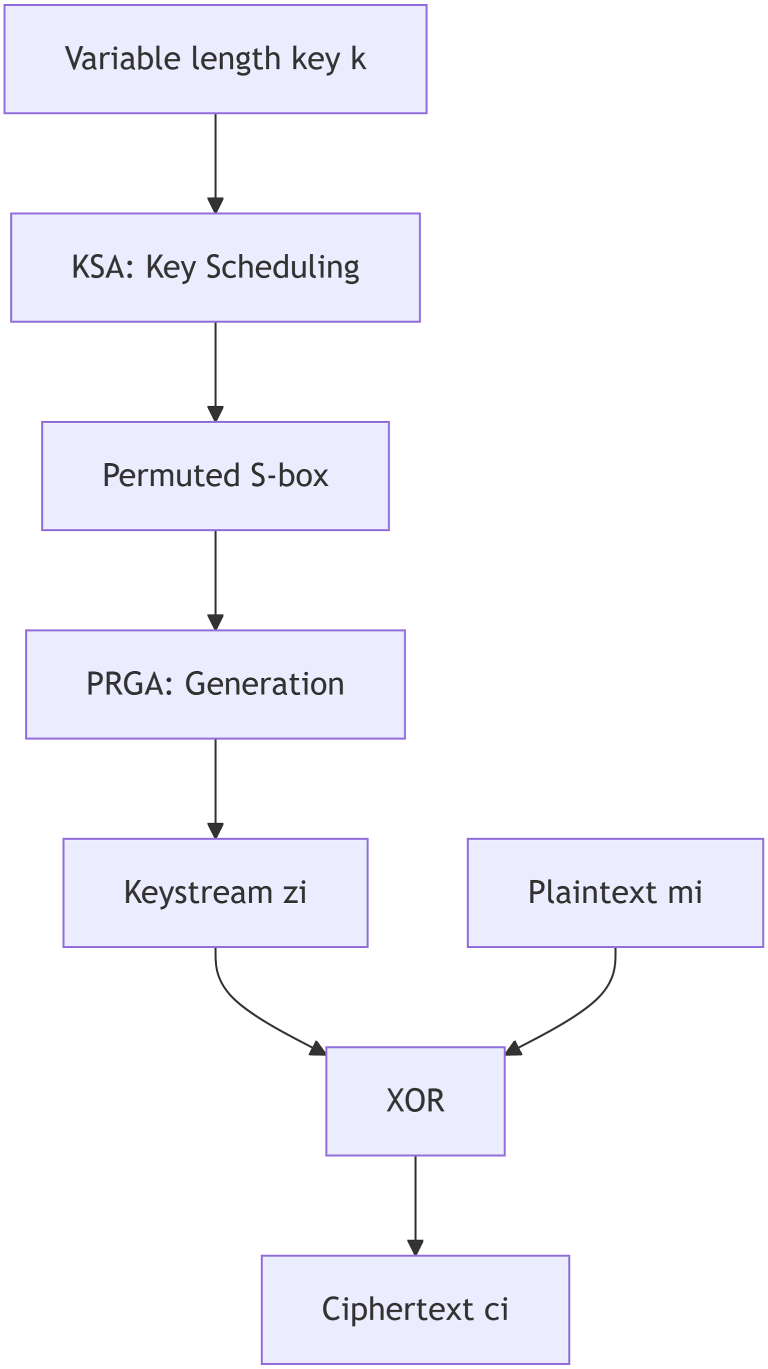
Analyses and Vulnerabilities:

- Exhaustive work on key scheduling and PRGA
- **Major flaw**: implementation in WEP (WiFi Wired Equivalent Privacy)
  - WEP protocol completely compromised
  - Problem: faulty usage mode, not the RC4 algorithm itself

Operation

RC4 decomposes into **two steps**:

1. **Key Scheduling Algorithm (KSA)**
   - Responsible for initial permutation of the S-box
   - Function of the variable length key $len(k) = l$
2. **Pseudo Random Generator Algorithm (PRGA)**
   - Generates keystream of arbitrary size
   - Relies on S-box permuted by KSA


Variable length key k
KSA: Key Scheduling
Permuted S-box
PRGA: Generation
Keystream zi
Plaintext mi
XOR
Ciphertext ci

Original text

Software Cipher Streams: RC4

- The major **disadvantage of stream ciphers based on registers** is that they are **very slow in programmed version** on a generic machine. **RC4™** is a **variable key stream cipher** developed in **1987 by Ron Rivest** for RSA security. It is **very fast** (**10 times faster than DES** !)
- For 7 years, this algorithm was **patented** and its internal operational details were disclosed only after **signing a confidentiality contract**. Since its **(unofficial) publication in a newsgroup in 1994**, it has been widely discussed and analyzed by the entire cryptographic community.
- The algorithm works in **synchronous mode** (the keystream is independent of the ciphertext and plaintext).
- It is composed of **linear and non-linear combinations**. The key element is an **8×8 substitution box (S-box)** whose entries are a **permutation of the numbers 0 to 255**. The permutation is a **function of the main key** of variable size with $0 < len(k) \leq 255$. The **final encryption is obtained by a xor** between the keystream and the plaintext.
- RC4 is used in a **large number of commercial applications**: Lotus Notes, Oracle SQL, MS Windows, SSL, etc. It is the subject of a **large number of analytical and exhaustive works** that have managed to **compromise the security** of the key scheduling and the PRGA.
- In particular the application of RC4 to the **Wired Equivalent Privacy (WiFi WEP) protocol has been "broken"** due to a **flaw in the protocol's usage mode**.

RC4: Operation

- The algorithm consists of **two steps**:
  - The **Key Scheduling Algorithm (KSA)**: Responsible for the **initial permutation** that will fill the S-box depending on the variable length key $len(k) = l$.
  - The **Pseudo Random Generator Algorithm (PRGA)**: Generates the **keystream of arbitrary size** relying on the S-box.

Quick revision

RC4: software stream cipher, variable key, 10× faster than DES.
Architecture: 8×8 S-box (permutation 0-255) + XOR.
2 steps: KSA (S-box permutation) + PRGA (keystream generation). Synchronous mode.
Vulnerability: WEP broken (usage flaw). Used in SSL, Windows, Oracle...

Block Ciphers (Block Encryption)

1. Introduction to Block Ciphers

Definition and Principle

A **block cipher** is a cryptographic function that:

- **Transforms fixed-size blocks**: maps a block of n bits to another block of the same size
- **Is parameterized by a key**: the key K of k bits defines the transformation
- **Must be bijective**: to allow unique decryption
- **Each key = different bijection**: guarantees variability

Nominal size: input block size on which encryption is applied

Quality Criteria

1. Key size/Entropy

- Keys ideally **equiprobable** with entropy = k bits
- Strong entropy protects against **brute-force attacks**
- **Minimum required**: 128 bits for modern block ciphers

2. Performance

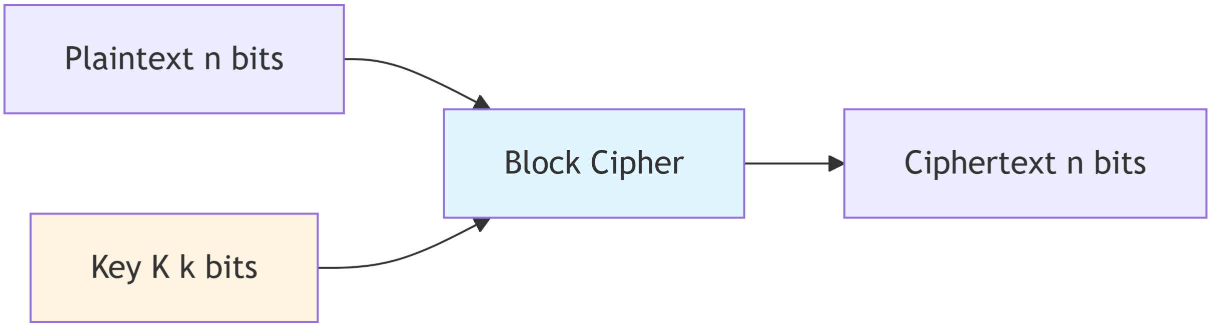
- Execution speed
- Software/hardware efficiency

3. Block size

- Too small block = vulnerability to **plaintext/ciphertext dictionaries**
- **Modern standard**: blocks 128 bits

4. Cryptographic resistance

- Resistance to known techniques:
  - Linear cryptanalysis
  - Differential cryptanalysis
  - Meet in the middle
- **Cryptanalysis effort** equivalent to brute force



Original text

Block Ciphers

- **Symmetric block ciphers** constitute the **cornerstone of cryptography**. Their main functionality is **confidentiality** but they are also the basis for **authentication**, **hashing functions**, **random generation**, etc.

- **Definition**: A block cipher is a **function** that maps a **block of n bits** to another block of **the same size**. The function is **parameterized by a key K of k bits**. To allow **unique decryption**, the function must be **bijective**. **Each key defines a different bijection**. The **input block size** on which encryption is applied is also called **nominal algorithm size**.

- **Criteria to evaluate the quality** of a block cipher:

  - **Key size/Entropy**: Ideally, keys are **equiprobable** and the key space has an **entropy equal to** k. A **strong key entropy** protects against **brute-force attacks** from chosen/known plaintexts. Modern block ciphers must have **keys of at least 128 bits**.
  - **Performance**
  - **Block size**: A **too small block** would allow attacks where **plaintext/ciphertext "dictionaries"** could be built. Nowadays, **blocks of size 128 bits** are becoming common.
  - **Cryptographic resistance**: The block cipher must show **resistance** to known cryptanalysis techniques: **linear or differential cryptanalysis**, **meet in the middle**, etc. The **inherent effort** of these attacks (complexity, storage, parallelization, etc.) must be **equivalent to that of a brute force attack**.

Quick revision

Block cipher: bijective function transforming blocks of n bits with key K of k bits. **Criteria**: key entropy 128 bits, block size 128 bits, cryptanalysis resistance = brute force effort. **Usage**: confidentiality, authentication, hashing, random generation.

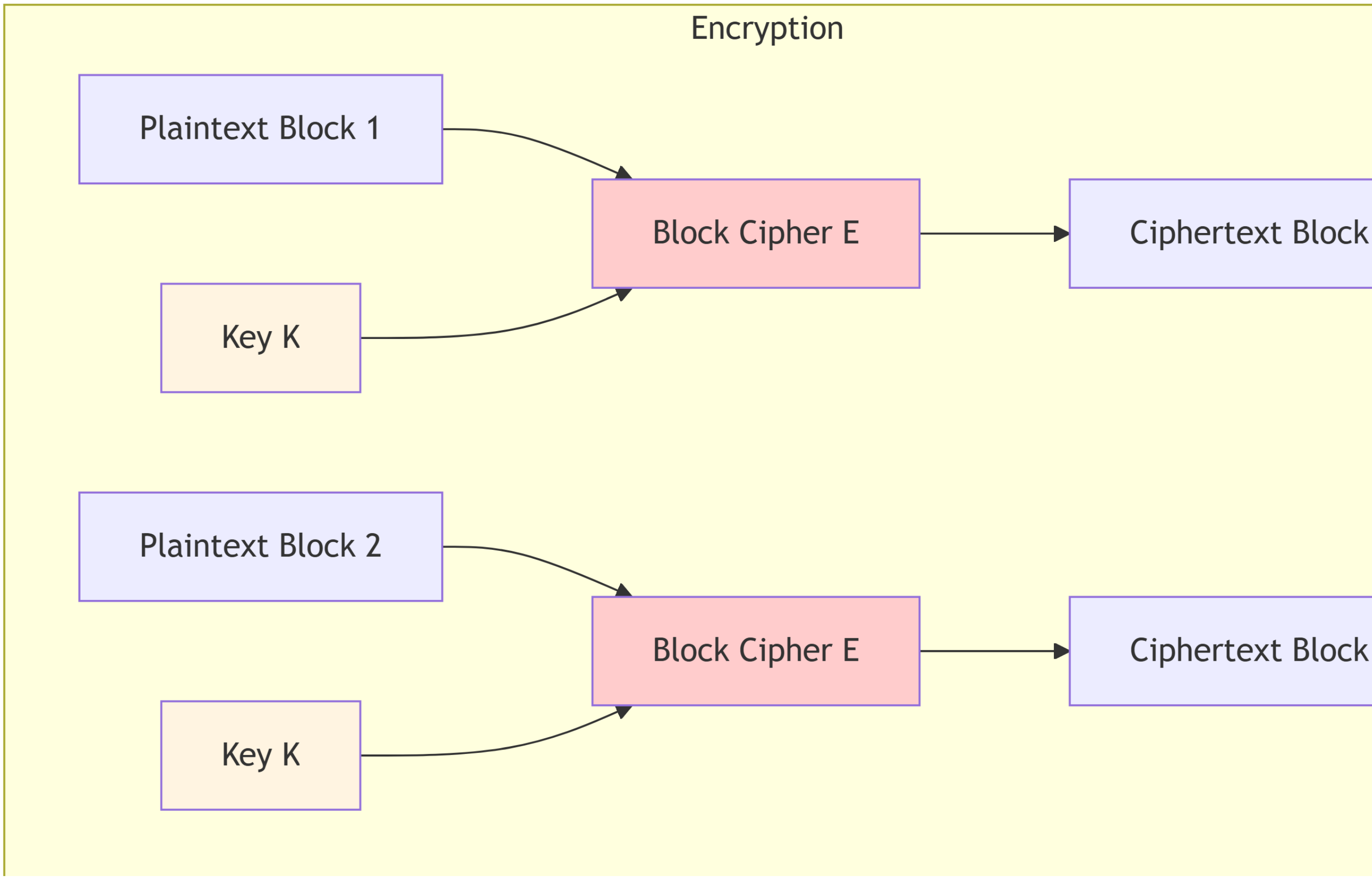
2. Block Cipher Modes of Operation

2.1 Electronic Codebook (ECB)

Principle: each plaintext block is encrypted **independently** with the same key.

$$c_i = E_K(m_i)$$

$$m_i = D_K(c_i)$$



Characteristics:

- **Identical plaintexts** → identical ciphertexts (predictable)
- **No error propagation**: error on c_j affects only m_j
- **Visible patterns**: plaintext structure transparent in ciphertext
- **Parallelizable**: each block processed independently

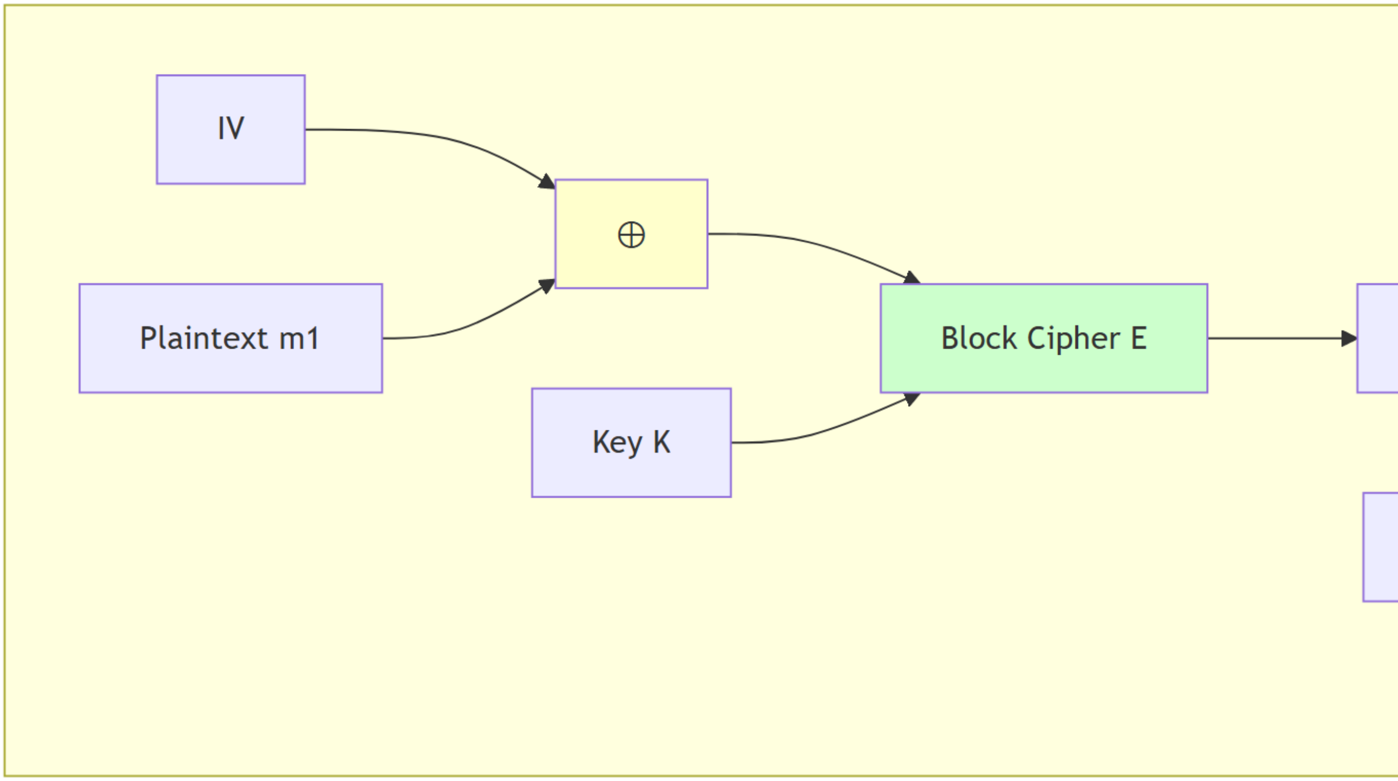
Major vulnerability: Should NOT be used for redundant data

2.2 Cipher Block Chaining (CBC)

Principle: each plaintext block is **XORed with the previous ciphertext** before encryption.

$$c_i = E_K(m_i \oplus c_{i-1})$$
$$m_i = D_K(c_i) \oplus c_{i-1}$$

With $c_0 = IV$ (Initialization Vector)



Characteristics:

- **Identical plaintexts** → different ciphertexts (if IV changes)
- **Patterns erased**: chaining masks the structure
- **Limited error propagation**: error on c_j affects m_j and m_{j+1} only
- **Not parallelizable** in encryption (sequential)
- **Parallelizable** in decryption

IV (Initialization Vector):

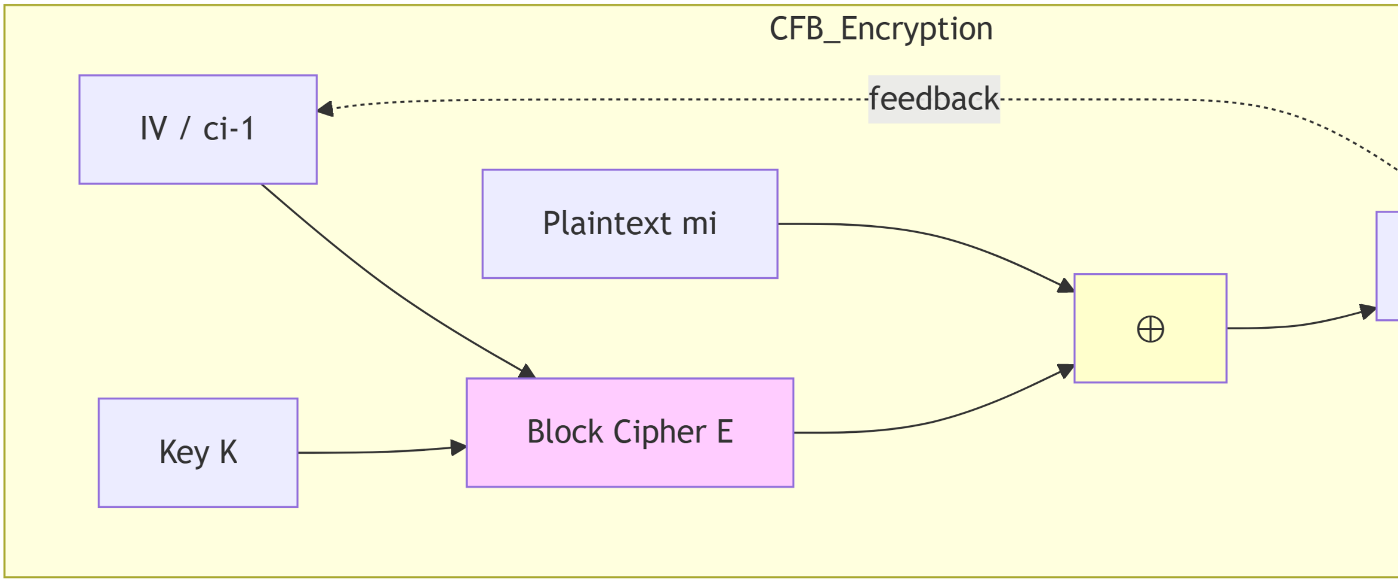
- Must be **random** or **pseudo-random**
- Can be transmitted **in clear**
- Must be **different** for each message with the same key

2.3 Cipher Feedback Mode (CFB)

Principle: works like a **stream cipher** where the keystream is generated by the block cipher. The keystream depends on **previous ciphertexts** (**asynchronous** mode).

$$c_i = m_i \oplus E_K(c_{i-1})$$
$$m_i = c_i \oplus E_K(c_{i-1})$$

With $c_0 = IV$



Characteristics:

- **Identical plaintexts** → different ciphertexts (if IV changes)
- **Chaining**: dependencies between ciphertexts
- **Error propagation**: error on c_j affects $\frac{n}{r}$ following blocks
  - n = nominal size of block cipher
  - r = size of plaintexts
- **Not parallelizable**
- **IV non-confidential** but must be transmitted

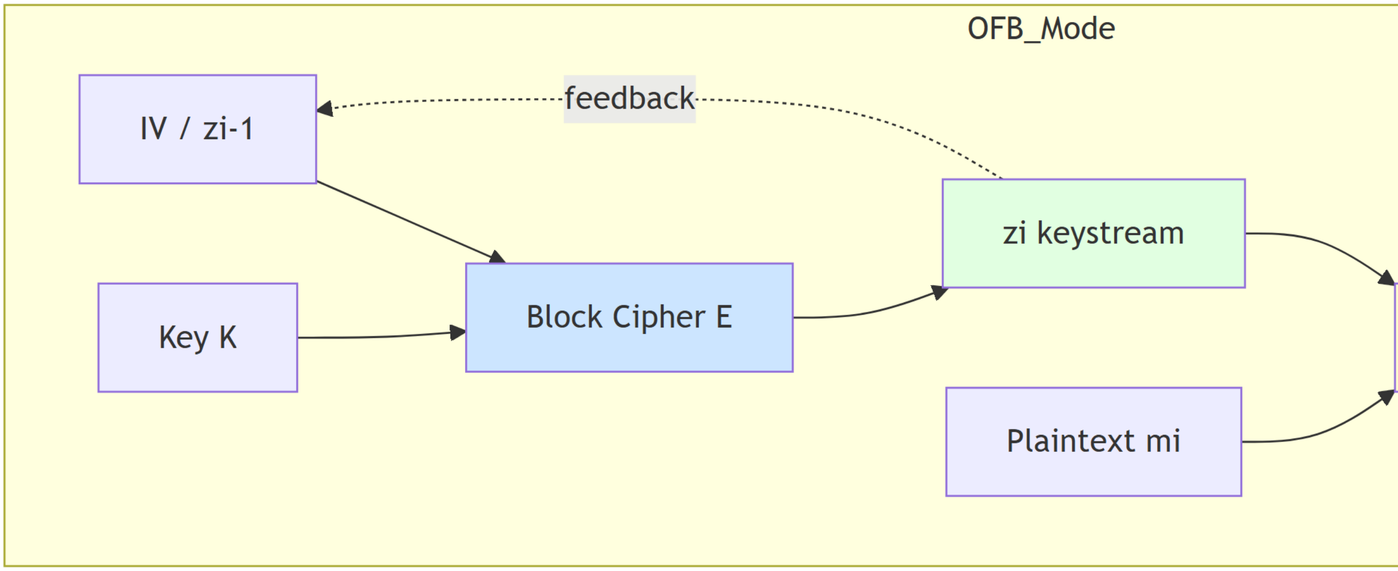
Usage: suitable for transmissions with frequent packet loss

2.4 Output Feedback Mode (OFB)

Principle: works like a **synchronous stream cipher**. The keystream is **entirely determined** by the key and IV, **independent** of plaintext and ciphertext.

$$z_i = E_K(z_{i-1})$$
$$c_i = m_i \oplus z_i$$
$$m_i = c_i \oplus z_i$$

With $z_0 = IV$



Characteristics:

- **Identical plaintexts** → different ciphertexts (if IV changes)
- **No error propagation**: error on c_j affects only m_j
- **Pre-computable keystream**: efficient
- **CRITICAL**: NEVER reuse the same IV with the same key (otherwise identical keystream)
- **Parallelizable** if keystream pre-computed

Reuse warning: Change IV for each new message!

ℹ Original text (CFB and OFB Modes)

CFB and OFB Modes: Characteristics
The **CFB and OFB modes** work as a **stream cipher** with a **keystream generated by the encryption block**. In **CFB**, the keystream depends on **previous ciphertexts** (**asynchronous**) whereas in **OFB**, the keystream is **entirely determined by the key and the IV** (**synchronous**).
Particularities of CFB:

- As in CBC mode, **identical plaintexts** are translated into **different ciphertexts** if the **IV changes**. The **IV is not necessarily confidential** and can be exchanged in clear between parties.
- **Chaining** also introduces **dependencies** between current ciphertexts and previous ciphertexts. In particular, if n is the **nominal algorithm size** and r is the **plaintext size**, the current ciphertext will depend on the $\frac{n}{r}$ **previous ciphertexts** (each iteration will shift the faulty input by r positions, after $\frac{n}{r}$ iterations the faulty ciphertext will be completely "expelled").
- **Error propagation** follows the same principle: an error in a ciphertext will result in incorrect decryption of the $\frac{n}{r}$ following ciphertexts.

Particularities of OFB:

- OFB has **identical behavior** to CBC and CFB modes for **encryption of identical plaintexts**.
- **No error propagation** on adjacent ciphertexts.
- **Modify the IV** if the key does not change to **avoid keystream reuse** !!!

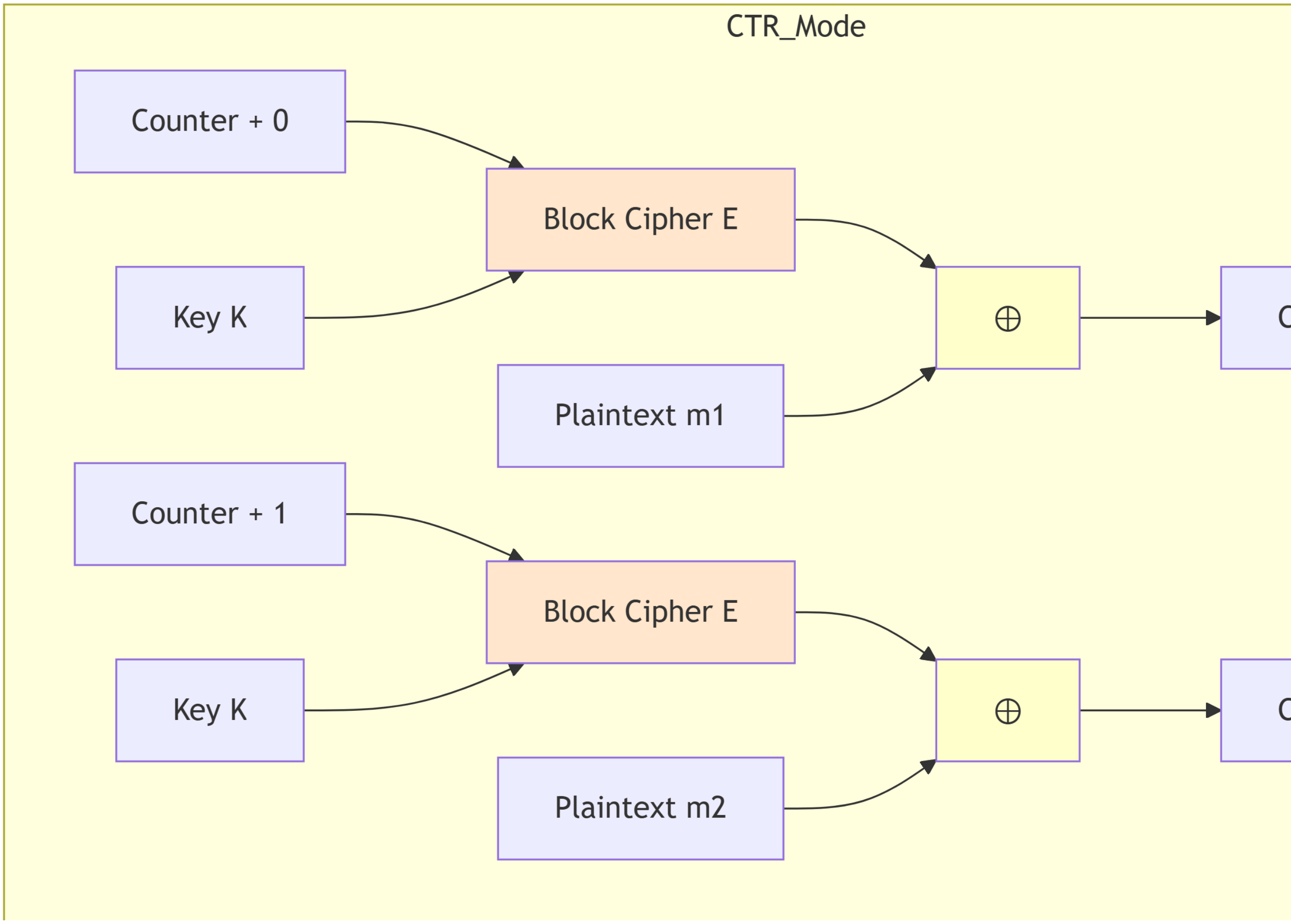
Quick revision (CFB/OFB)

CFB (asynchronous): keystream = f(previous ciphertexts). Limited error propagation ($\frac{n}{r}$ blocks).
OFB (synchronous): keystream = f(key + IV only). No error propagation.
CRITICAL: NEVER reuse same IV with same key. IV transmissible in clear.

2.5 Counter Mode (CTR)

Principle: the keystream is generated by **encryption of a counter** incremented at each block.

$$c_i = m_i \oplus E_K(counter + i)$$
$$m_i = c_i \oplus E_K(counter + i)$$



Characteristics:

- **Synchronous mode**: keystream = f(counter)
- **Parallelizable**: keystream pre-computable for encryption AND decryption
- **Random access**: each block decryptable independently
- **No error propagation**

- **Benefits from SIMD architectures**: no dependencies between blocks
- **Counter**: must be of size 2^b ($b =$ block size)
- **CRITICAL**: NEVER reuse the same counter with the same key

Counter management:

- **Increment modulo** 2^b after each iteration
- **Solution**: always increment for each encrypted stream
- First block of stream $i + 1 >$ last block of stream i

Applications:

- **ATM** (Asynchronous Transfer Mode)
- **IPsec** (IP security)
- **High-speed lines**: selective transmission of blocks
- **Large volume transfers**: video

> Original text (Counter Mode)
>
> **Counter Mode (CTR Mode)**
> Frequently used as encryption support in data transfer protocols like **ATM** (Asynchronous Transfer Mode) and **IPsec** (IP security).
>
> **Counter Mode (II)**
>
> - The **keystream** is generated by the **encryption of a random counter** of size 2^b (with b the block size) and necessary for decryption. This counter is **incremented modulo** 2^b after each iteration.
> - Works in **synchronous mode**. **Reuse of the same counter** results in an **identical keystream** !
> - **Solution**: Always **increment the counter** for each encrypted stream such that the counter of the first block of a stream is **larger than the last block** of the previous stream.
> - **Easily parallelizable**: The keystream can be **pre-calculated** both for encryption and decryption. Fully benefits from **SIMD architectures** because unlike other chaining modes there are no **dependencies between operations** of different blocks.
> - **Random access** to encryption/decryption of each block: Unlike other chaining modes where the i-th operation depends on the $(i - 1)$-th operation.
> - If we add **absence of error propagation**, the counter mode facilitates **selective (re)transmission** of ciphertext blocks, making it very attractive for **securing high-speed lines** as well as for **encrypted transfers of large volumes** of information (e.g. video).

Quick revision (Counter Mode)

CTR: keystream = $E_K(\text{counter} + i)$.
Advantages: parallelizable (encryption + decryption), random access, no error propagation, SIMD-friendly.
CRITICAL: never reuse counter.
Usage: ATM, IPsec, high speed, video.

---

3. Product Ciphers and Feistel Ciphers

Product Ciphers

Definition: encryption scheme combining a **series of successive transformations** to strengthen resistance to cryptanalysis.

Common transformations:

- Transpositions (permutations)
- Substitutions (S-boxes)
- XORs
- Linear combinations
- Modular multiplications

Feistel Ciphers

Definition: iterative product cipher with specific structure.

Operating principle:

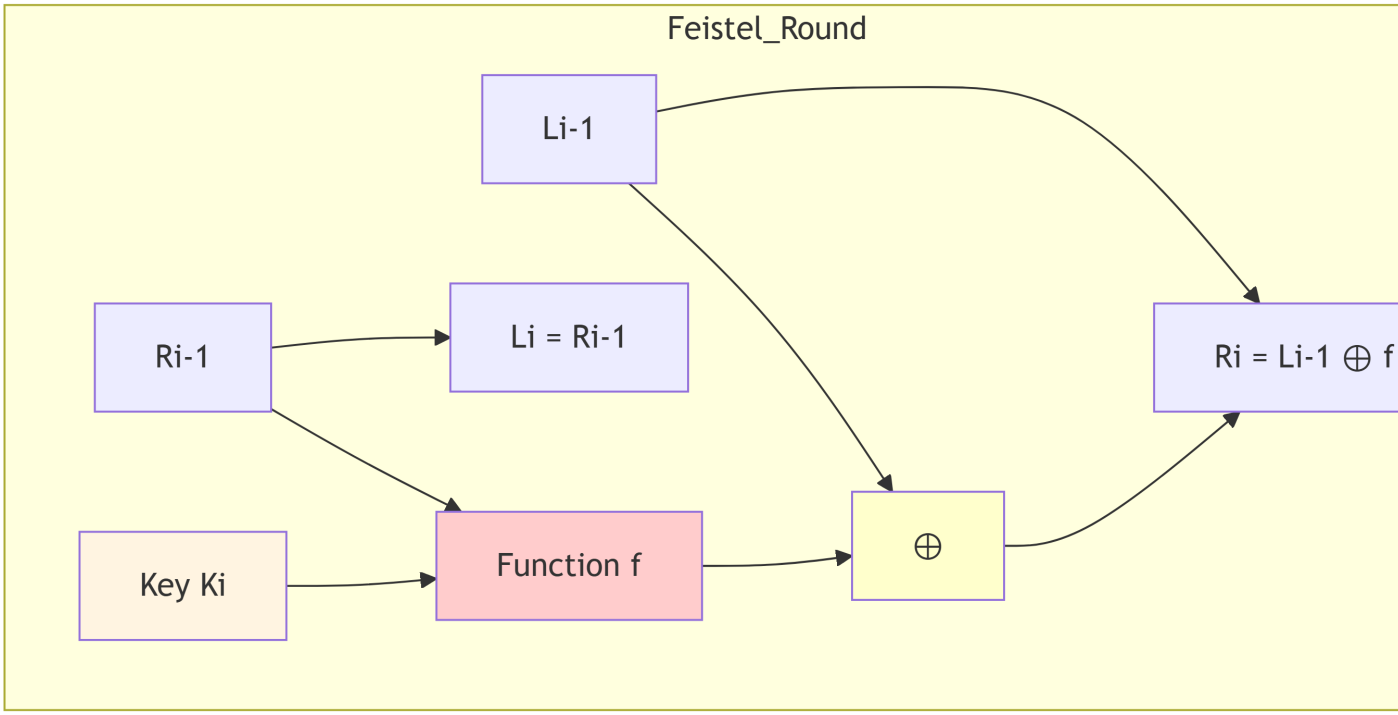
- **Input**: plaintext of $2t$ bits = (L_0, R_0) (two sub-blocks of t bits)
- **Output**: ciphertext of $2t$ bits = (R_r, L_r) after r steps (rounds)
- **Each step**: invertible bijection (for unique decryption)

Equations of step i $(1 \leq i \leq r)$:

$$(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$$

With:

- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$



Characteristics:

- K_i: subkeys generated from the main key K
- Number of steps r: generally **even** and ≥ 3
  - Example: DES has 16 steps
- **Final permutation**: $(L_r, R_r) \to (R_r, L_r)$
- **Decryption**: identical to encryption but subkeys applied in **reverse order** (from K_r to K_1)

Frequent operations:

- Permutations
- Substitutions (S-boxes)

Original text

Product Ciphers and Feistel Ciphers

- A **product cipher** is an **encryption scheme** combining a **series of successive transformations** to **strengthen resistance to cryptanalysis**. Common transformations for a product cipher are: **transpositions**, **substitutions**, **XORs**, **linear combinations**, **modular multiplications**, etc.

- A **Feistel cipher** is an **iterative product cipher** capable of transforming a **plaintext of** $2t$ **bits** of the form (L_0, R_0) composed of two **sub-blocks** L_0 and R_0 of t bits into a **ciphertext of size** $2t$ of the form (R_r, L_r) after r **successive steps (rounds)** with $r \geq 1$. **Each step** defines a **bijection (inversible !)** to allow unique decryption.

- **Permutations** and **substitutions** are the most frequent operations.

- The steps $1 \leq i \leq r$ are written: $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ with $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$. The K_i are **sub-keys**, **different for each step**, generated from the **main key** K of the encryption scheme.

- The **number of steps** proper to a Feistel cipher is normally **even** and ≥ 3 (e.g. **DES has 16 steps**)

- After execution of all steps, a Feistel cipher performs a **permutation** of the two parts (L_r, R_r) into (R_r, L_r).

- The **decryption** of a Feistel Cipher is **identical to encryption** except that the sub-keys K_i are applied in **reverse order** (From K_r to K_1).

Quick revision

Product cipher: combination of successive transformations (transpositions, substitutions, XOR).
Feistel cipher:

- iterative product cipher
- plaintext $2t$ bits $= (L_0, R_0)$
- r rounds with $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$.
- Decryption = encryption with reversed sub-keys.
- Example: DES (16 rounds).

---

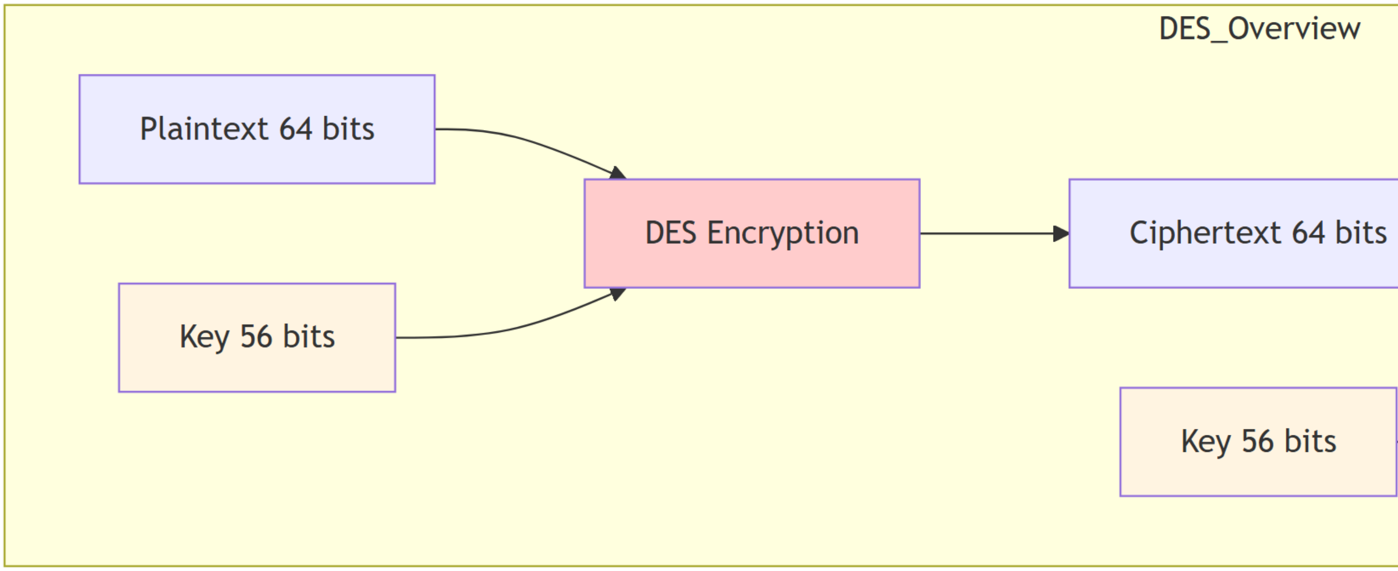
4. Data Encryption Standard (DES)

General Presentation

DES (Data Encryption Standard): most important cryptographic algorithm until the advent of AES in 2001.

Main characteristics:

- **Type**: Feistel Cipher
- **Block size**: 64 bits (nominal size)
- **Key size**: 56 effective bits (64 total bits with 8 parity bits)
- **Number of steps**: 16 rounds
- **Subkeys**: 16 subkeys of 48 bits (one per step)
- **Usage modes**: ECB, CBC, CFB, OFB



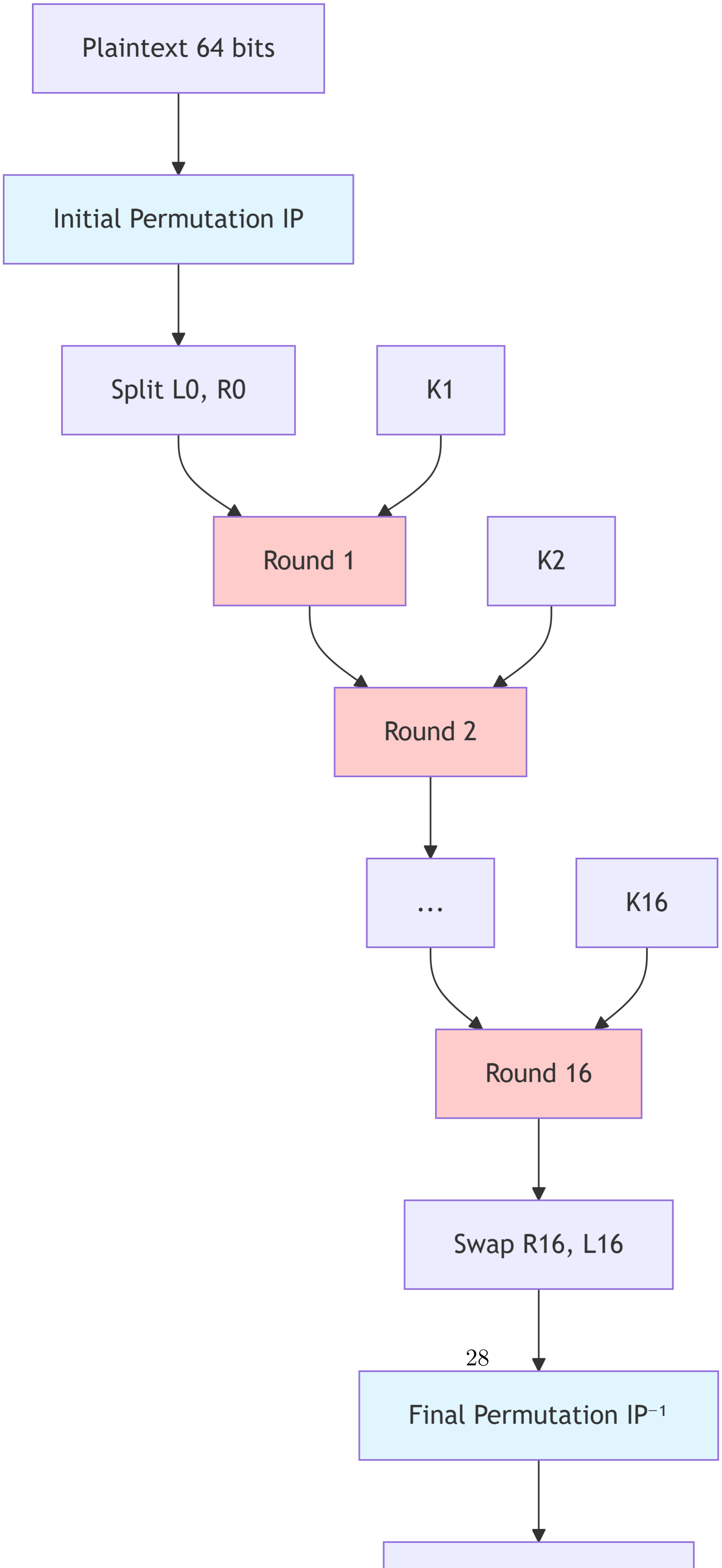
DES Structure

Main components:

1. **Initial permutation (IP)**: permutation of the 64 input bits
2. **16 Feistel rounds**: iterative transformation
3. **Final permutation (IP ^{1})**: inverse of IP

Each round applies:

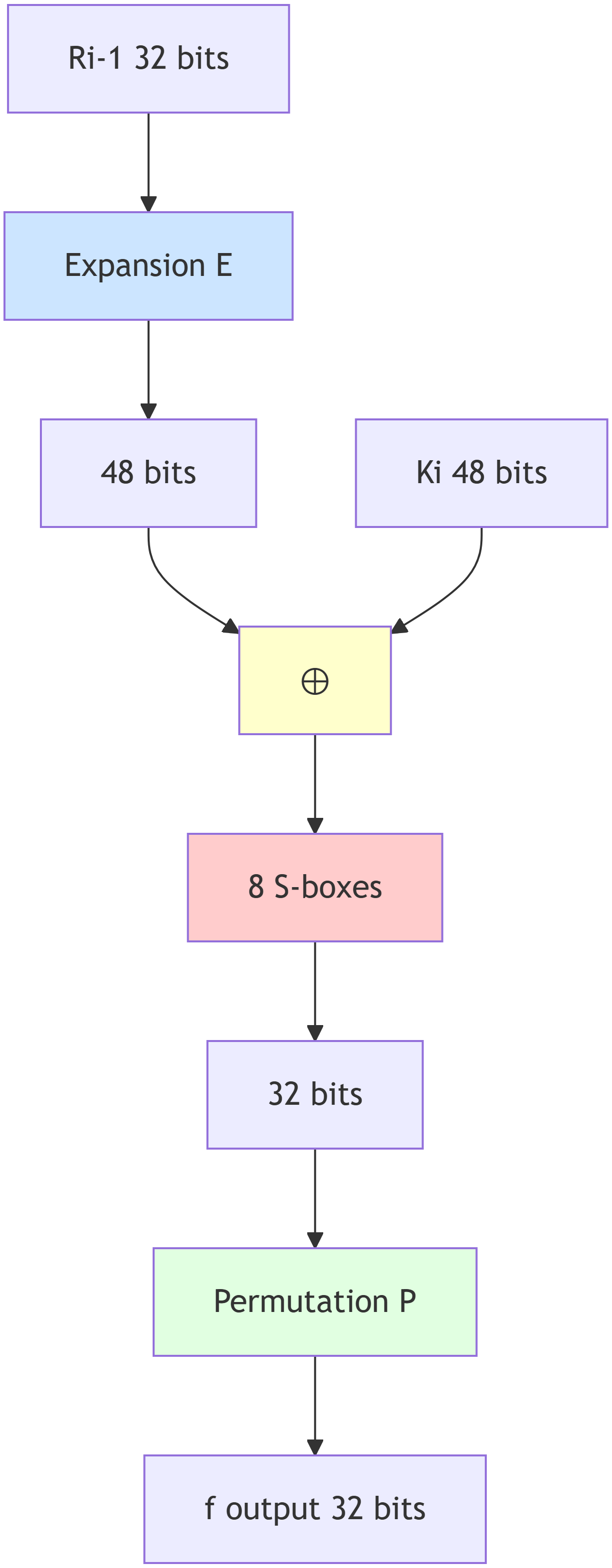
- Division into two halves: L_{i-1} and R_{i-1} (32 bits each)
- Function f on R_{i-1} with subkey K_i
- XOR with L_{i-1}
- Exchange of halves


Plaintext 64 bits
Initial Permutation IP
Split L0, R0
K1
Round 1
K2
Round 2
...
K16
Round 16
Swap R16, L16
Final Permutation IP^{-1}

DES Cipher Function

The **function** f for each round:

1. **Expansion E**: 32 bits → 48 bits (table E)
2. **Key Addition**: XOR with subkey K_i (48 bits)
3. **S-boxes**: 8 S-boxes transform 48 bits → 32 bits
    - Each S-box: 6 bits input → 4 bits output
4. **Permutation P**: permutation of the resulting 32 bits


Ri-1 32 bits
Expansion E
48 bits
Ki 48 bits
⊕
8 S-boxes
32 bits
Permutation P
f output 32 bits

S-box operation:

Input: $a_1 a_2 a_3 a_4 a_5 a_6$ (6 bits)

- **Row**: $a_1 + 2a_6$ (external bits)
- **Column**: $a_2 + 2a_3 + 4a_4 + 8a_5$ (internal bits)
- **Output**: value of the corresponding cell (4 bits)

Subkey Generation

Process:

1. Main key: 64 bits (56 effective + 8 parity)
2. **Permuted Choice 1 (PC-1)**: selection of 56 bits
3. Division into two halves: C_0 and D_0 (28 bits each)
4. For each round i:
    - Left circular rotation of C_{i-1} and D_{i-1}
    - **Permuted Choice 2 (PC-2)**: selection of 48 bits for K_i

Rotations:

- Rounds 1, 2, 9, 16: 1 position
- Other rounds: 2 positions

Original text (DES Operation)

DES: Operation

Cipher Function

- **Expansion E**: The **32 bits of the input** are transformed into a vector of **48 bits** using the **table E**. The first line of this table indicates how the first sub-block of 6 bits will be generated: first take the 32nd bit then bits 1,2,3,4,5. The second sub-block starts with the 4th bit then bits 5,6,7,8,9 and so on...

- **Key addition**: **XOR of the 48-bit vector** with the key.

- **S-boxes**: Apply **8 S-boxes** on the resulting 48-bit vector. Each of these S-boxes takes a **6-bit sub-block** and transforms it into a **4-bit sub-block**. The operation is performed as follows: If we denote the 6 input bits of the S-box as: $a_1 a_2 a_3 a_4 a_5 a_6$. The output is given by the content of the cell located in the **row** $a_1 + 2a_6$ and the **column** $a_2 + 2a_3 + 4a_4 + 8a_5$.

- **Permutation P**: Permutation P works as follows: The first bit is sent to the 16th position, the second to the 7th position and so on.

Permutations IP and IP ^{1}

- Act respectively at the **beginning** and at the **end** of the block processing and on the **entirety of the 64 bits**.

Quick revision (DES)

DES: Feistel cipher, 64-bit blocks, 56-bit effective key, 16 rounds.
Function f: Expansion E (32→48 bits) → XOR K_i → 8 S-boxes (48→32 bits) → Permutation P.
S-box: 6 bits input → 4 bits output via table (row = external bits, column = internal bits).
Permutations: IP (initial) and IP ^{1} (final) on 64 bits.

---

5. Triple-DES and DES Security

DES Vulnerabilities

Main problem: key space size $\{0,1\}^{56}$ insufficient.

Brute force attack:

- **1999**: key found in **24 hours**
- Technique: massively parallel brute force (100,000 PCs on Internet)
- Known plaintext attack

Triple-DES (3DES)

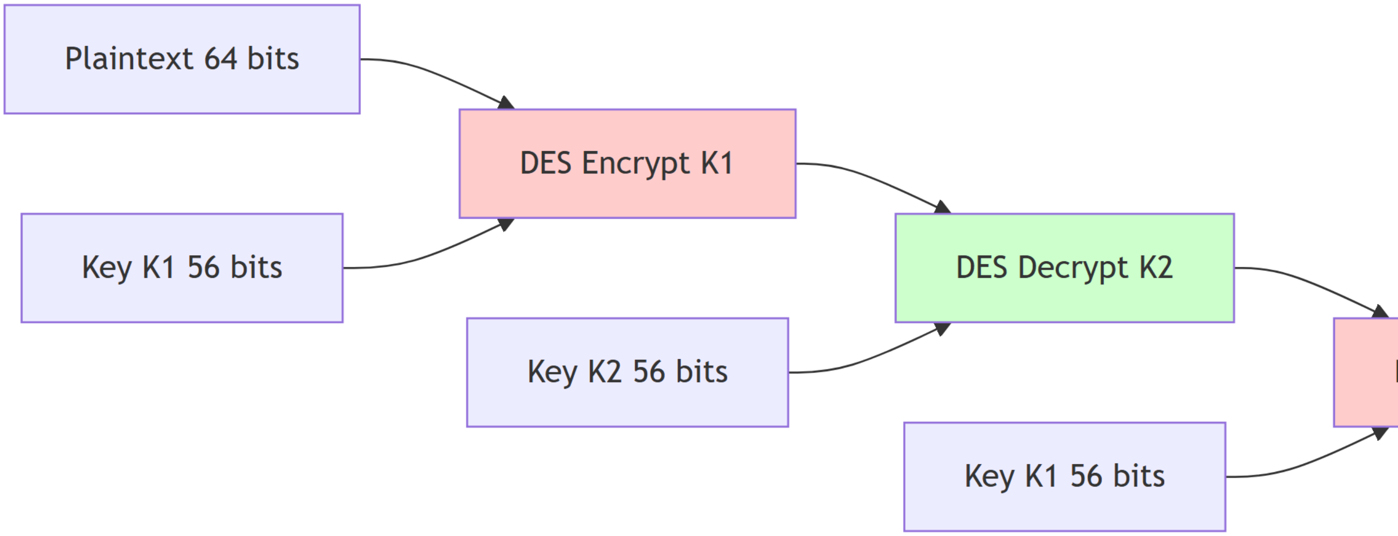
Solution: increase key space to $\{0,1\}^{112}$.

Scheme:

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$

With:

- E: DES encryption
- D: DES decryption
- K_1, K_2: two 56-bit keys



Advantages:

- **Satisfactory security**: key space 2^{112}
- **Compatibility**: reuse of existing DES hardware/software
- **Gradual migration**: while waiting for AES

Disadvantage:

- **Performance**: 3× slower (3 successive DES executions)

DES Properties

1. DES is not a group

DES is NOT a group under composition:

$$\nexists K_3 \text{ such that } E_{K_3}(E_{K_2}(E_{K_1}(x))) = E_{K_3}(x)$$

Consequence: composite encryption (Triple-DES) considerably increases security.

If DES were a group: exhaustive search on $\{0,1\}^{56}$ would break the algorithm regardless of the number of consecutive executions.

2. Weak and semi-weak keys

- **Weak key**: $E_K(E_K(x)) = x$
- **Pair of semi-weak keys**: $E_{K_1}(E_{K_2}(x)) = x$

Characteristic: weak keys generate identical subkeys in pairs:

- $k_1 = k_{16}$, $k_2 = k_{15}$, ..., $k_8 = k_9$
- Facilitates cryptanalysis

DES has 4 weak keys:

Weak key (hexadecimal)
0101 0101 0101 0101
0101 0101 FEFE FEFE
FEFE FEFE FEFE FEFE
FEFE FEFE 0101 0101

And 6 pairs of semi-weak keys

Original text (DES and 3DES)

DES and Triple-DES

- The **size of the key set** ($\{0,1\}^{56}$) constitutes the **greatest threat** weighing on DES with current computing resources. In **1999** it took only **24 hours** to find the key from a **known plaintext** using a **massively parallel brute force technique** (100,000 PCs connected to the Internet).

- **Triple DES** protects us from these **brute force attacks** by increasing the **possible key space** to $\{0,1\}^{112}$.

- This alternative allows continuing to use **DES "boxes"** (hardware and software) while waiting for migration to AES.

- The **security level** obtained by this solution is **very satisfactory**.

- The **performance impact** of three successive DES executions remains a **disadvantage** for some applications.

DES: properties

- **DES is not a group** (in the algebraic sense) under composition: In other words, DES being a permutation: $\{0,1\}^{64} \to \{0,1\}^{64}$, if DES were a group under composition, this would mean that: $\exists K_3$ such that $E_{K_3}(E_{K_2}(x)) = E_{K_3}(x)$

  This property ensures that **composite encryption** (like Triple-DES) **considerably increases the security** of DES. If DES were a group, exhaustive search on the possible key set ($\{0,1\}^{56}$) would allow "breaking" the algorithm **regardless of the number of consecutive executions** of DES.

- **Weak and semi-weak keys** (weak and semi-weak keys):
  - A key K is said to be **weak** if $E_K(E_K(x)) = x$.
  - A pair of keys (K_1, K_2) is said to be **semi-weak** if $E_{K_1}(E_{K_2}(x)) = x$.
- Weak keys have the particularity of generating **identical subkeys in pairs** ($k_1 = k_{16}$, $k_2 = k_{15}$, ..., $k_8 = k_9$), which **facilitates cryptanalysis**.
- **DES has 4 weak keys** (and 6 pairs of semi-weak keys).

Quick revision (3DES and security)

DES vulnerability: key space 2^{56} breakable in 24h (1999). **Triple-DES**: $E_{K_1}(D_{K_2}(E_{K_1}(P)))$, space 2^{112}, reuses DES hardware, 3× slower. **DES group** → composite encryption strengthens security. **4 weak keys** generating identical subkeys in pairs → facilitates cryptanalysis.

---

6. Advanced Encryption Standard (AES)

General Presentation

AES (Advanced Encryption Standard): standard adopted in November 2001.

Design: Johan Daemen and Vincent Rijmen (original name: **Rijndael**)

Main characteristics:

- **Type**: iterative block cipher (but **NOT a Feistel Cipher**)
- **Block size**: 128 bits
- **Variable key size**: 128, 192 or 256 bits
- **Number of rounds**: depends on key size
  - 10 rounds for 128-bit key
  - 12 rounds for 192-bit key
  - 14 rounds for 256-bit key
- **Usage modes**: ECB, CBC, CFB, OFB, CTR

Advantages over DES:

- **Open process**: consultation and analysis by worldwide experts
- ~**2× more performant** in software
- ~**10^{22} times more secure** (theoretically)

- **Scalable**: key size can be increased if necessary

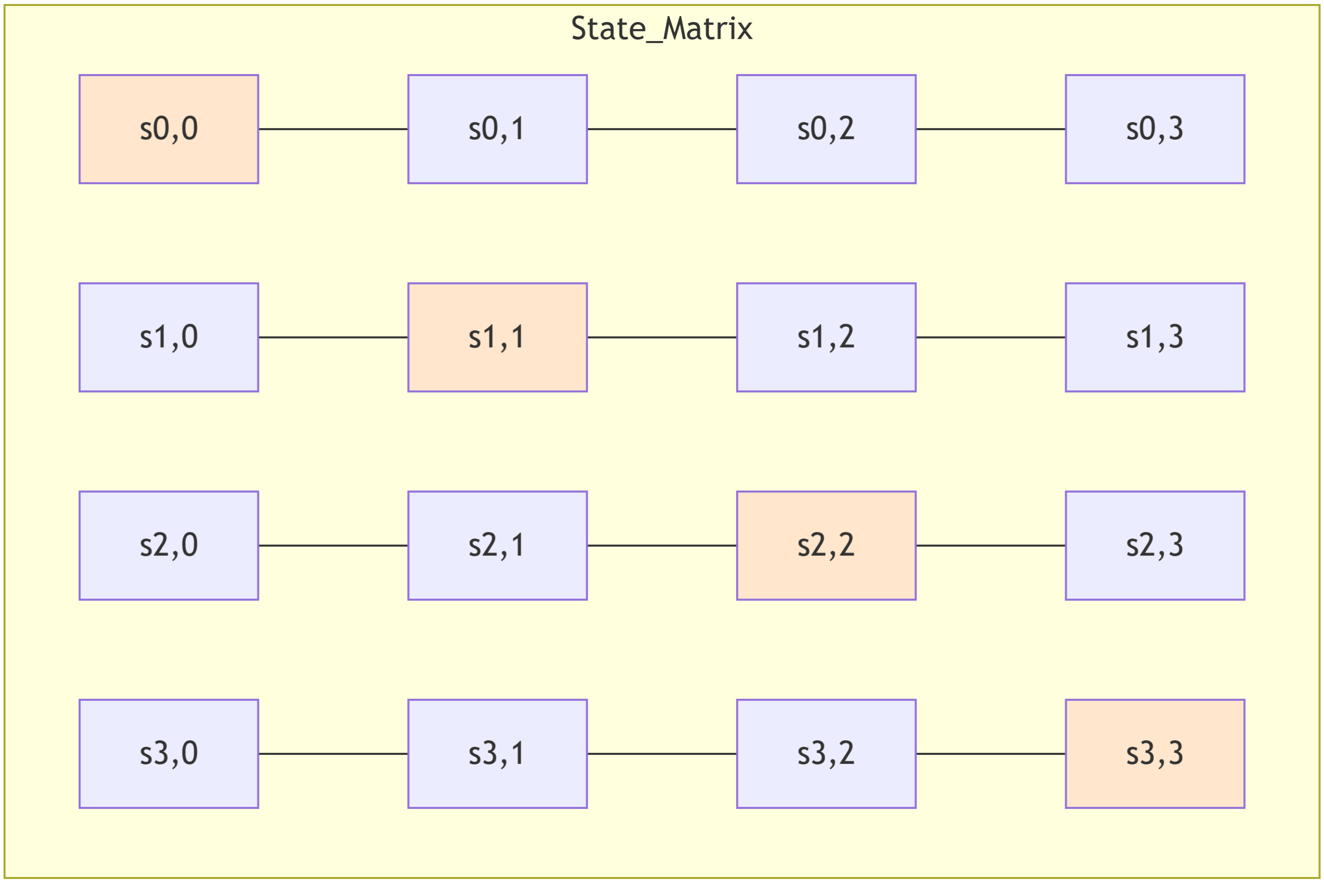
AES Structure

Basic unit: **State** matrix of 4 rows × 4 columns (for 128-bit key)

- Each element = 1 byte
- **Total**: 16 bytes = 128 bits

Operations on field $GF(2^8)$:

- Byte = element of $GF(2^8)$
- Finite field of polynomials of degree 7 with coefficients in $GF(2)$
- Additions, multiplications defined in $GF(2^8)$



s0,0	s0,1	s0,2	s0,3
s1,0	s1,1	s1,2	s1,3
s2,0	s2,1	s2,2	s2,3
s3,0	s3,1	s3,2	s3,3

AES Round Detail

Four operations per round:

1. SubBytes (ByteSub)

- Non-linear substitution via **S-box**
- Each byte transformed independently
- Resistance to linear and differential cryptanalysis

2. ShiftRows

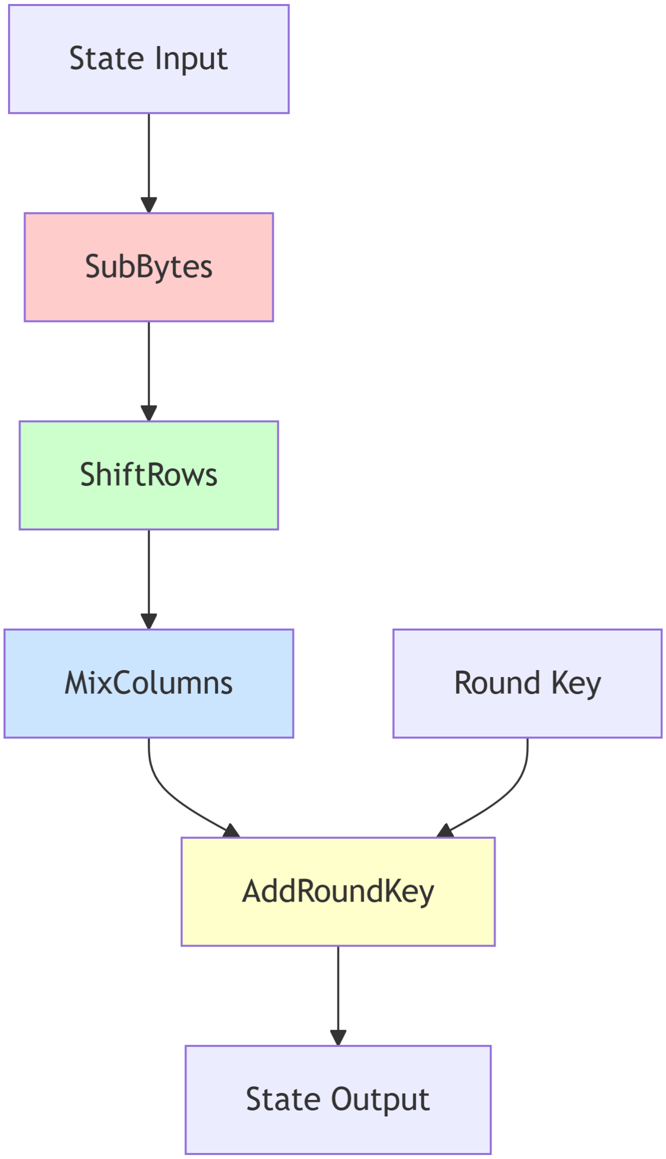
- **Permutation of bytes** with variable shifts per row
- Row 0: no shift
- Row 1: left shift 1 position
- Row 2: left shift 2 positions
- Row 3: left shift 3 positions

3. MixColumns

- Each column = linear combination of other columns
- **Matrix multiplication** in $GF(2^8)$
- Maximum diffusion

4. AddRoundKey

- **XOR** of the State matrix with the round subkey
- Subkey = result of Key Schedule



Final round: identical EXCEPT **no MixColumns**

Key Schedule (Subkey Generation)

Process:

1. **Key Expansion**: generation of an extended matrix
   - Key 128 bits → matrix $4 \times 4 \times (N_e + 1)$ bytes
   - N_e = number of rounds

2. **Key Selection**: extraction of subkeys
    - First subkey: first 4 columns
    - Second subkey: next 4 columns
    - Etc.

Operations:

- Byte rotations
- Substitutions via S-box
- XOR with constants (Rcon)

AES Pseudo-code

```
Rijndael(State, CipherKey) {
    KeyExpansion(CipherKey, ExpandedKey);  // Key Schedule

    AddRoundKey(State, ExpandedKey[0..3]); // Initial XOR

    for(i = 1; i < Ne; i++) {
        Round(State, ExpandedKey[4*i...(4*i)+3]);
    }

    FinalRound(State, ExpandedKey[4*Ne...4*Ne+3]);  // Without MixColumns
}
```

AES Decryption

Principle: apply the **inverse operations** in each round.

Inverse operations:

- **InvSubBytes**: inverse substitution via S-box ^{1}
- **InvShiftRows**: right shifts (instead of left)
- **InvMixColumns**: inverse matrix multiplication
- **AddRoundKey**: self-inverse (XOR)

Order: inverse of encryption with subkeys in reverse order

Original text (AES)

Advanced Encryption Standard (AES)

- Adopted as **standard in November 2001**, designed by **Johan Daemen and Vincent Rijmen** (hence its original name **Rijndael**).

- It is also an **iterative block cipher** (like DES) but **not a Feistel Cipher**.
- **Plaintext/Ciphertext Blocks**: **128 bits**.
- **Variable key length**: **128, 192, or 256 bits**.
- Unlike DES, AES comes from an **open consultation and analysis process** involving worldwide experts.
- Techniques similar to DES (substitutions, permutations, XOR…) complemented by **simple and very performant algebraic operations**.
- All operations are performed in the **field** $GF(2^8)$: the finite field of **polynomials of degree 7** with **coefficients in** $GF(2)$.
- In particular, a **byte for AES is an element in** $GF(2^8)$ and the **operations on bytes** (additions, multiplications,…) are **defined as in** $GF(2^8)$.
- ~**2 times more performant** (in software) and ~**10^{22} times (in theory…) more secure** than DES…
- **Scalable**: The key size can be increased if necessary.

Detail of an AES Step (round)

The **basic unit** on which calculations are applied is a **matrix of 4 rows and 4 columns** (in the case of a 128-bit key) whose elements are **bytes**:

- **ByteSub**: **Non-linear operation (S-box)** designed to **resist linear and differential cryptanalysis**.
- **ShiftRow**: **Permutation of bytes** introducing **variable shifts** on the rows.
- **MixColumn**: Each column is replaced by **linear combinations** of the other columns (**matrix multiplication** !)
- **AddRoundKey**: **XOR** of the current matrix with the **subkey** corresponding to the current step.

AES: Global Operation

- The **number of steps** of AES varies depending on the **key size**. For a **128-bit key**, **10 steps** must be performed. Each increase of 32 bits in the key size entails an **additional step** (14 steps for 256-bit keys).
- **Decryption** consists of applying the **inverse operations** in each of the steps (**InvSubBytes**, **InvShiftRows**, **InvMixColumns**). **AddRoundKey** (because of XOR) is **its own inverse**.

- The **Key Schedule** consists of:
  - An operation of **key expansion** of the main key. If N_e is the number of steps (depending on the key), a **matrix of 4 rows and 4** $\times$ **(**N_e **+ 1) columns** is generated.
  - An operation of **step key selection**: The **first subkey** will be constituted by the **first 4 columns** of the matrix generated during expansion and so on.

Quick revision (AES)

AES (Rijndael 2001): iterative block cipher (NOT Feistel), 128-bit blocks, keys 128/192/256 bits → 10/12/14 rounds.
State: 4×4 byte matrix in $GF(2^8)$.
4 operations/round:

- SubBytes (non-linear S-box)
- ShiftRows (row shifts)
- MixColumns (linear combinations)
- AddRoundKey (XOR subkey).

2× faster than DES, 10^{22} times more secure.

---

7. Attacks and AES Security

AES Strengths

Simplicity and performance:

- Simple and efficient algorithm
- Works on limited platforms (8-bit smart cards)
- Hardware and software optimizations

Published Attacks

1. Algebraic attacks (2002)

XSL technique (N. Courtois and P. Pieprzyk):

- Represents AES as **system of 8000 quadratic equations** with 1600 binary unknowns
- **Estimated effort**: 2^{100} operations (still a conjecture)

- **Characteristic**: requires few known plaintexts
- **Distinction**: different from linear/differential attacks

Critique: based on the “highly algebraic” character of AES (largely contested)

2. Related Key Attacks (2009-2011)

Principle: attacks based on **similar keys**

- Interesting results on **reduced versions** of AES
- Do not compromise full AES

3. Side Channel Attacks

Principle: attacks on **implementation** (not the algorithm)

Techniques:

- **Cache timing attacks**: cache access analysis
- **Power analysis**: power consumption
- **Electromagnetic analysis**: electromagnetic emissions

Example (2005): Osvik, Shamir, Tromer

- Extraction of 128-bit key with **6-7 plaintext/ciphertext pairs**
- Based on **cache access** analysis

4. Meet in the Middle on biclique structures (2011-2015)

Result:

- Reduces effort for AES-128 to 2^{126} (factor 4 vs brute force)
- **Remains well above** current capabilities

Practical Security

Fundamental assumption: key of **maximum entropy**

Recent attacks (WPA2, etc.):

- Exploit **weakness of passwords/passphrases**
- No flaw in AES itself
- Problem: key generation from weak passwords

Critical reminder: key quality = system security

Original text (AES Attacks)

AES: Final Remarks and Attacks (I)

- The greatest **strength of AES** lies in its **simplicity** and its **performance**, including on **reduced computing capacity platforms** (e.g. **smart cards** with 8-bit processors).
- Since its official publication, **many cryptanalysis works** have been published with very interesting results. In particular, **N. Courtois and P.Pieprzyk** presented a technique called **XSL** allowing to represent AES as a **system of 8000 quadratic equations** with **1600 binary unknowns**. The **effort needed** to break this system is estimated (it is still a **conjecture**...) to be 2^{100}.
- These attacks are based on the **highly algebraic character** (and largely contested...) of AES. Moreover, only **a few known plaintexts** are needed to set them up, which distinguishes them from linear and differential attacks.
- In recent years (2009-2011) **attacks based on similar keys** (related key attacks) have obtained interesting results on **reduced versions** of AES.
- Another family of attacks called **side channel attacks** acting directly on the **algorithm implementation** allows extracting cryptographically relevant information during encryption execution.

AES: Final remarks and Attacks (II)

- In **2015** a **Meet in the Middle** type attack based on **biclique structures** showed that it was possible to reduce the **effort needed** to find an AES-128 key to 2^{126}, i.e., a **factor of 4** compared to brute force. This nevertheless remains **well above** current computing capabilities.
- Another family of attacks called **side channel attacks** acting directly on the **algorithm implementation** allows extracting cryptographically relevant information during encryption execution. In particular, the authors manage to **extract the 128-bit key** with only **6-7 plaintext/ciphertext pairs** based on **cache accesses**.
- The **security of AES** (as for any other encryption algorithm) is always based on the assumption of a **key of maximum entropy**. The **attacks published recently** on protocols based on AES (like WPA2) exploit the **weakness of passwords/passphrases** that are the origin of the keys used.

Quick revision (AES Security)

Strengths: simplicity, performance (even 8-bit cards). **Attacks**: XSL (2^{100}, algebraic), related keys (reduced versions), side channel (implementation, cache), Meet-in-Middle biclique (2^{126}). **Security**: assumption of max entropy key. Practical attacks = weak passwords, not AES flaw.

---

8. Block Cipher Cryptanalysis Techniques

8.1 Differential Cryptanalysis

Principle: **chosen plaintext** attack analyzing the **propagation of differences** between two plaintexts through the rounds.

Method:

1. Choose two plaintexts with known difference: x_a and x_b
2. Observe propagation: $\Delta x = x_a \oplus x_b$
3. Analyze ciphertexts: $\Delta y = y_a \oplus y_b$
4. **Assign probabilities to keys** according to observed changes
5. **Most probable key** = correct key (after many trials)

Characteristics:

- Requires 2^{47} **chosen plaintext pairs** for DES
- **Probabilities**: depend on S-boxes and structure
- The more pairs increase, the more success probability increases

Sensitivity: very sensitive to **number of rounds**

- Chances of success increase **exponentially** when rounds decrease

8.2 Linear Cryptanalysis

Principle: **known plaintext** attack creating a **linear simulator** of the block cipher.

Method:

1. Create **linear approximations** of the algorithm
2. Analyze a large number of plaintext/ciphertext pairs
3. The bits of the simulator key **tend to coincide** with those of the real key (probabilistic calculation)

Complexity for DES:

- 2^{38} **known plaintexts** → 10% probability of guessing correctly
- 2^{43} **known plaintexts** → 85% success probability

Characteristics:

- **Most powerful analytical attack** to date on block ciphers
- Also **sensitive to number of rounds**

8.3 Differential vs Linear Comparison

Common difficulties:

- **Parallelization**: less efficient than parallel brute force
- **Sensitivity to rounds**: efficiency decreases exponentially with number of rounds

DES and these attacks:

- Widespread conjecture: DES designers **knew these attacks** (1970s, unpublished at the time)
- **S-box design**: very high resistance to both techniques

8.4 Meet-in-the-Middle Attack

Principle: exploits **composite constructions** of type $y = E_{K_2}(E_{K_1}(x))$.

Method:

1. Build list L_1: $L_1 = \{E_{K_1}(x) \mid K_1 \in \text{KeySpace}\}$
2. Build list L_2: $L_2 = \{D_{K_2}(y) \mid K_2 \in \text{KeySpace}\}$
3. Identify **repeated elements** in L_1 and L_2
4. Verify hypothesis with **second known plaintext**
5. The associated keys K_1 and K_2 are probably the sought keys

Example for DES:

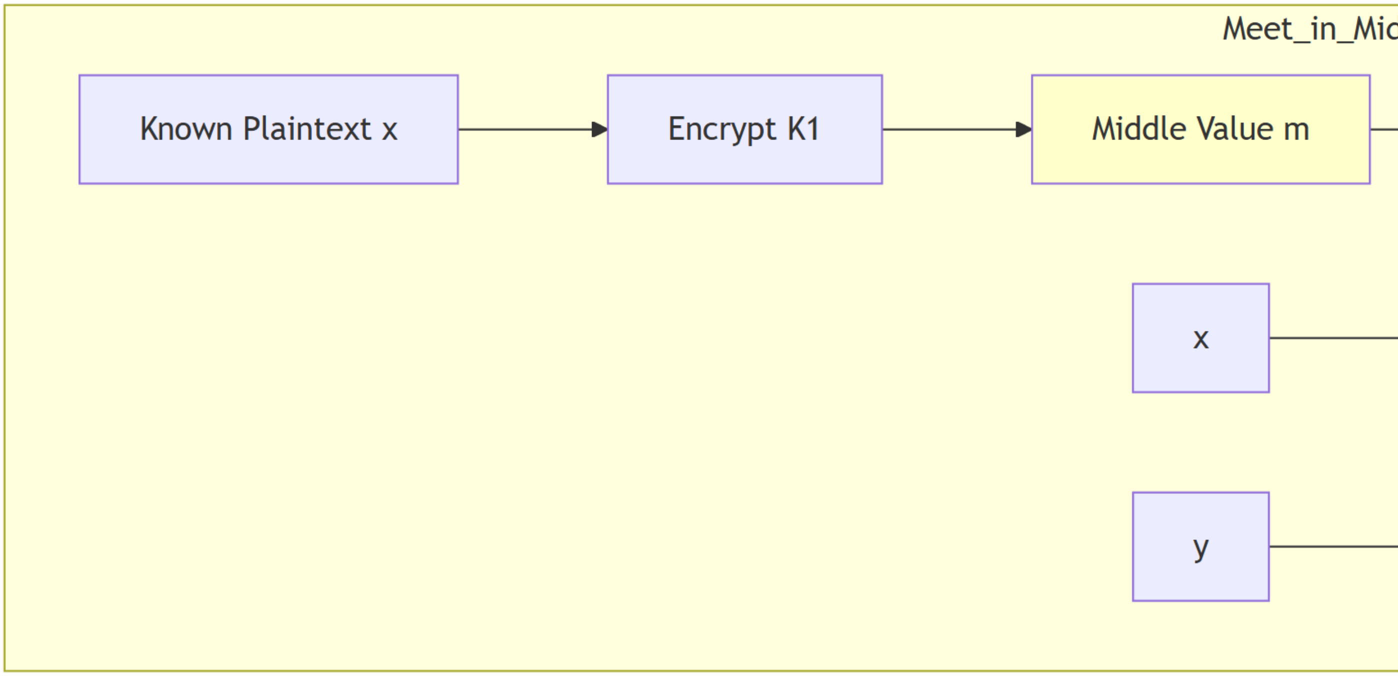
Intuitive key space for $E_{K_2}(E_{K_1}(x))$: $\{0,1\}^{112}$

Actual effort:

- 2^{57} **operations** to establish the two lists
- 2^{56} **blocks** of 64 bits storage
- **Significantly lower** than the intuitive 2^{112}

Applications:

- Attacks on **composite constructions**
- **Internal** cryptanalysis of block ciphers



Original text (Cryptanalysis)

Block Cipher Cryptanalysis Techniques

Differential Cryptanalysis

- This is a **chosen plaintext attack** that focuses on the **propagation of differences** in two plaintexts as they evolve through the different steps of the algorithm.
- It **assigns probabilities to keys** it “guesses” based on the **changes** they induce on the ciphertexts. The **most probable key** has a good chance of being the correct key after a **large number** of plaintext/ciphertext pairs.
- Requires 2^{47} **chosen plaintext pairs** (for DES) to obtain correct results.

Linear Cryptanalysis

- This is a **known plaintext attack** that creates a **block simulator** from **linear approximations**. By analyzing a **large number** of plaintext/ciphertext pairs, the **bits of the simulator key** tend to **coincide** with those of the analyzed block cipher (**probabilistic calculation**)

- For DES an attack based on this technique requires 2^{38} **known plaintexts** to obtain a probability of **10%** of guessing correctly and 2^{43} **for 85%** !
- It is the **most powerful analytical attack** to date on block ciphers.

Block Cipher Cryptanalysis Techniques (II)

- The practical implementation of **differential and linear attacks** presents **difficulties in parallelizing** calculations compared to an exhaustive key search.
- These two attacks are **very sensitive to the number of steps** of the block cipher: chances of success increase **exponentially** as the number of algorithm steps decreases.
- A widespread conjecture among cryptographers is that these attacks, at the time **unpublished**, were **known to the designers of DES**. In particular, the **design of the S-boxes** offers a **very high resistance** to both techniques.

Meet-in-the-Middle Attack

- Applies to constructions of the type $y := E_{K_2}(E_{K_1}(x))$. For DES, the key space for this solution would be $\{0,1\}^{112}$. First build **two lists** L_1 and L_2 of 2^{56} messages of the form: $L_1 = E_{K_1}(x)$ and $L_2 = D_{K_2}(y)$ with E and D the encryption and decryption operations respectively. Then **identify elements that repeat** in both lists and **verify our hypothesis** with a second known plaintext. The K_1 and K_2 associated with this pair of known plaintexts will (in all likelihood) be **the sought keys** !
- **Effort required** to carry out the attacks (for DES): 2^{57} **operations** to establish the two lists + 2^{56} **blocks** of 64 bits of storage to memorize intermediate results... **significantly lower** than the intuitive 2^{112}...
- These meet-in-the-middle techniques are also applied to the **internal cryptanalysis** of block ciphers.

Quick revision (Cryptanalysis)

Differential: chosen plaintext, difference propagation, probabilities on keys, 2^{47} pairs (DES).
Linear: known plaintext, linear approximations, 2^{38}-2^{43} plaintexts (DES), most powerful attack.
Meet-in-Middle: composite constructions, 2 lists 2^{56}, effort 2^{57} « 2^{112}.
Sensitivity: very dependent on number of rounds.