

## **1. Introduction**

The symmetrical encryption algorithms are very important field in computer systems security, primarily in securing data transfers over unsecured networks. Most current cryptographic systems are a block cipher systems because they offer a very high security level and a relatively short running time. In addition, the architecture of these systems is well suited for hardware implementation. In this chapter, we propose a new design of symmetric cryptographic block cipher algorithm that deals with the problems of resource-limited systems. Then, we will present in detail the techniques and principles used in this design.

## **2. The proposed algorithm**

The aim of this work is to present a improvement for the 512-bit AES, new light design called High Light weight Encryption Standard (HLES) that can be used when higher level of security is required, and processing power, memory space and bandwidth are limited (case of multimedia transmissions and mobile systems). The HLES algorithm use a key size of 512 bit and data block size of 512 bit. Both of key and data block are divided to 4 blocks of 128 bit for each. Each set of those four 128-bit blocks will be encrypted using an encryption functions that produce a set of four 128-bits encrypted blocks using four 128-bits encryption keys. The encryption process use each one of the four key parts to encrypt the four data block parts respectively but not in same time so that the memory consummation can be minimized. The four encrypted parts are not independents because each part needs the previous one to continue the encryption process (except the first one because it has no previous part).

In what follows, we will detail the encryption function  $E: M (128 \times 4 \text{ bits}) \rightarrow C (128 \times 4 \text{ bits})$ , and then we will present the decryption function  $D$ .

### **2.1. The Encryption Algorithm (E)**

The proposed algorithm is a symmetrical block design. It has four main different transformations: The byte substitution, The SH-Z transformation, The K-XOR transformation, and The L-XOR transformation. The final transformation has the same role of the previous one (XOR function) but this one is applied with the current part of data block and result of previous part.

### 2.1.1. General Schemes of the encryption algorithm

The placement of functions is designed and chosen carefully to achieve the best possible confusion and diffusion through substitutions and permutations, and to guarantee a lower cost of computational resources and good performance. In addition, its architecture provides high level of security because of the 512-bit key length and a special coordination and synchronization between its functions. The encryption process and functions placements are shown as a flowchart in Fig III.1:

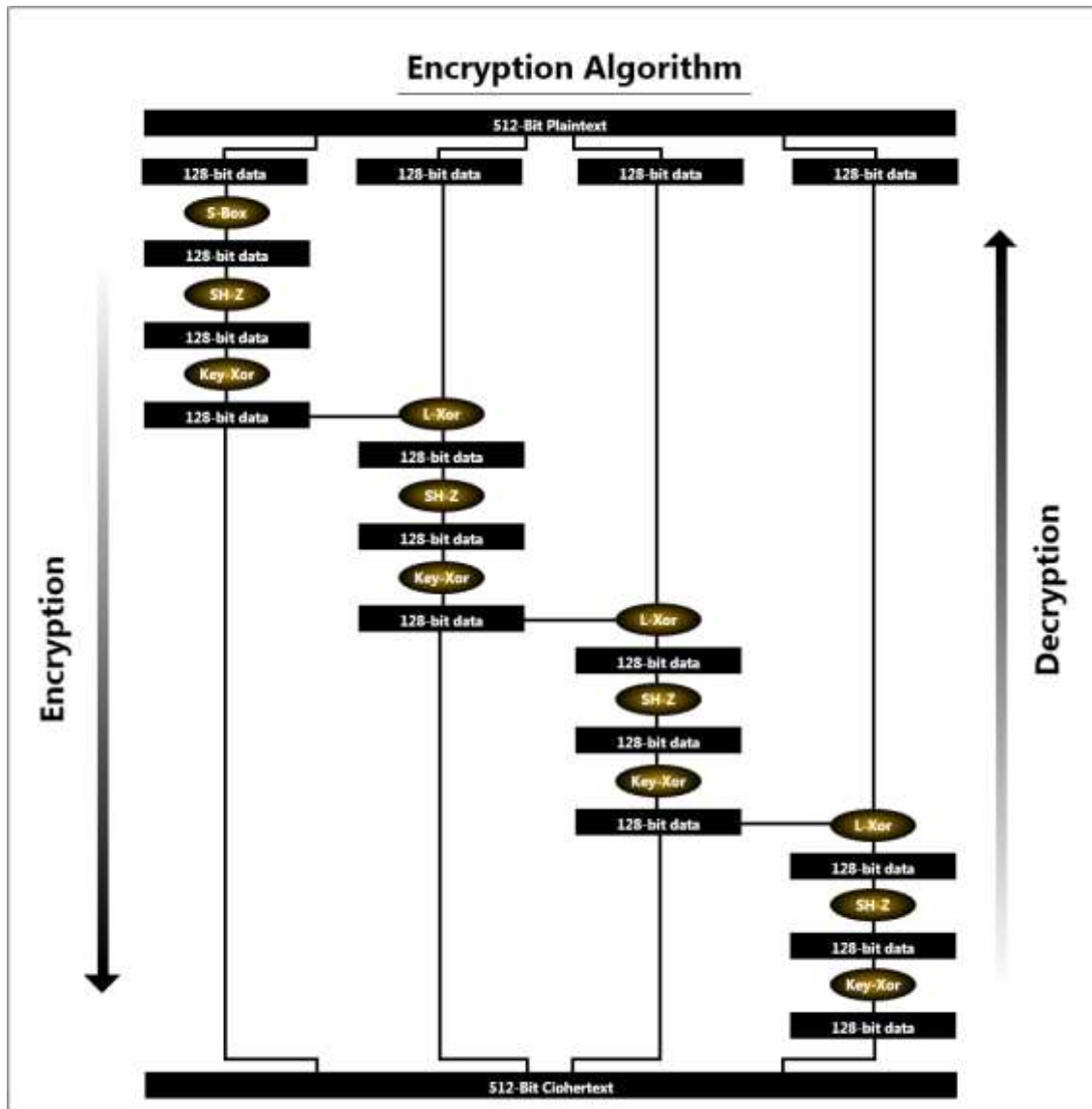


Figure III.1: General architecture of the proposed algorithm.

### 2.1.2. Overview of the operation

The encryption algorithm generates four ciphertext blocks of 128-bits from four plaintext blocks of 128-bits and four keys of 128-bits, the sub-key generator supplies 10 sub-keys of 512-bits for each main key (each one is divided to four 128-bit sub-keys in each round).

The steps of the encryption algorithm are the following:

- 1)     Divide the plaintext for (4 \* 128-bit) data blocks.
- 2)     A loop of 10 iterations that contains three modules:
  - Apply the S-Box on the first block and the L-Xor on the other three blocks.
  - Apply the SH-Z on all four blocks.
  - Apply the Key-Xor on all the four blocks.
- 3)     Collect the four 128-bit blocks into one block of 512-bit (the Ciphertext).

here is the code:

```
Encrypt (blocks A, B, C, and D: Key K1, K2, K3, and K4) // A B C D: 128 bit data blocks. K1 K2 K3 K4: 128 bit key blocks.
begin
  for i = 1 to Nr // Nr: number of turns
    apply the S-Box on the first block and the L-Xor on the other three blocks.
    apply the SH-Z on all the four blocks.
    apply the key-Xor on all the four blocks.
  end for
end
```

**Figure III.2:** General code of the encryption algorithm.

### 2.1.3. The encryption algorithm transformations

#### 2.1.3.1. Bytes substitution

The 512 bits input plaintext is divided into four 128-bit parts. Each part is organized in array of 16 bytes that are substituted by values obtained from substitution boxes. The S-Box used in the proposed algorithm is the same one used in the AES algorithms in case there is no need to generate a new one because of its proven effectiveness. This is done to raise the security level according to diffusion-confusion Shannon's principles for cryptographic algorithm design [20]. The S-Box is shown in Fig III.3.

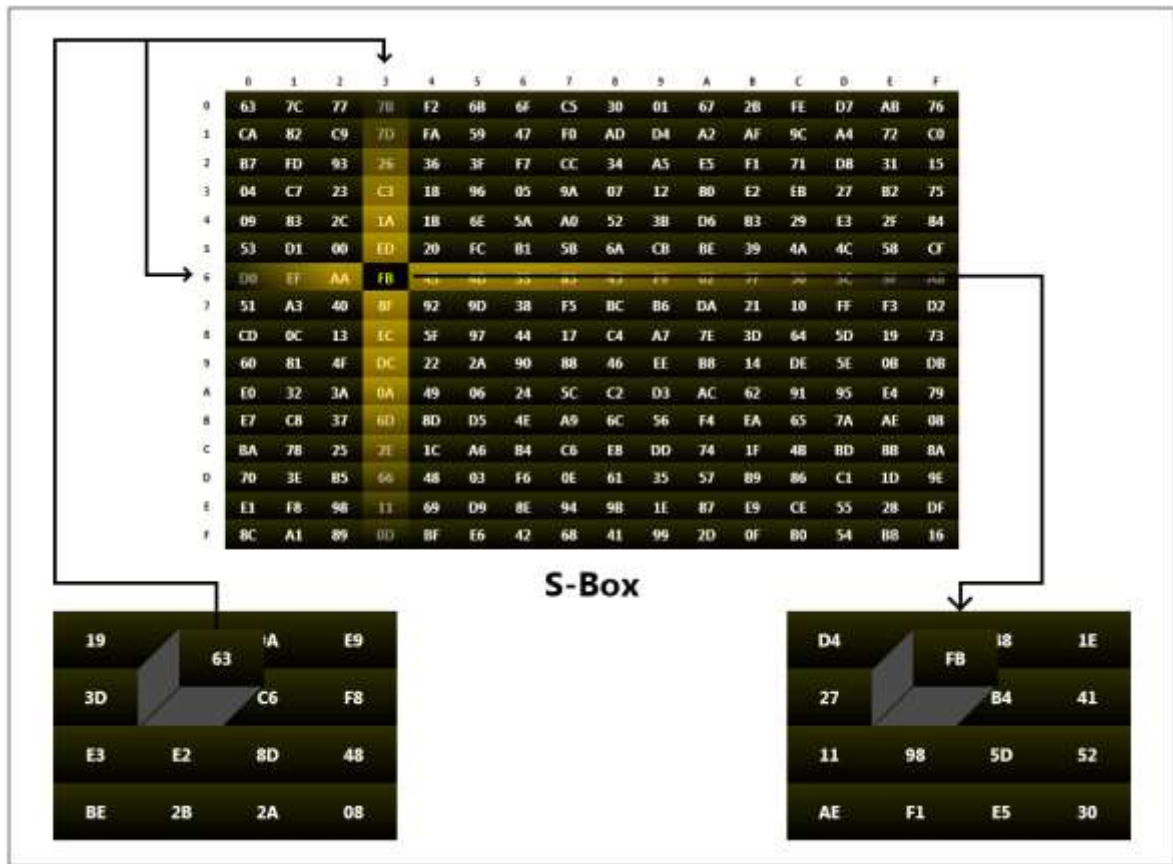
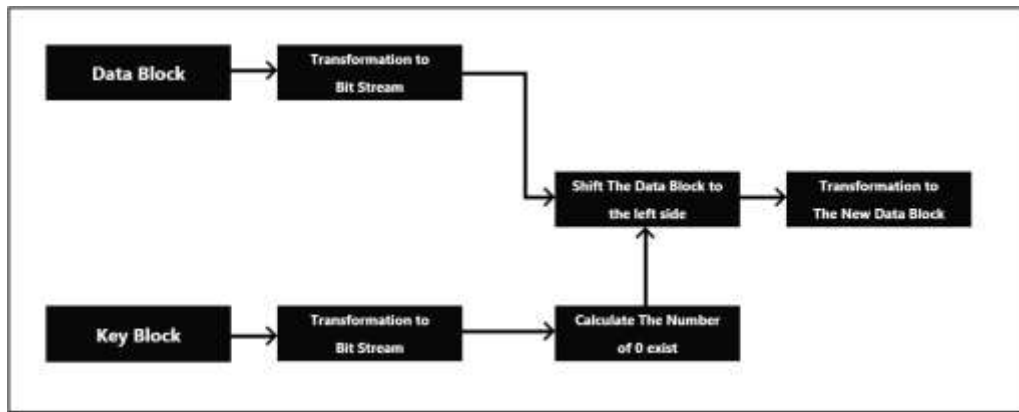


Figure III.3: Byte substitution.

### 2.1.3.2. SH-Z transformation

This transformation depends on the number of zeros exist in each 128-bit part of key applied in the block. First, two matrix of bytes (data block part and key block part) are translated to bit-stream blocks. Second, number of zeros exist in the bit-stream block of key part is calculated. Then, bit-stream block of data is shifted to the left side as the calculated number of zeros. Finally, that bit-stream block of data is retranslated to a result matrix of bytes. Fig III.4 explain the procedure:



**Figure III.4:** SH-Z transformation.

Let us take an example to understand better how this transformation works:

We have two matrix (data block matrix and sub-key matrix):

Data block matrix:

D4	E0	B8	1E
BF	B4	41	27
5D	52	11	98
30	AE	F1	E5

**Figure III.5:** Data block.

Sub-key matrix:

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

**Figure III.6:** Sub-key block.

First, we translate the two matrix of bytes (data block and sub-key block) to two bit-stream blocks. In this step, we collect all the bit-stream block that were bytes in the two matrix in two main bit-stream blocks. We have as result:

The bit-stream block of data: « 11010100 10111111 01011101 00110000 11100000 10110100 01010010 10101110 10111000 01000001 00010001 11110001 00011110 00100111 10011000 11100101 ».

And the bit-stream block of sub-key: « 00101011 01111110 00010101 00010110 00101000 10101110 11010010 10100110 10101011 11110111 00010101 10001000 00001001 11001111 01001111 00111100 ».

Second, we calculate the number of zero exist on the bit-stream block of sub-key. In this example, number of zeros in this sub-key block is 38.

Third, shift the bit-stream block of data to the left side as the calculated number of zeros. After shifting, we have this result: « 01101010 10111111 01110001 01011000 10000000 10011100 11110100 11110011 11000010 10110111 11100001 01010001 01100010 10001010 11101101 00101010 ».

The order of bits is completely different now.

Fourth, we retranslate the bit-stream block of data to a matrix of bytes.

6A	80	C2	62
BF	9C	B7	8A
71	F4	E1	ED
58	F3	51	2A

**Figure III.7:** The result block.

Obviously, we see the data block has been changed to become a new one and no thing is like first time.

### 2.1.3.3. Key-Xor and L-Xor transformations

Both functions are EXCLUSIVE-OR functions (called EXCLUSIVE DISJUNCTION functions in some references) the difference between them is the inputs that they use. Key-Xor function depends on the outcome of each process of HS-Z function and each 128-bit key part block to produce a part of cipher text, this function is applied on all the four block parts as shown in the flowchart of the HLES general architecture. L-Xor function depends on the previous outcome of the Key-Xor function and the current block part; it is applied on the last three block parts, because the first one has no previous outcome of HS-Z function.

## 2.2. Decryption algorithm (D)

The decryption process (D) has four main different transformations that do the inverse of work that the encryption transformations do: Inverse of S-Box transformation, Inverse of SH-Z transformation, Inverse of Key-Xor transformation, and The L-Xor transformation. It means that for any plaintext  $M$ , we have  $D(E(M)) = M$ . The keys of encryption and decryption are exactly the same because the proposed algorithm has a symmetrical design.

The encryption / decryption algorithm is presented in the following general manner:

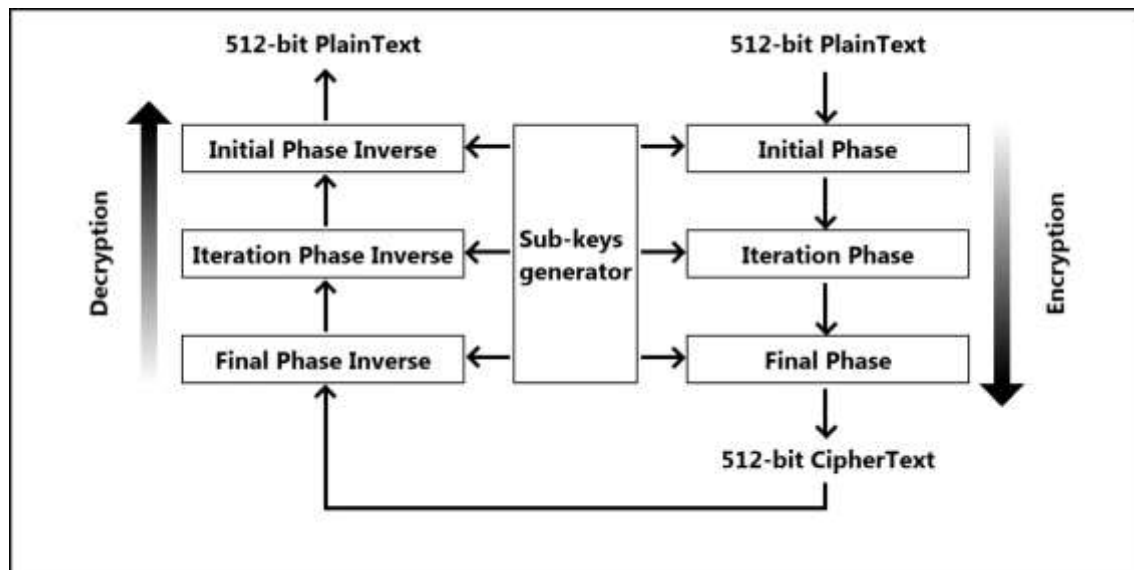


Figure III.8: The encryption / decryption algorithm.

### 2.2.1. Overview of the operation

The decryption algorithm starts with four ciphertext blocks of 128-bits and four keys of 128-bits to reach the four plaintext blocks of 128-bits. The 10 sub-keys of 512-bits supplied by

the sub-key generator in the encryption process are the same here; the only difference is that they start being used from the last one to the first one now. It means that the first round in the decryption process use the last sub-key, and the last round use the first sub-key. The steps of the decryption algorithm are as the following:

- 1)     Divide the Ciphertext for (4 \* 128-bit) data blocks.
- 2)     A loop of 10 iterations that contains three modules:
  - Apply the inverse of Key-Xor on all the four blocks.
  - Apply the inverse of SH-Z on all the four blocks.
  - Apply the inverse of S-Box on the first block and the inverse of L-Xor on the other three blocks.
- 3)     Collect the four 128-bit blocks into one block of 512-bit (the Plaintext).

Here is the code:

```
Decrypt (blocks A, B, C, and D; Key K1, K2, K3, and K4) // A B C D: 128 bit data blocks, K1 K2 K3 K4: 128 bit key blocks.
begin
  for i = 1 to Nr // Nr: number of turns
    apply the inverse of key-Xor on all the four blocks;
    apply the inverse of SH-Z on all the four blocks;
    apply the inverse of S-Box on the first block and the inverse of L-Xor on the other three blocks.
  end for
end
```

**Figure III.9:** General code of the decryption algorithm.

## **2.2.2. Details of the decryption algorithm**

### **2.2.2.1. Inverse of S-Box**

A different S-Box is used here, it is the inverse of the bytes substitution used in the encryption process, in which the inverse S-Box is applied to each byte of the state. Like the first one, it takes an input of byte and transforms it to an output byte in order to get the original data from an existing one. The inverse S-Box used in the decryption process is presented in the following figure:



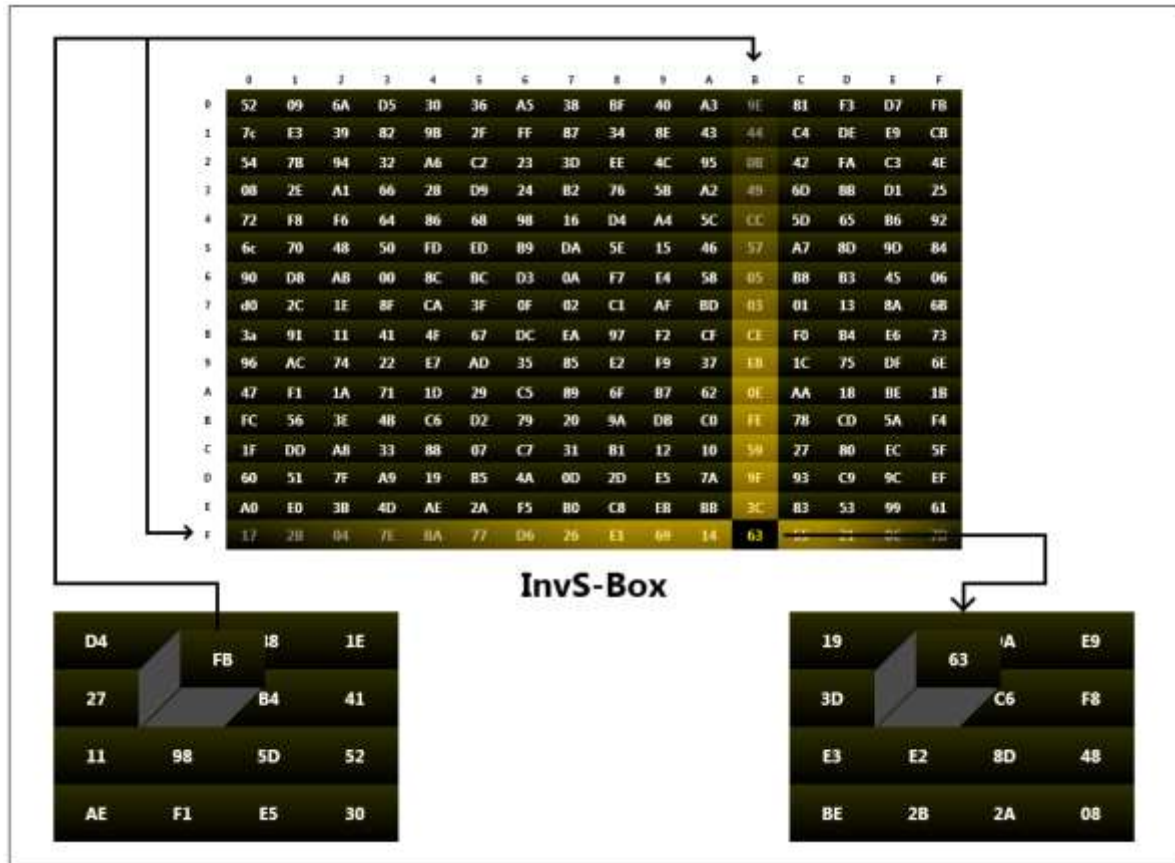


Figure III.10: Inverse of Byte substitution.

#### 2.2.2.2. Inverse of SH-Z

This transformation does the same thing just like the one used in the encryption process, it depends on the number of zeros exist in each 128-bit part of key applied in the block. The only difference is that after converting data block to bit-streams, it shifts those bits to the right side (instead of the left side) as the calculated number of zeros, and then it reconverts them to a data block to retrieve the original block data.

#### 2.2.2.3. Inverse of Key-Xor and L-Xor transformations

Those two transformations do the same work that the others do in the encryption process, they just follow the law of the XOR function. Which means that there no big deal between the XOR and the inverse of XOR.

### 2.3. Sub key generator

The original AES uses a key whose length is 128, 192 or 256 bits. The cipher key is expanded to into 10, 12, or 14 round keys, respectively, using the Key Expansion Algorithm,

where each round key is of 128 bits. This Key Expansion algorithm depends only on the cipher key, and is independent of the processed data. It can therefore be executed independently of the encryption / decryption phase. The round keys are the combination of transformations SubWord (RotWord(tmp)) and SubWord (tmp) and the use of the RCON value. The AES Key Expansion algorithm is described by the pseudo code in Fig III.11 (the pseudo code is written in terms of double words). [21].

```

parametres
Nb = 4
Nk = number of doublewords in the cipher key (4, 6, 8 for AES-128, AES-192, AES-256, resp.)
Nr = number of rounds in the cipher (Nr = 10, 12, 14 for AES-128, AES-192, AES-256, resp.)

The KeyExpansion routine

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word tmp
i = 0
while (i < Nk)
    w[i] = word(key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3])
    i = i + 1
end while
i = Nk
while (i < Nb * (Nr + 1))
    tmp = w[i - 1]
    if (i mod Nk = 0)
        tmp = SubWord(RotWord(tmp)) xor RCON[i / Nk]
    else
        if (Nk > 6 and i mod Nk = 4)
            tmp = SubWord(tmp)
        end if
    end if
    w[i] = w[i - Nk] xor tmp
    i = i + 1
end while

```

**Figure III.11:** The pseudo code of sub-key generation.

Through research and analysis of 128-bit AES key generation and extension mechanism, only if the attacker gets the wheel of the sub-keys, he can deduce all the sub keys by the AES key generation expansion mechanism. As we can find a new way that can generate sub keys from front to back quickly, but the reverse derivation of the keys causes difficulties. It can both increase the difficulty of brute force AES and can effectively prevent the weaknesses of a variety of AES key expansion attack, and will not affect the speed of its current run. [22]. In HLES algorithm, the key is divided to four parts of 128-bit that are expanded to occupy 10 rounds. Therefore, the key expansion algorithm that the original 128-bit AES use is enough since it was and still until now one of the most strong key expansion algorithms against the related key attacks. [23].

## **2.4. Complexity of the algorithm**

To design a symmetric cipher algorithm resistant to block all the possible attacks, it is highly advisable to strengthen the algorithm by permutations and substitutions. The statistical attacks follow the encryption and decryption process to find a statistical relationship between plaintext and ciphertext and get the information, and there success depends on the existence and the number of those permutations and substitutions in the design of the algorithm. However, in the case of multimedia, resource limited systems and real time applications, the cost will be huge when it comes to the memory consummation and the time required for the encryption and decryption process. That is why a light design that does not have a lot of permutations and substitutions and in same time provides the level of security required is needed. Means that more the design have less permutations and substitutions, more the algorithm is light and good. The proposed algorithm is better than the 512-bit AES because it is not heavy and it works in several conditional places like multimedia, resource-limited systems and real time applications.

## **2.5. Analysis of Security of the method**

It has been always believed that the security of a block cipher relies on the secrecy of the key. If an adversary breaks a block cipher and recovers the secret key, he is able to obtain the plaintext from a corresponding ciphertext, and vise versa. A block cipher is considered unbroken if all known attacks have time/data complexity larger than the exhaustive-key search, but that doesn't mean that discovering weaknesses in a cipher design might not end up obtaining the whole or part of the plaintext from a ciphertext. an exhaustive key search (or a brute force attack), is a cryptanalytic attack that can, in theory, be used against any encrypted data. Such an attack might be used when it is not possible to take advantage of other weaknesses in an encryption system (if any exist) that would make the task easier. It consists of systematically checking all possible keys or passwords until the correct one is found. In the worst case, this would involve traversing the entire search space. When password guessing, this method is very fast when used to check all short passwords, but for longer passwords it will be so difficult or may be impossible to be used because of the time that it takes. The key length used in the cipher determines the practical feasibility of performing a brute-force attack, with longer keys exponentially more difficult to crack than shorter ones. A cipher with a key length of  $N$  bits can be broken in a worst-case time proportional to  $2^N$  and an average time of half that. NIST and NSA consider that the AES algorithm is strong against this type of attacks with a key of 128,

192 or 256 bits. However, in fact it will be easy to brake it after a few years of hardware development.

The proposed algorithm has four 128-bits keys (4 X 128-bit). Therefore, the complexity will be:

$O((2^{128})^4)$ . And we know that the number of possible keys for 128 bit is:  
340282366920938463463374607431768211456).

The number of possible keys that we will have with 512-bit key size is very huge comparing to what the original AES key size is.

### **2.6. Analysis of performance of the algorithm**

Many powerful encryption algorithms exist with different key sizes. Those who use a big size of keys provides a high security level, but they are not good in performance because they require too much of time and memory space in the encryption and decryption process.

When it comes to the criteria of the memory space, there are two approaches to effect an ideal comparison. The first one is measuring the memory space spent by all the functions of the encryption process; in this case, to encrypt one data block of 512-bit size, the 512-bit AES uses four functions that occupy together 2048 bits (512-bit x 4). While the area design of the proposed algorithm (HLES) allows to occupy only 1536-bits (128-bit x 3 x 4), which is less than the other one by 1/4. The second one is measuring the memory space spent by each function of the encryption process. In this case, we have to take in consideration the size of memory space that each function consume to process one data block at one point of time for each algorithm. In 512-bit AES, all the functions work with 512-bit data block size. While in HLES algorithm, there are functions that work with 128 bit block size and functions that work with 256 bit block size (functions that need two 128 bit blocks to produce one 128 bit block), but generally HLES algorithm has no function that consume more than 256 bit.

When it comes to the encryption time, the proposed algorithm applies the principle of "more diffusion and confusion functions in design means more security ". It uses a mathematical functions like shifting rows and the XOR function that can ensure a high level of security without spending too much of time. Note that the mixcolumn function is doing well in the 128-bits AES algorithm, but it consumes a lot of time in the 512-bit AES algorithm. That is why it effects on its performance.

### **3. Conclusion**

In this chapter, we presented a new design of a cryptographic algorithm as an improvement for 512-bit AES algorithm with efficient utilization of resources such as encryption time and memory space, and the details of each step that constitutes it. We explained all the criteria assumed in the design, and the recommendations set out by world experts in cryptography. The proposed design is based on recommendations and standard techniques and contains a module consisting of several phases inspired from cryptographic methods and mathematical mechanisms.

In the next chapter, we will present the implementation of this algorithm, and the results of experimental tests.