IDEA

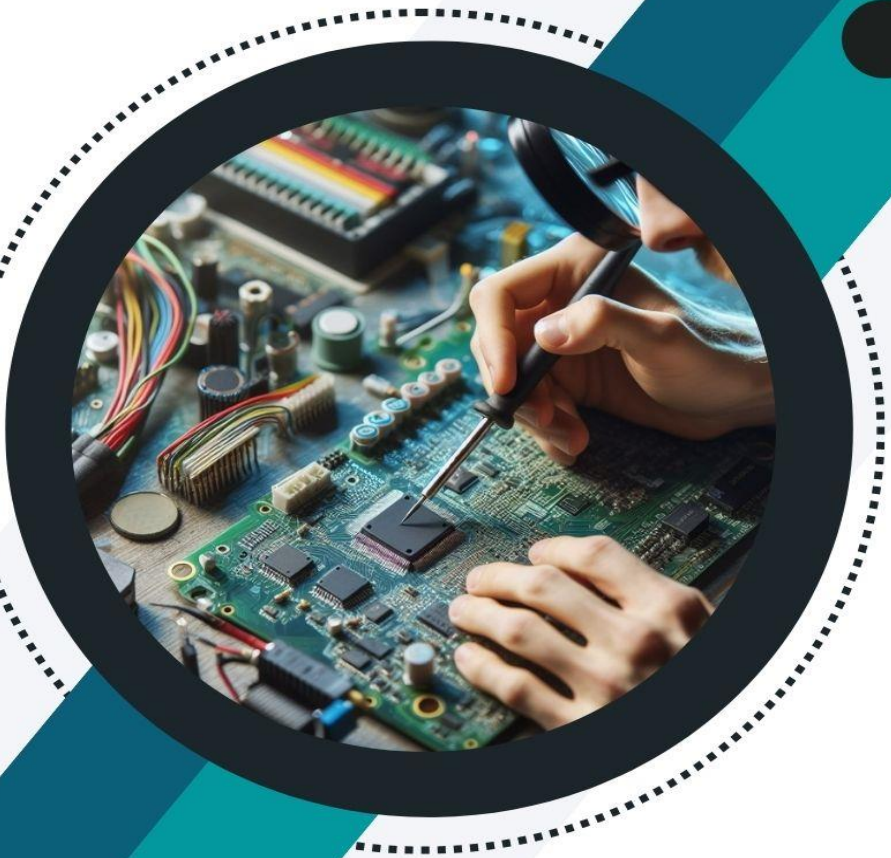
ENSAM
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

INTERNATIONAL DATA ENCRYPTION ALGORITHM

1ÈRE ANNÉE DU CYCLE D'INGÉNIEUR

-

GEM/MSEI



Réalisé par :

- BOUZIANE Hamza
- Khamalla Marouane
- HASNAOUI FOUAD
- CHABBI ABDERAHMANE

Encadré par :

Pr. El Moumni

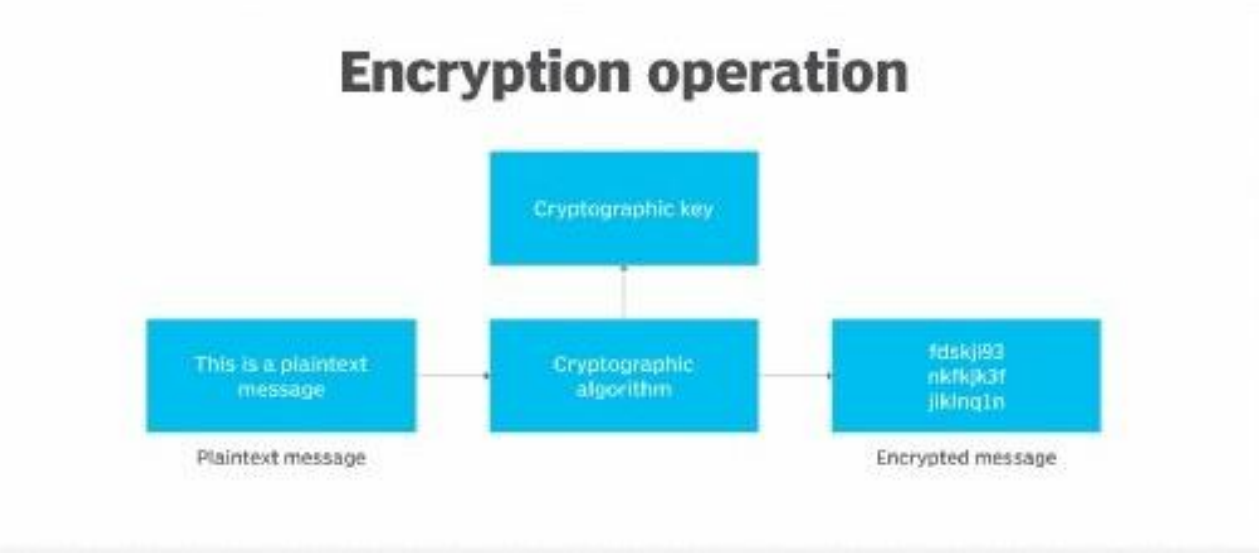
IDEA Encryption

Introduction :

Encryption is crucial for securing sensitive information in digital communication, ensuring that data remains confidential, integral, and authentic. By converting data into a coded format, encryption prevents unauthorized access and alterations, thereby protecting privacy and sensitive information. This security measure is vital across various applications, including online banking, where it safeguards financial transactions; secure email communication, where it ensures the privacy and security of email content; and data storage, where it protects stored data against unauthorized access and breaches. Through these means, encryption maintains the trustworthiness and safety of digital interactions and information storage.

IDEA Encryption Algorithm:

The International Data Encryption Algorithm (IDEA) is a symmetric key block cipher renowned for its robustness and efficiency in securing data. Developed by Xuejia Lai and James Massey in 1991, IDEA uses the same key for both encryption and decryption, making it a reliable choice for protecting sensitive information. The algorithm operates on 64-bit blocks of data and employs a 128-bit key, providing strong security through multiple rounds of complex transformations. Its design makes it resistant to a wide range of cryptanalytic attacks, ensuring the confidentiality and integrity of the encrypted data. IDEA's efficiency and strength have made it a popular choice in various cryptographic applications, including secure email communications and data storage.



Background on IDEA Encryption:

What is IDEA?

The International Data Encryption Algorithm (IDEA) is a symmetric key block cipher encryption algorithm designed to encrypt text to an unreadable format for transmission via the internet. It uses a typical block size of 128 bits and takes 64 bits as an input, 64-bit data. IDEA performs 8.5 identical encryption and decryption rounds using six different subkeys. It uses four keys for output transformation.

Key Size and Block Size:

- The key size of 128 bits provides high security, making it difficult for brute-force attacks.
- The block size of 64 bits balances between security and efficiency.

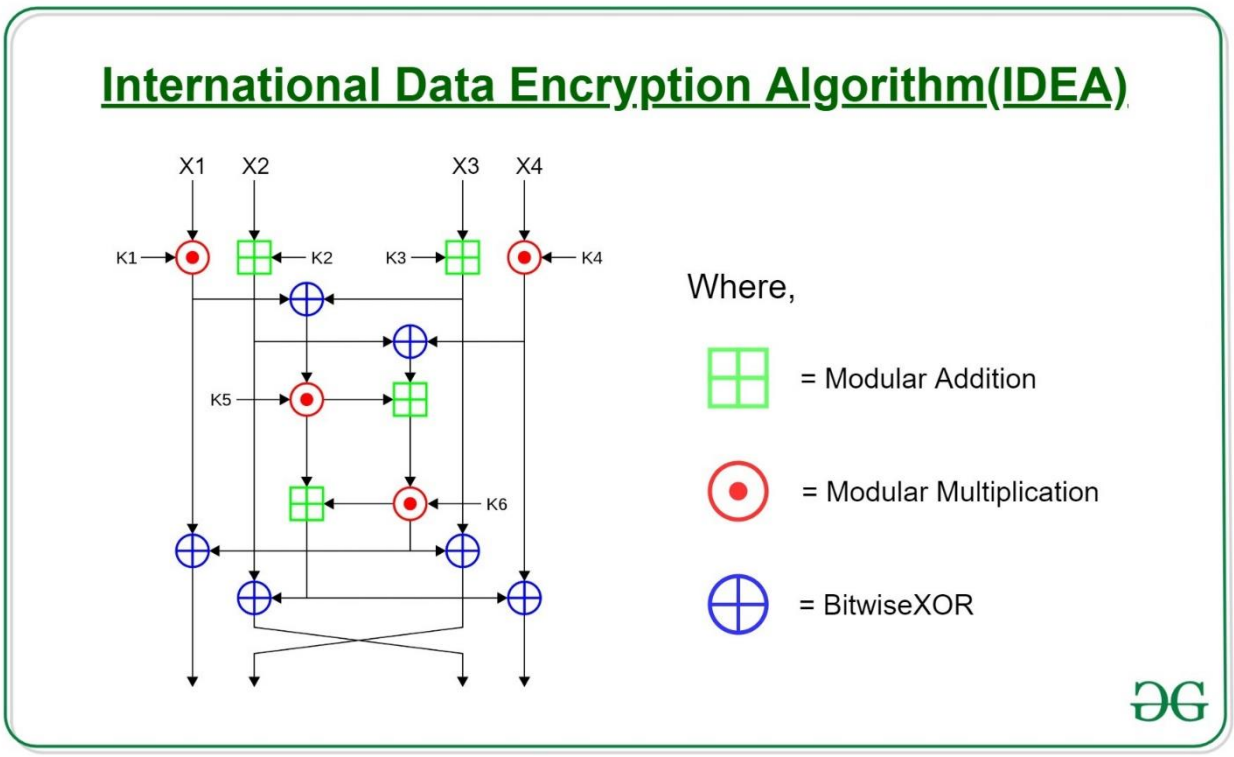
Algorithm's Strengths:

- Resistant to various forms of cryptanalysis, including linear and differential cryptanalysis.
- Provides a good balance of security and performance.

Applications:

- Used in secure data communication.
- Popular in PGP (Pretty Good Privacy) for email encryption.
- Known for its strength against cryptanalysis and efficiency in both software and hardware implementations.

IDEA Algorithm Details



Overall Algorithm Structure:

- The IDEA algorithm processes data in 64-bit blocks using a 128-bit key.
- The algorithm consists of 8 identical rounds, followed by a final transformation round.
- Each round involves a series of modular arithmetic and bitwise operations.

Key Schedule:

- The 128-bit key is expanded into 52 sub-keys, each 16 bits long.
- Sub-Key Generation: The main key is rotated and extracted to generate the sub-keys for each round.

Operations in Each Round:

- Modular Addition: Addition modulo 2^{16}.
- Modular Multiplication: Multiplication modulo $2^{16} + 1$.
- Bitwise XOR: Exclusive OR operation.

The code

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity IDEA_BLOCK is
   generic (
      KEY_SIZE : integer := 128;  -- Taille de la clé en bits
      BLOCK_SIZE : integer := 64   -- Taille du bloc en bits
   );
     port(
                  X1,X2,X3,X4 : in std_logic_vector(BLOCK_SIZE/4 - 1 downto 0);
                  key : in std_logic_vector(KEY_SIZE - 1 downto 0);
                  Y1,Y2,Y3,Y4 : out std_logic_vector(BLOCK_SIZE/4 - 1 downto 0)
     );
end IDEA_BLOCK;

architecture Desc_IDEA_BLOCK of IDEA_BLOCK is
Signal Step1,Step2,Step3,Step4,Step5,Step6,Step7,Step8,Step9,Step10,Step11,Step12,Step13,Step14,Step15: std_logic_vector(BLOCK_SIZE/4 - 1 downto 0);
     begin

                  Step1 <= std_logic_vector((unsigned(X1) * unsigned(key(15 downto 0))) mod (2 ** 16 + 1))(15 downto 0) ;
                  Step2 <= std_logic_vector((unsigned(X2) + unsigned(key(31 downto 16))) mod 2 ** 16 );
                  Step3 <= std_logic_vector((unsigned(X3) + unsigned(key(47 downto 32))) mod 2 ** 16 );
                  Step4 <= std_logic_vector((unsigned(X4) * unsigned(key(63 downto 48))) mod (2 ** 16 + 1) )(15 downto 0);
                  Step5 <= Step1 xor Step3;
                  Step6 <= Step2 xor Step4;
                  Step7 <= std_logic_vector((unsigned(Step5) * unsigned(key(79 downto 64))) mod (2 ** 16 + 1) )(15 downto 0);
                  Step8 <= std_logic_vector((unsigned(Step6) + unsigned(Step7)) mod 2 ** 16 );
```

```
                    Step9 <=std_logic_vector((unsigned(Step8) * unsigned(key(95 downto 80))) mod (2 ** 16 + 1) )(15 downto 0);
                    Step10 <= std_logic_vector((unsigned(Step7) + unsigned(Step9)) mod 2 ** 16 );
                    Y1 <= Step1 xor step9;
                    Y2 <= Step2 xor Step9;
                    Y3 <= Step2 xor Step10;
                    Y4 <= Step4 xor Step10;
end Desc_IDEA_BLOCK;

Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity IDEA is
   generic (
      KEY_SIZE : integer := 128;  -- Taille de la clé en bits
      BLOCK_SIZE : integer := 64   -- Taille du bloc en bits
   );
   port (
      Input : in std_logic_vector(BLOCK_SIZE - 1 downto 0);
      key : in std_logic_vector(KEY_SIZE - 1 downto 0);
      Output : out std_logic_vector(BLOCK_SIZE - 1 downto 0)
   );
end IDEA;

architecture Desc_IDEA of IDEA is
        Signal Intermediary1, Intermediary2,Intermediary3,Intermediary4,Intermediary5,Intermediary6,Intermediary7,Intermediary8:std_logic_vector(BLOCK_SIZE-1
downto 0);
        Signal IntermediaryKey1,IntermediaryKey2,IntermediaryKey3,IntermediaryKey4,IntermediaryKey5,IntermediaryKey6,IntermediaryKey7,IntermediaryKey8:
std_logic_vector(KEY_SIZE - 1 downto 0);
        component IDEA_BLOCK        port( X1,X2,X3,X4 : in std_logic_vector(BLOCK_SIZE/4 -1 downto 0); key : in std_logic_vector(KEY_SIZE - 1 downto 0);
Y1,Y2,Y3,Y4 : out std_logic_vector(BLOCK_SIZE/4 - 1 downto 0) ); end component;
        begin
        B1: IDEA_BLOCK port map(INPUT(15 downto 0),INPUT(31 downto 16), INPUT(47 downto 32),INPUT(63 downto 48),key,Intermediary1(15 downto
0),Intermediary1(31 downto 16), Intermediary1(47 downto 32),Intermediary1(63 downto 48));
        IntermediaryKey1 <= key(KEY_SIZE-26 downto 0) & key(KEY_SIZE-1 downto KEY_SIZE-25);

        B2: IDEA_BLOCK port map(Intermediary1(15 downto 0),Intermediary1(31 downto 16), Intermediary1(47 downto 32),Intermediary1(63 downto
48),IntermediaryKey1,Intermediary2(15 downto 0),Intermediary2(31 downto 16), Intermediary2(47 downto 32),Intermediary2(63 downto 48));
        IntermediaryKey2 <= IntermediaryKey1(KEY_SIZE-26 downto 0) & IntermediaryKey1(KEY_SIZE-1 downto KEY_SIZE-25);

        B3: IDEA_BLOCK port map(Intermediary2(15 downto 0),Intermediary2(31 downto 16), Intermediary2(47 downto 32),Intermediary2(63 downto
48),IntermediaryKey2,Intermediary3(15 downto 0),Intermediary3(31 downto 16), Intermediary3(47 downto 32),Intermediary3(63 downto 48));
        IntermediaryKey3 <= IntermediaryKey2(KEY_SIZE-26 downto 0) & IntermediaryKey2(KEY_SIZE-1 downto KEY_SIZE-25);

        B4: IDEA_BLOCK port map(Intermediary3(15 downto 0),Intermediary3(31 downto 16), Intermediary3(47 downto 32),Intermediary3(63 downto
48),IntermediaryKey3,Intermediary4(15 downto 0),Intermediary4(31 downto 16), Intermediary4(47 downto 32),Intermediary4(63 downto 48));
        IntermediaryKey4 <= IntermediaryKey3(KEY_SIZE-26 downto 0) & IntermediaryKey3(KEY_SIZE-1 downto KEY_SIZE-25);

        B5: IDEA_BLOCK port map(Intermediary4(15 downto 0),Intermediary4(31 downto 16), Intermediary4(47 downto 32),Intermediary4(63 downto
48),IntermediaryKey4,Intermediary5(15 downto 0),Intermediary5(31 downto 16), Intermediary5(47 downto 32),Intermediary5(63 downto 48));
        IntermediaryKey5 <= IntermediaryKey4(KEY_SIZE-26 downto 0) & IntermediaryKey4(KEY_SIZE-1 downto KEY_SIZE-25);

B6: IDEA_BLOCK port map(Intermediary5(15 downto 0),Intermediary5(31 downto 16), Intermediary5(47 downto 32),Intermediary5(63 downto
48),IntermediaryKey5,Intermediary6(15 downto 0),Intermediary6(31 downto 16), Intermediary6(47 downto 32),Intermediary6(63 downto 48));
        IntermediaryKey6 <= IntermediaryKey5(KEY_SIZE-26 downto 0) & IntermediaryKey5(KEY_SIZE-1 downto KEY_SIZE-25);

B7: IDEA_BLOCK port map(Intermediary6(15 downto 0),Intermediary6(31 downto 16), Intermediary6(47 downto 32),Intermediary6(63 downto
48),IntermediaryKey6,Intermediary7(15 downto 0),Intermediary7(31 downto 16), Intermediary7(47 downto 32),Intermediary7(63 downto 48));
        IntermediaryKey7 <= IntermediaryKey6(KEY_SIZE-26 downto 0) & IntermediaryKey6(KEY_SIZE-1 downto KEY_SIZE-25);
```

```
B8: IDEA_BLOCK port map(Intermediary7(15 downto 0),Intermediary7(31 downto 16), Intermediary7(47 downto 32),Intermediary7(63 downto
48),IntermediaryKey7,Intermediary8(15 downto 0),Intermediary8(31 downto 16), Intermediary8(47 downto 32),Intermediary8(63 downto 48));
    IntermediaryKey8 <= IntermediaryKey7(KEY_SIZE-26 downto 0) & IntermediaryKey7(KEY_SIZE-1 downto KEY_SIZE-25);

    Output(15 downto 0) <= std_logic_vector((unsigned(Intermediary8(15 downto 0)) * unsigned(IntermediaryKey8(15 downto 0))) mod (2 ** 16 + 1))(15 downto 0) ;
    Output(31 downto 16) <= std_logic_vector((unsigned(Intermediary8(31 downto 16)) + unsigned(IntermediaryKey8(31 downto 16))) mod 2 ** 16 );
    Output(47 downto 32) <= std_logic_vector((unsigned(Intermediary8(47 downto 32)) + unsigned(IntermediaryKey8(47 downto 32))) mod 2 ** 16 );
    Output(63 downto 48) <= std_logic_vector((unsigned(Intermediary8(63 downto 48)) * unsigned(IntermediaryKey8(63 downto 48))) mod (2 ** 16 + 1) )(15 downto 0);

    end Desc_IDEA;
```

This code implements the IDEA (International Data Encryption Algorithm) encryption algorithm in VHDL. It consists of:

Entity IDEA_BLOCK:

Inputs: Four input data blocks (X1, X2, X3, X4) and a key (key).
Outputs: Four output data blocks (Y1, Y2, Y3, Y4).

Generic parameters: KEY_SIZE and BLOCK_SIZE determine the size of the key and block.
Functionality: Computes a series of operations on the input blocks and key to generate the output blocks.

Architecture Desc_IDEA_BLOCK:

Signals: Step1 to Step15 are intermediate signals representing the steps of the IDEA algorithm.
Process: Performs various arithmetic operations on the input data and key according to the IDEA algorithm.
Output assignment: Assigns the calculated values to the output ports.

Entity IDEA:

Inputs: A single input data block (Input) and a key (key).
Outputs: A single output data block (Output).
Generic parameters: KEY_SIZE and BLOCK_SIZE determine the size of the key and block.

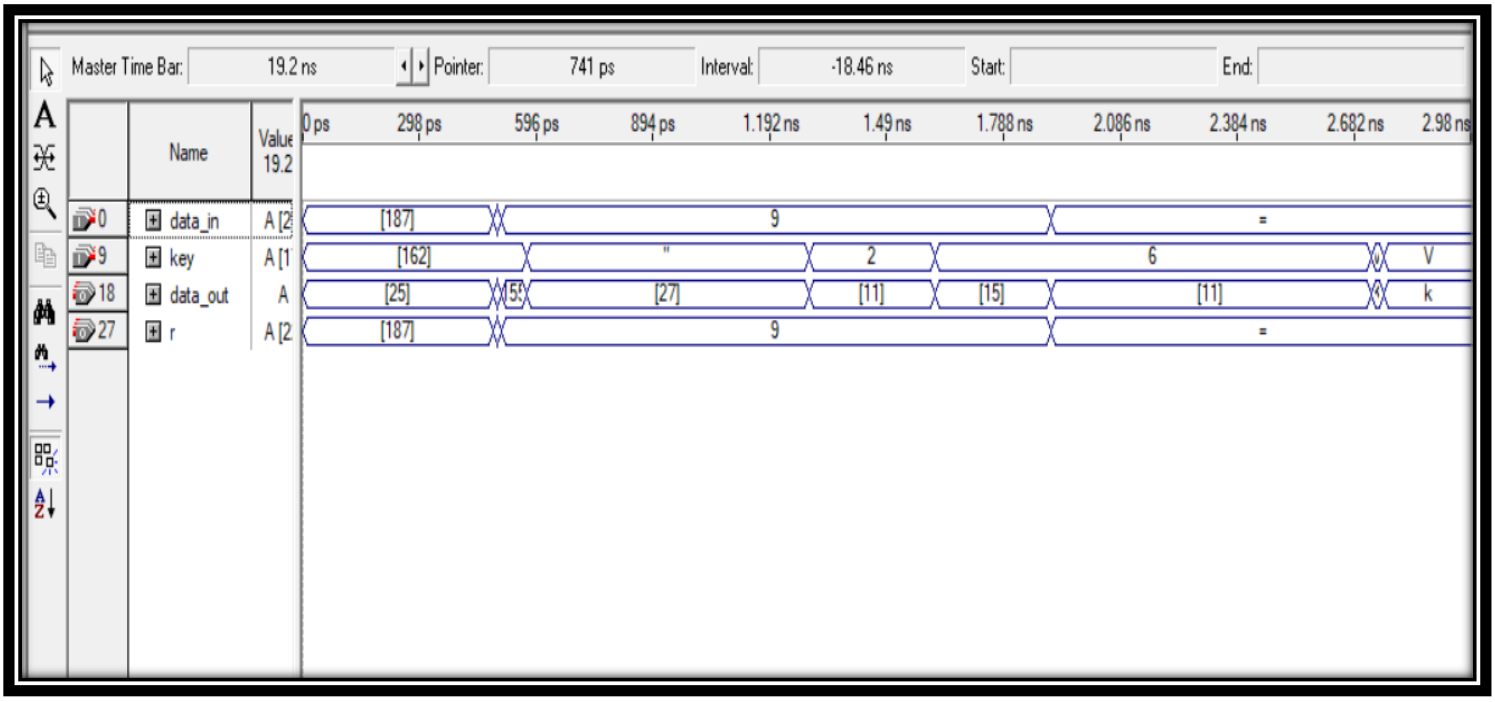
Architecture: Implements the IDEA algorithm by instantiating eight instances of the IDEA_BLOCK component and connecting them serially.
Functionality: Encrypts or decrypts the input data using the IDEA algorithm.

Architecture Desc_IDEA:

Signals: Intermediaries and IntermediaryKeys are arrays containing intermediate values and keys, respectively.
Component instantiation: Eight instances of IDEA_BLOCK are instantiated and connected serially, with each instance using the output of the previous instance as input.

Output assignment: Computes the final output based on the output of the last instance of IDEA_BLOCK.
Overall, this code defines the IDEA encryption/decryption algorithm and implements it in a modular and reusable manner.

simulation



Nous avons tenté de créer un chiffrement où un signal en entrée produit un signal en sortie, et le déchiffrement de ce signal de sortie renvoie le signal initial(r). Après avoir chiffré et déchiffré, nous avons observé que le signal d'entrée est identique au signal obtenu après déchiffrement, validant ainsi le bon fonctionnement de l'algorithme.