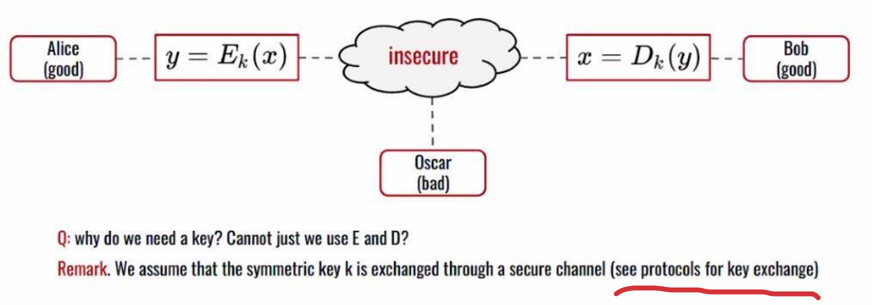
1. Symmetric cipher



Cipher: algorithm for performing encryption or decryption

Example:

Shift cipher (Caesar's cipher) → shift letters of the alphabet by k position

$$y_i = E_k(x_i) = (x_i + k) \mod 26$$
$$x_i = D_k(y_i) = (y_i - k) \mod 26$$

Substitution cipher: map each letter to a different one

Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher: ZEBRASCDFGHIJKLMNOPQTUVWXY

How to break cipher?

- Brute force -> try every key, for shift 26 combinations, for substitution 2^88 combinations.
- Letter Frequency Analysis ->statistical property of letters in an alphabet and generalize it to most used words.

To avoid it: increase space key and make substitution to appear random

Perfect cipher

given a plaintext space = {0,1}^n, D known, ciphertext with probability that exist k: Dk(y)=x equal to: P[x|y] = P[x]

In other words, the ciphertext doesn't reveal any info:

$Pr[x|y] = Pr[x \wedge y] \,/\, Pr[y]$ (conditional probability)
$Pr[x \wedge y] = Pr[x|y]\, Pr[y] = Pr[y|x]\, Pr[x]$ (Bayes)
$Pr[x \wedge y] = Pr[x]\, Pr[y]$ (if x and y are independent)

:

$$Pr[x]\, Pr[y] = Pr[y|x]\, Pr[x] \Rightarrow Pr[y] = Pr[y|x]$$

important: ciphertext should not reveal nothing to the plaintext

One-Time-Pad (OTP)

Plaintext space :
{0,1}^n Key space :
{0,1}^n

$$y = E_k(x) = x \oplus k$$
$$x = D_k(y) = y \oplus k = (x \oplus k) \oplus k = x \oplus (k \oplus k) = x \oplus 0$$

OTP is a perfect cipher: → doesn't reveal anything about plaintext.

- TRNG
- Key is used only once

Problem: |k|=|x|, requires TRNG, too much keys --> It's unpractical; use it just for something secure

We can't use a keystream twice because we will have the input images overlapped

Shannon's Theorem:

Def: "A cipher cannot be perfect if the size of its key space is less than the size of its message space"

Proof by contradiction.

$$2^{|k|} < 2^{|x|}$$

$$Pr[y_0] > 0 \qquad \text{(ciphertext must exists)}$$

$$S = \{D_k(y_0) : k \in K\} \qquad \text{(K is the set of all possible keys)}$$

Then: $\exists\, x \text{ such that } x \notin S$

If we know: $\forall\, k \in K : E_k(x) \neq y_0 \Rightarrow Pr[y_0] = 0$

2 main approaches

1. **Stream ciphers**: inspired by OTP, given a key, generate a keystream, encrypt/decrypt using XOR
2. **block ciphers**: split message in blocks, encrypt/decrypt each block, different "operation mode"

Stream ciphers

|x|==|s|

Encryption: $y_i = E_{s_i} = x_i \oplus s_i = x_i + s_i \bmod 2$
Decryption: $x_i = D_{s_i} = y_i \oplus s_i = y_i + s_i \bmod 2$

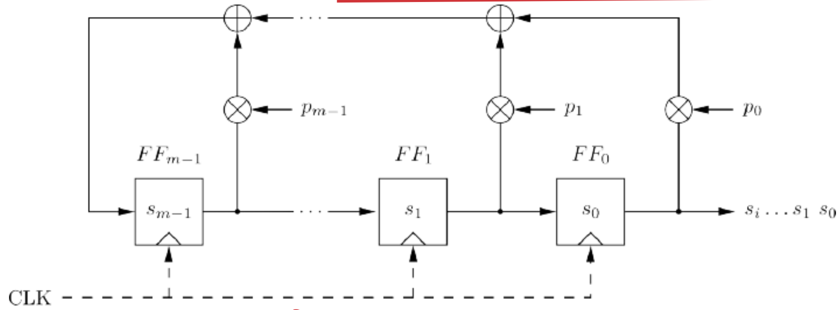
Are called:

- synchronous: Si is a function of the key
- asynchronous: when Si is a function of the key and previous bits of

y Notice that modulo 2 is equivalent to a bitwise XOR operation.

A5/1 (GSM) and LFSR

A5/1 based on three LFSR (Linear feedback Shift Registers):



$$s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \bmod 2 \qquad p_j \in \{0,1\},\ s_i \in \{0,1\}$$

Or as a polynomial:

$$P(x) = x^m + p_{l-1}x^{m-1} + \dots + p_1 x + p_0$$

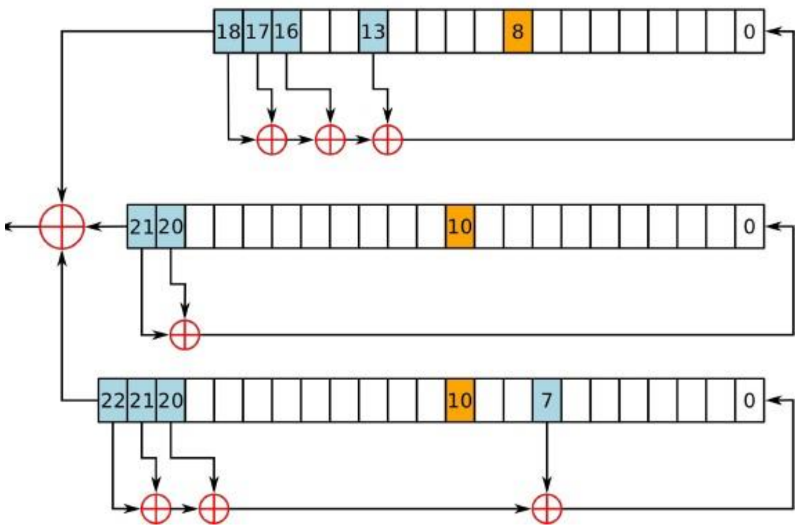
Composed by Flip Flops (which can store 1 bit of information). Each FF: if CLK=1, then FF stores the input IN, emitting the stored value into OUT (even when CLK=0). pi enable/disable feedback line (switch variable).

Attack to single LFSR:
recover the first 2m-1 bits of the keystream (known):
$s_i = y_i + x_i \bmod 2$ recover the other bits of the keystream:

$$\begin{array}{llll}
i=0, & s_m & \equiv p_{m-1}s_{m-1} + \ldots + p_1 s_1 + p_0 s_0 & \bmod 2 \\
i=1, & s_{m+1} & \equiv p_{m-1}s_m + \ldots + p_1 s_2 + p_0 s_1 & \bmod 2 \\
\vdots & \vdots & \vdots\ \vdots & \vdots \\
i=m-1, & s_{2m-1} & \equiv p_{m-1}s_{2m-2} + \ldots + p_1 s_m + p_0 s_{m-1} & \bmod 2
\end{array}$$

These are m equations: we can solve this system and recover pj. A5/1 used 3 LSFR (much stronger).

LFSR number	Length in bits	Feedback polynomial	Clocking bit	Tapped bits
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8	13, 16, 17, 18
2	22	$x^{22} + x^{21} + 1$	10	20, 21
3	23	$x^{23} + x^{22} + x^{21} + x^{8} + 1$	10	7, 20, 21, 22



Clocking bit is used to determine if CLK is enabled based on a majority rule.

RC-4: Ron's Code (Rivest, most well-known cryptanalyst)

Broken in 1994, synchronous, variable key length, very fast to compute; from the key generate a random stream, eventually it will repeat but the period is long (>10^100). break it when used in TLS protocol.

- Key scheduling algorithm (KSA)

S is an array of 256 integers.

```
int j = 0;
for(int i = 0; i < 256; i++)
    S[i] = i;
for(int i = 0; i < 256; i++) {
    j = (j + S[i] + key[i % len]) % 256;
    swap(&S[i], &S[j]);
}
```

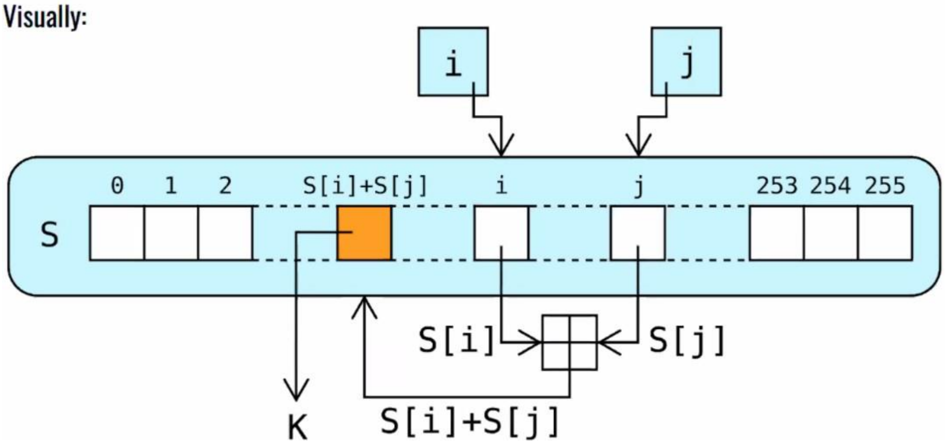
PERMUTATIONS

At the end S is a permutation of {0, ..., 255} generated based on the value of the key.

- Pseudo-Random Generation Algorithm (PRGA)

```
int i = 0, j = 0;
for(size_t n = 0, len = strlen(plaintext); n < len; n++) {
    i = (i + 1) % 256;
    j = (j + S[i]) % 256;
    swap(&S[i], &S[j]);
    int K = S[(S[i] + S[j]) % 256];   // keystream byte
    ciphertext[n] = K ^ plaintext[n]; // XOR encryption
}
```

Visually:

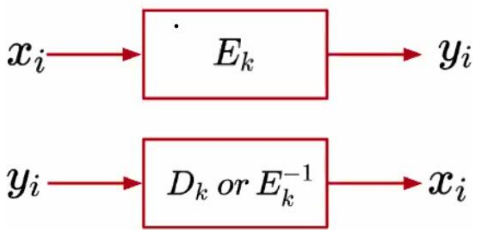


We need a way to randomize each keystream with an initialization vector (IV); btw are weak to attack due to an unsafe combination of IV and key.

Block ciphers

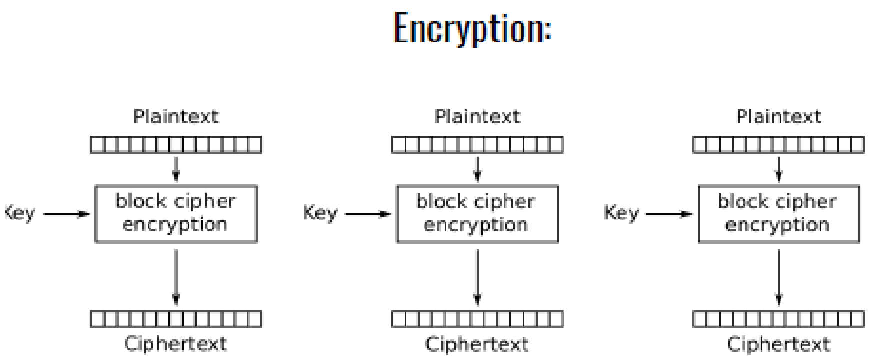
Community is switching to block ciphers (e.g.: DES, Blowfish, AES)

takes as input a block of fixed size, a key fixed, key length can be different to block length.



If the message is larger than the block size:

- Electronic Code Block (ECB): (simple, efficient, parallel)



The Decryption is the

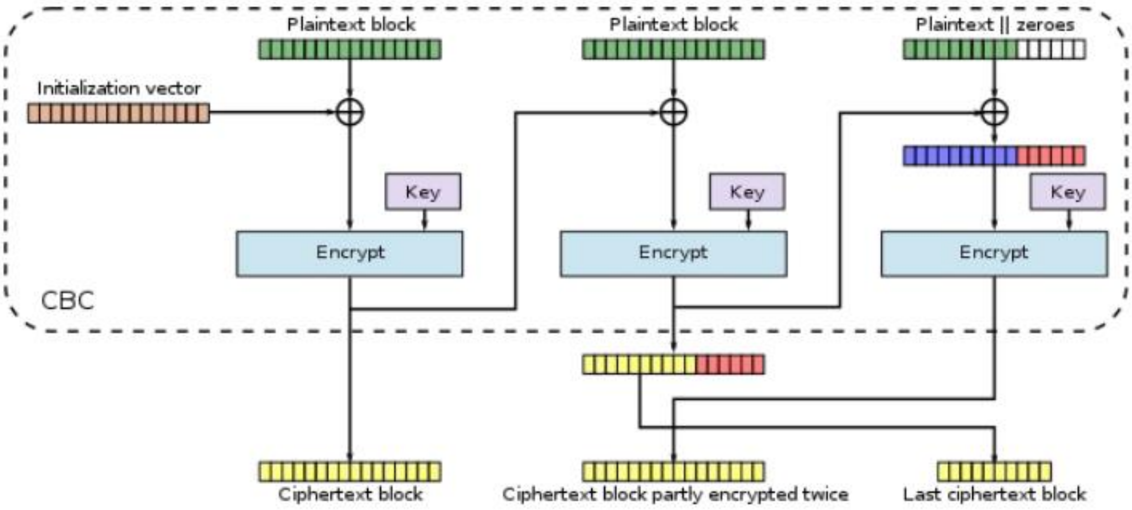
opposite ECB: attacks

- **Blocks reordering**
- **Blocks can be replaced, removed, appended:** problem for the integrity of message
- **The same message encrypted twice, will produce the same ciphertext**

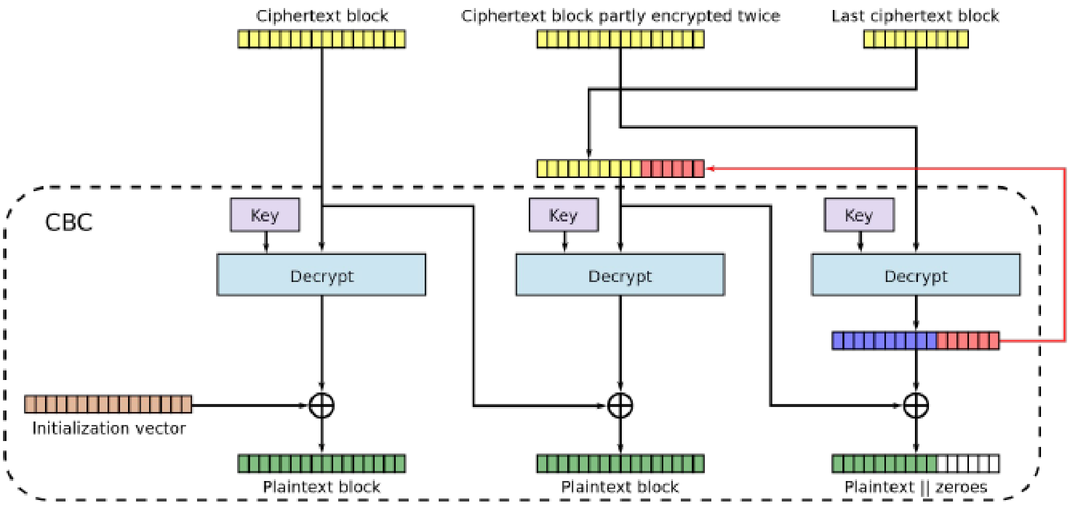
- **Cipher Block Chaining (CBC):**
  Chain of block, randomized with **IV** (different for each message, can be public), encryption **no parallel**, decryption **Parall**

  - error propagation: If one bit flipped in x_i all subsequent blocks are affected
  - if one bit is flipped in yi-1 then xi is affected in an unpredictable manner, while x_i in a predictable manner. This could be exploited by an attacker. Hence use CRC/etc.

  It can be seen as an asynchronous stream cipher; message must be padded (using of padding or ciphertext stealing



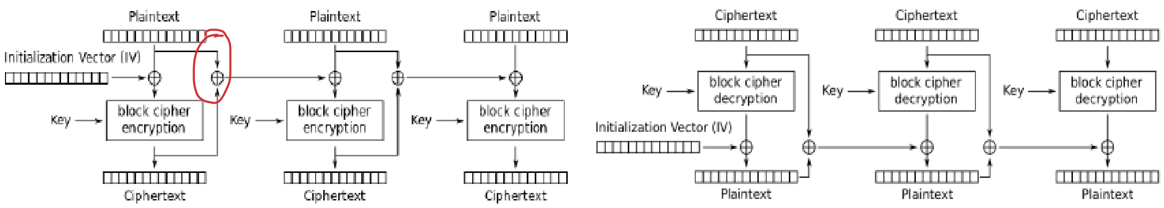
Swap the last cipher blocks, then truncate the ciphertext to the original length of the plaintext



Swap the last cipher blocks, decrypt, then truncate the plaintext to the original length of the ciphertext

- Propagating Cipher Block Chaining (PCBC)

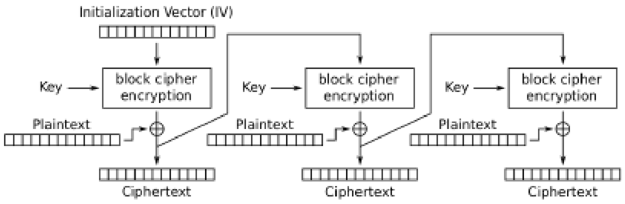
Encryption: Decryption:



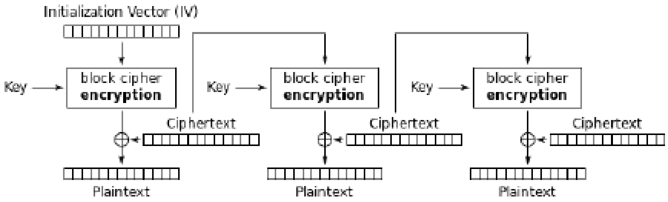
designed to propagate small changes to all subsequent blocks both during encryption and decryption; if two adjacent blocks are exchanged, subsequent decrypted blocks are not affected.

- Cipher FeedBack (CFB)

Encryption:



Decryption:

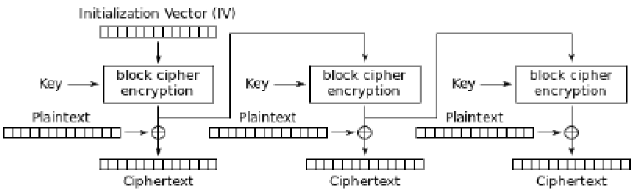


- asynchronous stream cipher
- error propagation in encryption
- encryption is not parallelizable
- encryption algorithm is both used in encryption and decryption

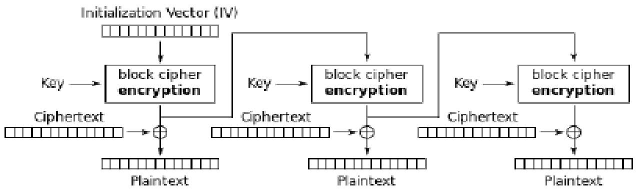
- decryption can be parallelized
- one bit error in ciphertext blocks, affect two plaintext blocks, other blocks are fine
- no need of padding

- Output Feedback (OFB)

Encryption:



Decryption:



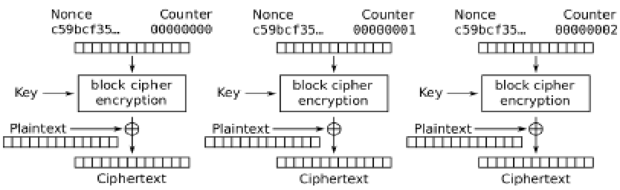
- synchronous stream cipher
- one bit flip in ciphertext affect only one bit in the output (this helps using error correction codes)

- encryption is not parallelizable
- decryption is not parallelizable
- encryption algorithm is both used in encryption and decryption

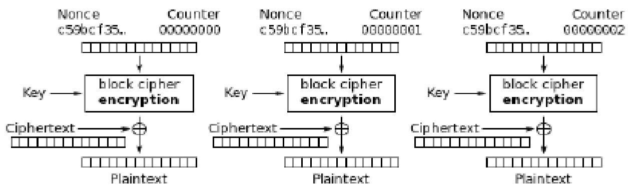
Problems: if function and key are public → IV must be secret (otherwise, can be sent in clear)

- Counter Mode (CTR)

Encryption:



Decryption:



- synchronous stream cipher
- Besides IV (nonce), it uses a counter that is incremented for each block

- encryption is parallelizable
- decryption is parallelizable
- encryption algorithm is both used in encryption and decryption

	ECB Electronic Code Book	CBC Cipher Block Chaining	CFB Output Feedback	OFB Cipher Feedback	CTR Counter
Information leakage	High	low	low	low	low
Encryption parallelizable	Yes	No	No	No	Yes
Decryption parallelizable	Yes	Yes	Yes	No	Yes
Ciphertext manipulation	Yes	No	No	No	No
Precompute	No	No	No	Yes	Yes
Encryption error propagation	No	Yes	Yes	No	No
Decryption error propagation	No	Partial (2 Blocks)	Partial (2 Blocks)	No	No

Initialization Vector

Dummy block to kick off the process.

It mustn't be used twice.

For CBC and CFB the reuse can leak some info

In CBC must be unpredictable

In OFB and CTR, the reuse can destroy the security

CBC with PKCS#7 padding scheme

PKCS#7 padding scheme: the value of each added byte is the number of bytes that are added

Padding can be used by attacker for returning malicious code; to avoid it we put a feedback to check whether after the decryption the message is valid (*padding oracle*).

- Padding oracle attack against CBC + PKCS#7

  In CBC the encryption of a block x_i is performed as:

  $$y_i = E_k(x_i \oplus y_{i-1})$$

  while the decryption:

  $$x_i = D_k(y_i) \oplus y_{i-1}$$

  $$x_i = D_k(E_k(x_i \oplus y_{i-1})) \oplus y_{i-1}$$

  Assuming the cipher is correct:

  $$x_i = x_i \oplus y_{i-1} \oplus y_{i-1}$$

-

- To decrypt y_i the attacker builds a new ciphertext with two blocks:
    - Y'= Y'_0 || Y_i
    - Choose randomly Y'_0 in [0,255]
    - Oracle will compute: $x'_i = D_k(y_i) \oplus y'_0$
    - And checks whether the padding scheme is respected
    - Anyway, the first block must be decrypted with IV
    - we can expect than there is at least one value where the oracle will give OK since we will have: $x'_i[k] = 1$ (k is the last byte)

Now, the attacker can compute $x_i[k]$:

$$x'_i[k] = D_k(y_i)[k] \oplus y'_0[k]$$
$$x'_i[k] = (x_i \oplus y_{i-1})[k] \oplus y'_0[k]$$
$$x'_i[k] = x_i[k] \oplus y_{i-1}[k] \oplus y'_0[k]$$
$$x_i[k] = x'_i[k] \oplus y_{i-1}[k] \oplus y'_0[k]$$

We have that $x_i'[k] = 1$, $y_{i-1}[k]$ is known (a block from the ciphertext!), and $y_0'[k]$ is known (decided by the attacker!). Hence, he can compute $x_i[k]$.

The attacker can now iterate this process (for k-1) to get other bytes in the same block; Now he needs to find a y0' such that xi'[k] = 2 and xi'[k-1] = 2 (2 as an example). He tries 256 till he succeeds.

After computing the last byte (as seen before), we can check whether our assumption on xi'[k] was correct: it is enough to XOR the last second byte with 0x1 and check if the padding scheme is still valid (if it is not valid than xi'[k] is different from one and our assumption is wrong, we need to test another value with the padding oracle).

However, it`s not possible to decrypt the first block.

To prevent this attack: NO ADDING A PADDING ORACLE.

Shannon suggested two operations for a cipher:

- **Confusion:** every bit of the ciphertext should depend on the key, obscuring the connection (with substitution).
- **Diffusion:** changing a single bit of the plaintext, then half of the bits in the ciphertext should change (with permutation).

Avalanche effect: the combination of the two

Data Encryption Standard (DES)

Key length: 56 bits very weak, can be broken in less than 24 hours using "linear attack"

Block size 64 bits; based on Feistel Network (F does not have to be invertible); used till 1997

DES can be broken with 2 approaches:

1. Move to AES
2. Try to fix DES reusing hw and sw implementation (not so good)

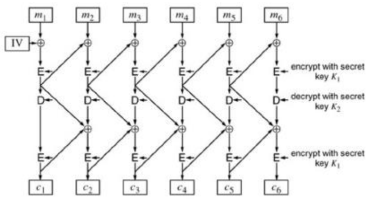
We can make DES stronger increasing the number of the keys:

- EEE mode: $y = E_{k_1}(E_{k_2}(E_{k_3}(x)))$
  E.g. 3-DES

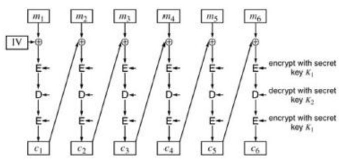
- EDE mode: $y = E_{k_1}(D_{k_2}(E_{k_1}(x)))$
  - Very common as it requires only two keys.
  - With two keys the complexity is only 2^57 (not 2^112)
  - 3-DES thus has 2^112 key strength

EDE:

- inside CBC



- outside CBC



1. Bit Flipping:

- **CBC Outside:** one bit flip in the ciphertext causes that block of plain text and next block garbled ⇒ Self-Synchronizing (i.e., after some garbled blocks, you get correct blocks)
- **CBC Inside:** one bit flip in the ciphertext causes more blocks to be garbled

2. Pipelining: more pipelining possible in CBC inside implementation

3. Flexibility of Change: CBC outside: can easily replace CBC with other feedback modes (ECB, CFB, ...)

Another idea to make a cipher stronger: key whitening

- XOR-ENCRYPT-XOR
- DES-X: $\text{DES-X}(x) = k_2 \oplus (\text{DES}_k(x \oplus k_1))$
  - Three keys: 56 bits, 64 bits, 64 bits

False positives: when key space is larger than message space:

Given a block cipher with a key length of k bits and block size of n bits, as well as t (plaintext, ciphertext) pairs (x1, y1), ... , (xt , yt), the expected number of false keys which encrypt all plaintexts to the corresponding ciphertexts is: 2k−tn