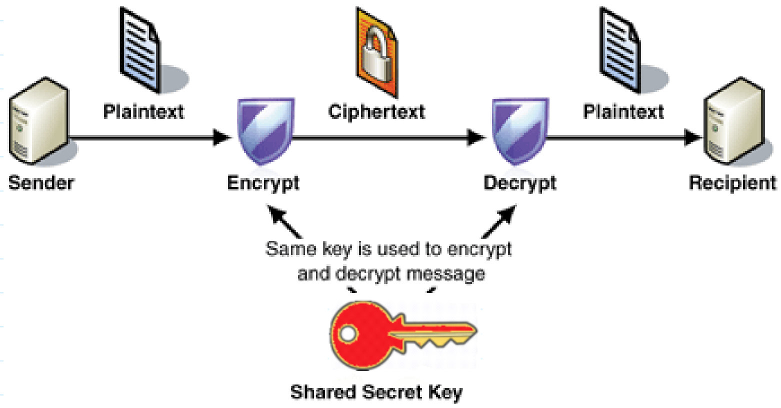
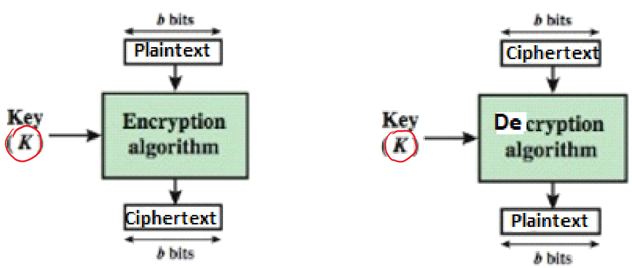


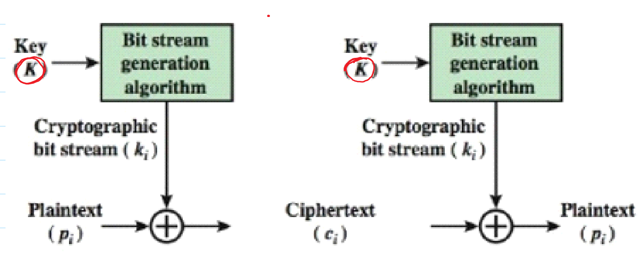
Symmetric ciphers

AES: **Block Ciphers**

128, 192, 256

Stream Ciphers





bitwise XOR

Block cipher AES - 128, 192, 256 --> Encryption --> Decryption NSA

Advanced Encryption Standard ~ 2000

Key length 128, 192, 256, bits: $|k| \in \{128b, 192b, 256b\}$ $2^{256} \approx 10^{80}$

Data to be encripted : message m

| B1 | B2 | B3 | ---- | Bn |

The length of any block Bi should be $|B_i| = 128$ bits

192 bits

256 bits

$|B_i| = |k| = 128$ bits

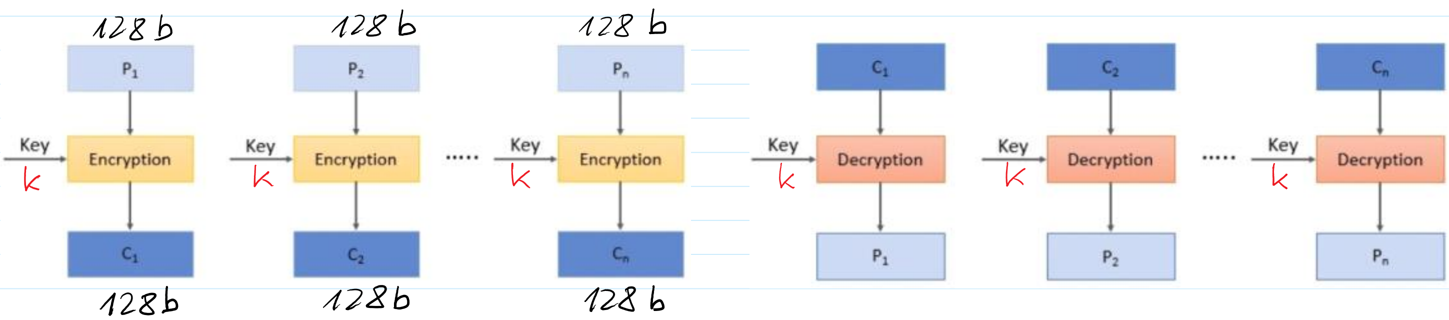
Block Cipher: Electronic Code Book -**ECB** mode of encryption

From <https://binaryterms.com/block-cipher.html>

1. Electronic Code Book (ECB) mode in AES-128

This is considered to be the easiest block cipher mode of operation. In electronic codebook mode (ECB) the plain text is divided into the blocks, each of 128-bit. Each block is encrypted one at a time to produce the cipher block. The same key is used to encrypt each block.

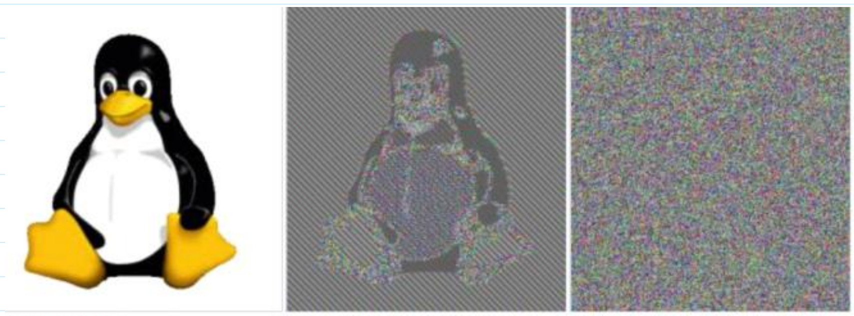
When the receiver receives the message i.e. ciphertext. This ciphertext is again divided into blocks, each of 128-bit and each block is decrypted independently one at a time to obtain the corresponding plain text block. Here also the same key is used to decrypt each block which was used to encrypt each block.



$$EncAES(k, P_1) = C_1$$
$$EncAES(k, P_2) = C_2$$
$$- - - - - - - -$$
$$EncAES(k, P_n) = C_n$$

(a) plaintext

(b) plaintext encrypted in ECB mode using AES



Original image

Encrypted using ECB mode

Modes other than ECB result in pseudo-randomness

https://binaryterms.com/block-cipher.html

2. Cipher Block Chaining - CBC Mode

To overcome the limitation of ECB i.e. the repeating block in plain text produces the same ciphertext, a new technique was required which is Cipher Block Chaining (CBC) Mode. CBC confirms that even if the plain text has repeating blocks its encryption won't produce same cipher block.

To achieve totally different cipher blocks for two same plain text blocks **chaining** has been added to the block cipher. For this, the result obtained from the encryption of the first plain text block is fed to the encryption of the next plaintext box.

In this way, each ciphertext block obtained is dependent on its corresponding current plain text block input and all the previous plain text blocks. But during the encryption of first plain text block, no previous plain text block is available so a random block of text is generated called **Initialization vector**.

Now let's discuss the encryption steps of CBC

Step 1: The initialization vector and first plain text block are XORed and the result of XOR is then encrypted using the key to obtain the first ciphertext block.

Step 2: The first ciphertext block is fed to the encryption of the second plain text block. For the encryption of second plain text block, first ciphertext block and second plain text block is XORed and the result of XOR is encrypted using the same key in step 1 to obtain the second ciphertext block.

Similarly, the result of encryption of second plain text block i.e. the second ciphertext block is fed to the encryption of third plain text block to obtain third ciphertext block. And the process continues to obtain all the ciphertext blocks.

Decryption Steps:

Step 1: The initialization vector is placed in the shift register. It is encrypted using the same key.

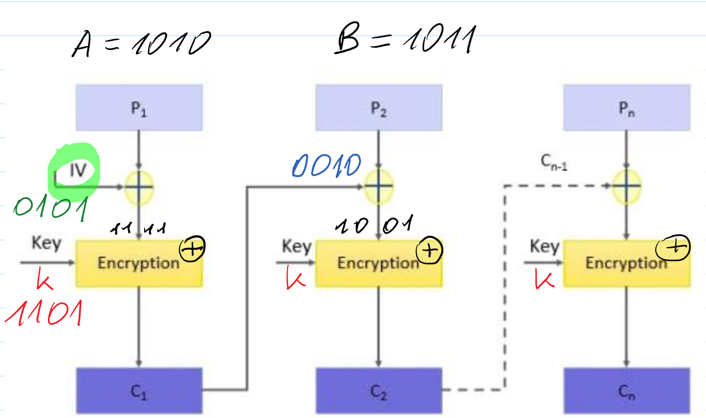
Keep a note that even in the **decryption process** the **encryption** algorithm is implemented instead of the decryption algorithm.

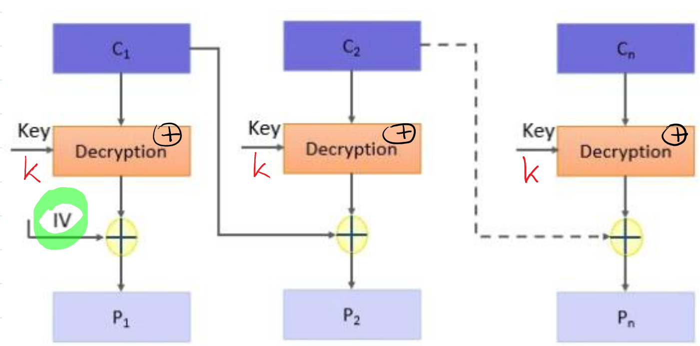
Then from the encrypted IV s bits are XORed with the s bits ciphertext C1 to retrieve s bits plain text P1.

Step 2: The IV in the shift register is left-shifted by s bits and the s bits C1 replaces the rightmost s bits of IV.

The process continues until all plain text fragments are retrieved.

CBC	
Cipher block chaining	
Encryption parallelizable:	No
Decryption parallelizable:	Yes
Random read access:	Yes





$C_1 = P_1 \oplus IV \oplus k$, $C_2 = P_2 \oplus C_1 \oplus k$ ----

$P_1 = C_1 \oplus k \oplus IV$, $P_2 = C_2 \oplus k \oplus C_1$ ---

$\oplus$ 1010 $\oplus$ 1111 $\oplus$ 1011 $\oplus$ 1001

$C_1 \oplus k \oplus IV = P_1 \oplus IV \oplus k \oplus C_1 \oplus k \oplus IV$

$1010 \oplus 0101 = 1111$; $1111 \oplus 1101 = 0010$; $1011 \oplus 0010 = 1001$; $1001 \oplus 1101 = 0100$

$1010 \oplus 0010 = 1000$; $1000 \oplus 1101 = 0101$

$C_1 \oplus k \oplus IV = P_1 \oplus IV \oplus k \oplus C_1 \oplus k \oplus IV$

$= P_1 \oplus IV \oplus IV \oplus k \oplus k = P_1 \oplus 0 \oplus 0$

$0101 \oplus 0101 = 0000$; $1101 \oplus 1101 = 0000$; $1010 \oplus 0000 = 1010 = A$

https://binaryterms.com/block-cipher.html

5. Counter Mode - CTR

It is similar to OFB but there is no feedback mechanism in counter mode. Nothing is being fed from the previous step to the next step instead it uses a sequence of number which is termed as a **counter** which is input to the encryption function along with the key. After a plain text block is encrypted the counter value increments by 1.

Steps of encryption:

Step1: The counter value is encrypted using a key.

Step 2: The encrypted counter value is XORed with the plain text block to obtain a ciphertext block.

To encrypt the next subsequent plain text block the counter value is incremented by 1 and step 1 and 2 are repeated to obtain the corresponding ciphertext.

The process continues until all plain text block is encrypted.

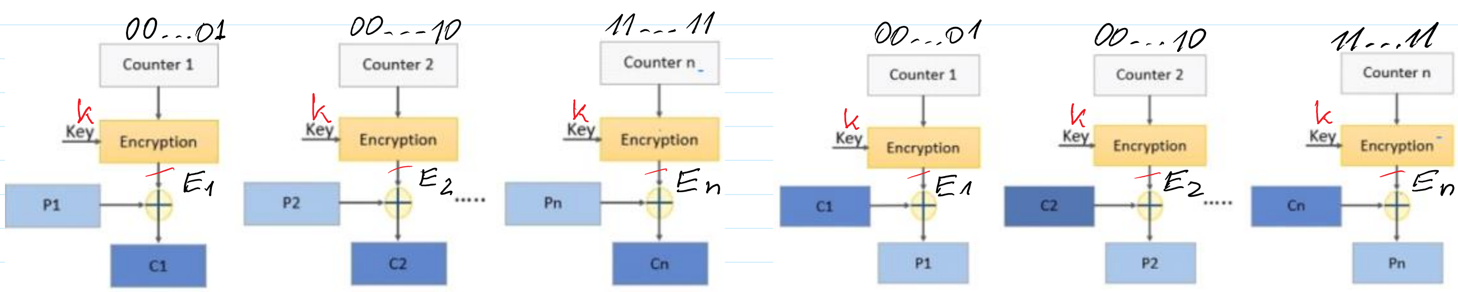
Steps for decryption:

Step1: The counter value is encrypted using a key.

Note: Encryption function is used in the decryption process. The same counter values are used for decryption as used while encryption.

Step 2: The encrypted counter value is XORed with the ciphertext block to obtain a plain text block.

CTR	
Counter	
Encryption parallelizable:	Yes
Decryption parallelizable:	Yes
Random read access:	Yes

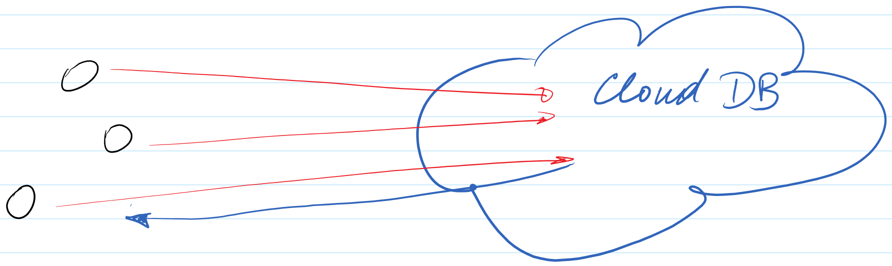


$C_1 = P_1 \oplus E_1$

$C_1 \oplus E_1 = P_1 \oplus E_1 \oplus E_1 = P_1 \oplus 0 = P_1$

IoT

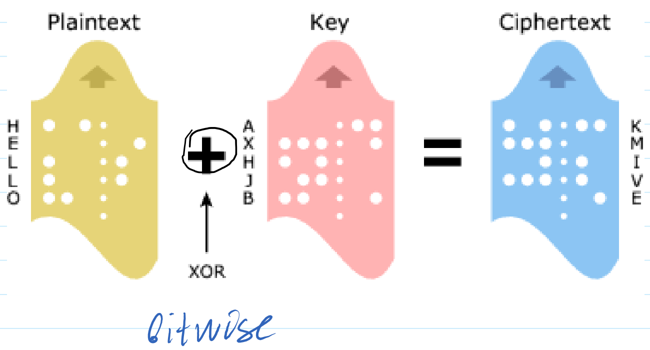
Prevention of Replay attack

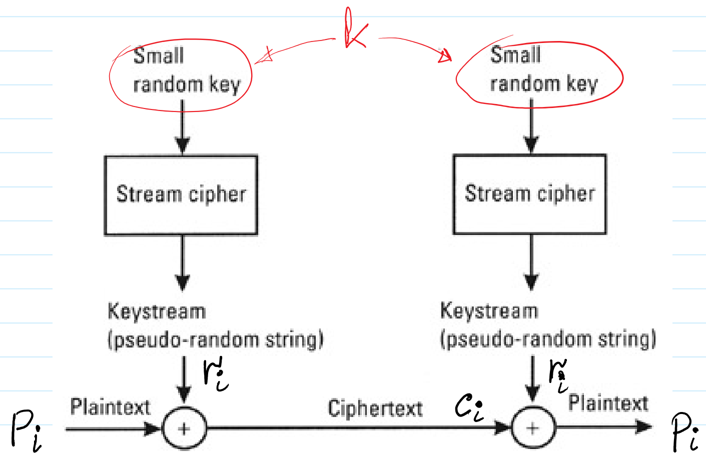


EncAES(k, m) = c

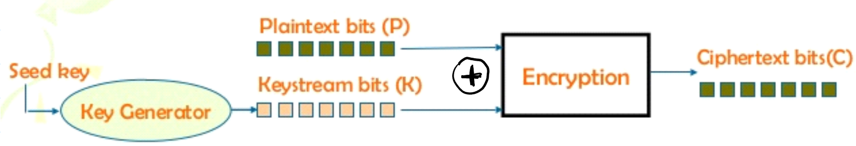
DecAES(k, c) = m

Stream Cipher - Vernam Cipher - One-Time Pad





Stream Ciphers



- To encrypt plaintext stream
  - A random set of bits is generated from a seed key, called keystream which is as long as the message
  - Keystream bits are added modulo 2 to plaintext to form the ciphertext stream
- To decrypt ciphertext stream
  - use the same seed key to generate the same keystream used in encryption
  - Add the keystream modulo 2 to the ciphertext to retrieve the plaintext
  - i.e. $C = P \oplus K \Rightarrow C \oplus K = (P \oplus K) \oplus K = P$

Pseudo Random Numbers Generators - PRNG

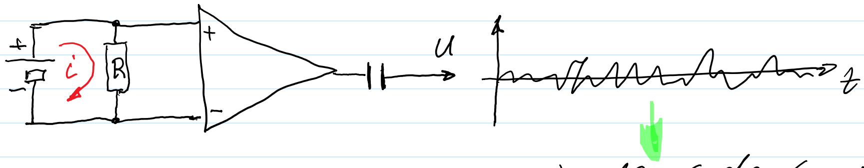
$r_{i+1} = PRNG(r_i)$

$r_1 = PRNG(r_0)$; r_0 - initial value: $r_0 = k$.

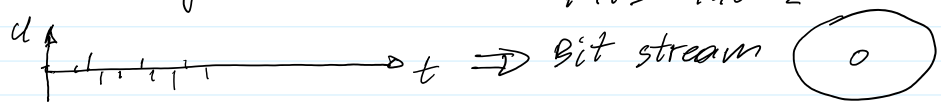
$r_2 = PRNG(r_1)$ Non-linear dynamic systems

— — — — — — — —

$r_n = PRNG(r_{n-1})$



Analog code converter FIPS-140-2



Till this place