

1. Introduction

In this chapter, we will present the implementation of the proposed algorithm and the results of various tests in order to evaluate its performance. The tests of cryptographic methods are usually performed on two major axes: Execution time axis: These tests are aimed to assess the performance of the encryption / decryption time of the algorithm, and by varying some of its parameters (number of revolutions, encryption mode... etc.). Security axis: These tests are aimed to assess the security of the algorithm, in other words, evaluating the strength of the algorithm against the most powerful attacks. However, in the modern communicating world of resource-limited systems, another axis appeared to be performed with the same importance of the other two written, which is the memory space axis.

2. Environment of development

For the realization of good test and having reliable results we used different tools and technics to guarantee that, they are explained as follow:

2.1. Operating system:

The operating system chosen for the realization of our testing application is Windows 7 Professional it is a system known for its efficiency, reliability, robustness, and security, as well as its wealth of programming tools.

2.2. Programming language:

We have chosen the C # programing language to write our program, this choice is motivated by the following reasons:

- C# allows the use of object classes and apply consequently the advanced object-oriented programming techniques.
- The C# compilers are currently implemented on all Windows platforms, which makes it very common programming tool.
- The code generated by the C# compiler is highly optimized, making it executable files more compact and faster.
- Most implementations of standard algorithms are implemented on C # language base, which make the comparing tests easy to perform.

2.3. Programming tool, platform, and environment:

We have exploited Visual Studio 2013 programming tool, and the WPF environment for the realization of the GUI (Graphical User Interface). Since the WPF is the new platform for Microsoft Windows application development, based on the .NET Framework that provides a clear, object-oriented, extensible set of classes that enable to develop rich Windows applications. In addition, the XAML code makes it easy to create and edit a GUI, and allows the work to be split between a designer (XAML) and a programmer (C#, VB.NET ... etc.), It's newer and thereby more in tune with current standards And Microsoft is using it for a lot of new applications.

3. Architecture of the application

We opted for object-oriented programming (OOP) because it allows a better readability of the source code and a considerable simplicity of debugging and error detection, reuse, modularity and ease of maintenance. In addition, the WPF environment allows splitting the GUI design code from the programming source code so the work on the implementation of the algorithms can be done easily.

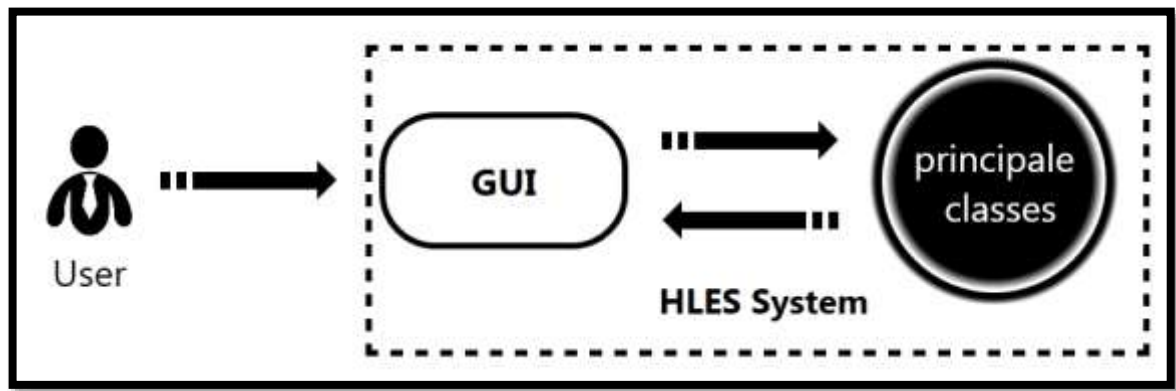


Figure IV.1: architecture of the application.

3.1. GUI (Graphical User Interface)

Cryptographic algorithms are not intended for the general public. However, we have tried to give a visual, attractive and friendly aspect to our application through an ergonomic graphical user interface that allows users to interact with the application easily. The GUI that we have done is explained latter.

3.2. The principal classes

In this section, we present the classes used throughout the implementation. All the principal classes and their methods are presented on tables in what follow:

- **FirstAESEncryption**

This class contains the necessaire methods and fields for the Encryption process of the 128-bit AES algorithm.

Methods & fields	Description
SBox()	Byte substitution
ShiftRow()	Shifting rows
Xor()	XOR function for a single bit
MisColumns()	GF Multiplication
KeyXor()	XOR function for a hole block

Table IV.1: FirstAESEncryption class.

- **FirstAESDecryption**

This class contains the necessaire methods and fields for the Decryption process of the 128-bit AES algorithm.

Methods & fields	Description
Inv_SBox()	Inverse of Byte substitution function
Inv_ShiftRow()	Shifting rows to the opposite side
Xor()	XOR function for a single bit
Inv_MisColumns()	Inverse of GF Multiplication
KeyXor()	XOR function for a hole block

Table IV.2: FirstAESDecryption class.

- **SecondAESEncryption**

This class contains the necessaire methods and fields for the Encryption process of the 512-bit AES algorithm.

Methods & fields	Description
SBox()	Byte substitution function
ShiftRow()	Shifting rows
Xor()	XOR function for a single bit
MisColumns()	GF Multiplication
KeyXor()	XOR function for a hole block

Table IV.3: SecondAESEncryption class.

- **SecondAESDecryption**

This class contains the necessaire methods and fields for the Decryption process of the 512-bit AES algorithm.

Methods & fields	Description
Inv_SBox()	Inverse of Byte substitution function
Inv_ShiftRow()	Shifting rows to the opposite side
Xor()	XOR function for a single bit
Inv_MisColumns()	Inverse of GF Multiplication
KeyXor()	XOR function for a hole block

Table IV.4: SecondAESDecryption class.

- **HLESEncryption**

This class contains the necessaire methods and fields for the Encryption process of the proposed algorithm (512-bit HLES).

Methods & fields	Description
SBox()	Byte substitution function
SHZ()	Shifting bits
Xor()	XOR function for a single bit
KeyXor()	XOR function for a hole block

Table IV.5: HLESEncryption class.

- **HLESDecryption**

This class contains the necessaire methods and fields for the Decryption process of the proposed algorithm (512-bit HLES).

Methods & fields	Description
Inv_SBox()	Byte substitution function
Inv_SHZ()	Shifting bits
Xor()	XOR function for a single bit
KeyXor()	XOR function for a hole block

Table IV.6: HLESEncryption class.

- **BaseTransformation**

This class contains the necessaire methods and fields of all the kinds of the transformations needed in the application.

Methods & fields	Description
FromStringToHex()	Transforming String to Hexadecimal
FromHexToString()	Transforming Hexadecimal to String
ByteArrayToStringHex()	Transforming Byte array to Hexadecimal
StringHexToByteArray()	Transforming Hexadecimal to Byte array
BlockSizing()	Resize the block stream to be multiplied of a specific unit of size

Table IV.7: BaseTransformation class.

- **Settings**

This class contains the necessaire methods and fields needed in uploading and downloading files.

Methods & fields	Description
OFDFilter	String containing possible File Extensions
CncStr	Connection string of database
Plc	Registered File path
Ext	String to save the File extension
FileIO()	Open File Dialog to upload the file

Table IV.8: Settings class.

- **Main Class**

We have created five user controls to operate the three algorithms separately, the main class has nothing to do except loading theme. Each one play the same role that a programming window does. Those user controls are explained in what follow:

- **User control 1:** used for uploading statistical results of old encryption process.



Figure IV.2: user Control 1.

- **User control 2:** used for the encryption / decryption process of the 128-bit AES algorithm.



Figure IV.3: user Control 2.

- **User control 3:** used for the encryption / decryption process of the 512-bit AES algorithm.



Figure IV.4: user Control 3.

- **User control 4:** used for the encryption / decryption process of the HLES algorithm.

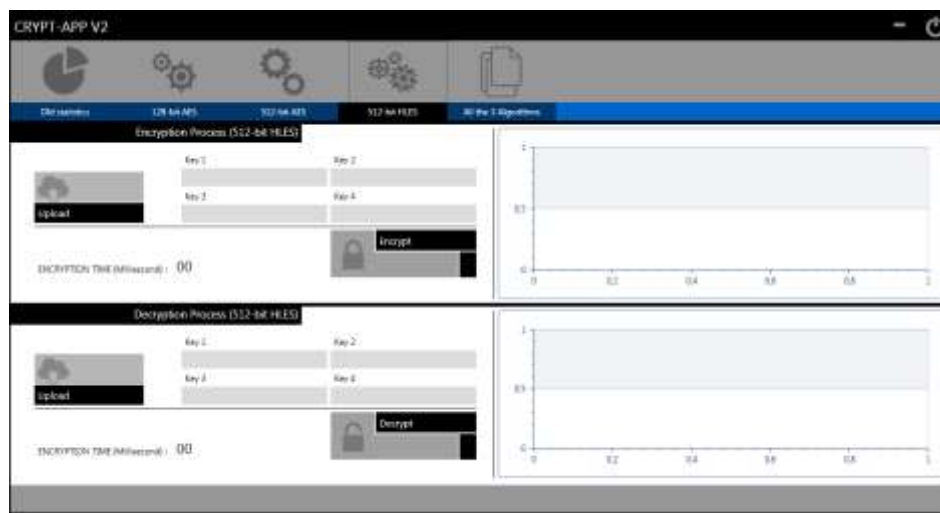


Figure IV.5: user Control 4.

- **User control 5:** used for the encryption / decryption process of all the three algorithms together.



Figure IV.6: user Control 5.

The following diagram presents the classes mentioned before:

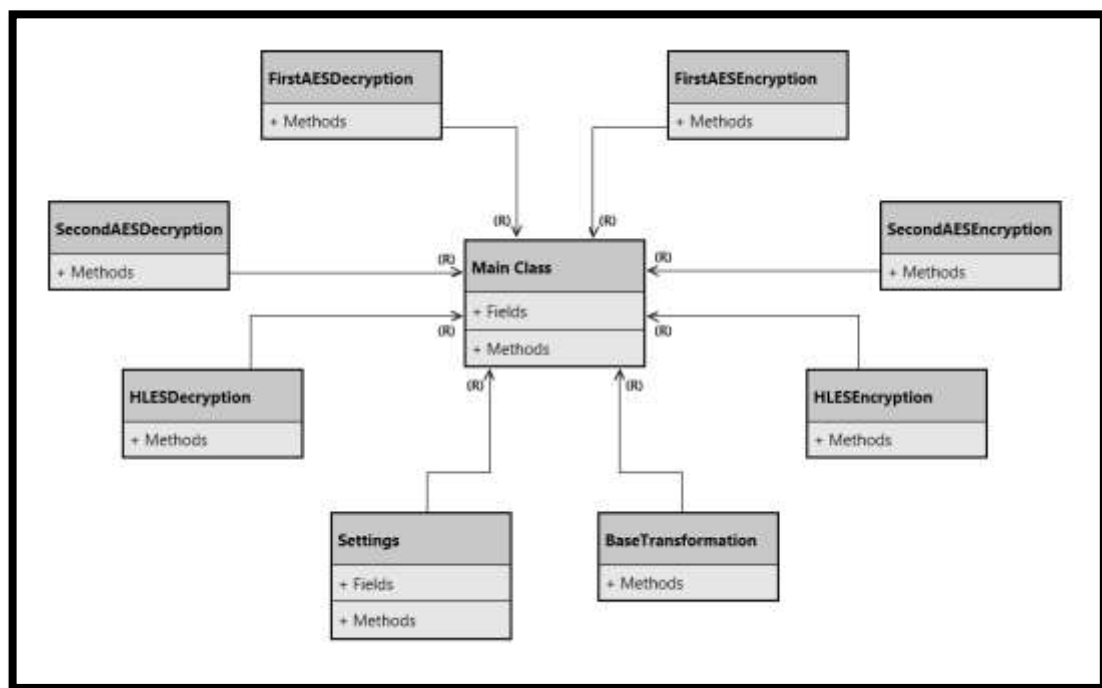


Figure IV.7: principal classes.

4. Tests and experimental results

In what follows, we will present the various tests and evaluations carried out and the results obtained by the proposed algorithm. To apply those tests, we used a Windows 7 – 64 bit operating system, and a machine with a microprocessor INTEL i3 2.53GHz and 4GB of RAM.

To evaluate the improvement that the proposed algorithm gives, we made a comparison between the new design and the two current standards AES (218-bit AES and 512-bit AES). This comparison is based on three criteria: execution time, memory space, and security.

The tests carried out are pointed on the criteria of Encryption / decryption time, they consist of calculating the time that the algorithm takes to Encrypt / Decrypt a data file. For this goal, three executions are made for each type of file, than the mean of execution time is calculated and given as a final result for each file. We will explain this part in details in what follows.

4.1. Criteria of the encryption time

In order to compare between the three encryption algorithms (128-bit AES, 512-bit AES and HLES), there designs were coded in same operating system (Microsoft Windows 7), same programming platform (Visual Studio Platform), same environment (WPF environment) using same programming language, which is C# language. The used operation mode is the ECB mode (Electronic Code Book), which is the simplest one.

4.1.1. Tests and results

We used five different data block files of different memory capacities to be encrypted (text file, JPG image, PNG image, MP3 sound file, and MP4 video file), we conducted three tests for each file and we calculated there Moyne. The test results are shown in tables below of each file, and presented in a succinct and clear in graphs. Each table contains the Encryption / Decryption time of each algorithm, and followed by a graph explains the final results of each algorithm (Moyne).

Text file:

The used text file has the following characteristics:

- Size: 1.41 Kbytes.

- Format: txt.

	128-bit AES		512-bit AES		512-bit HLES	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
Execution 1	85	56	99	99	61	53
Execution 2	45	49	63	74	54	48
Execution 3	54	43	76	54	51	38
Mean	61	55	79	75	47	39

Table IV.9: Encryption / Decryption time of text file.

To visualize the data better, we present them in the following graph:

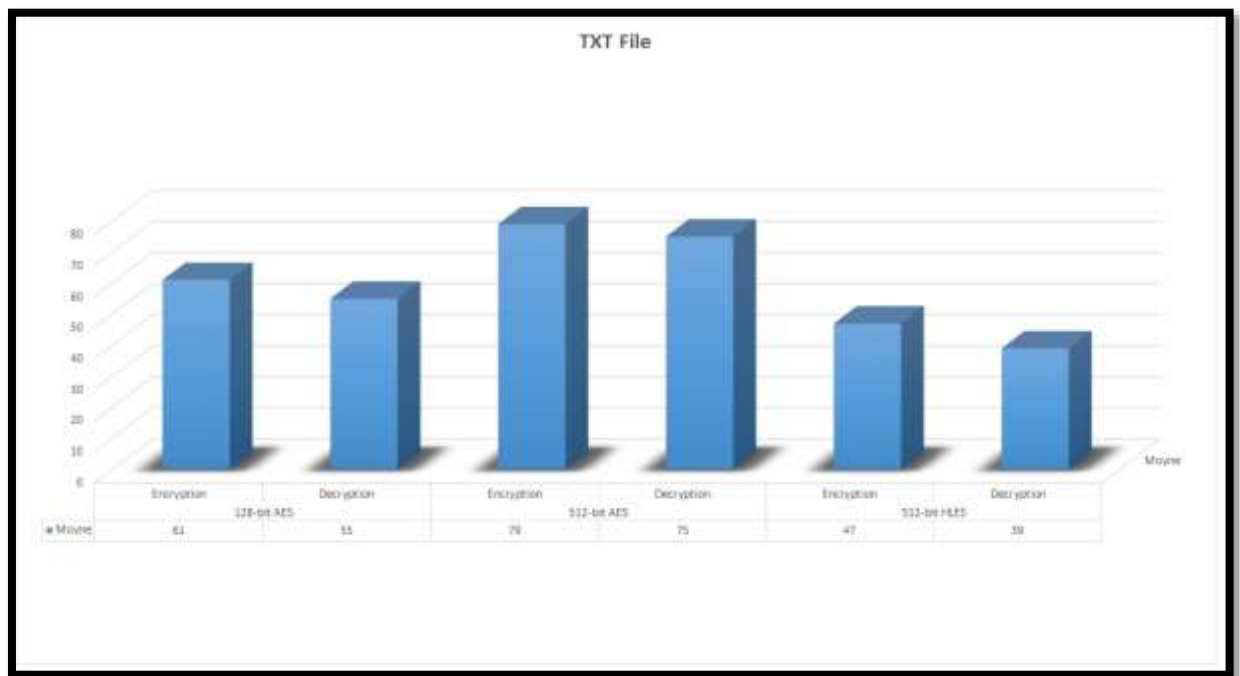


Figure IV.8: Encryption / Decryption time for TXT File.

JPG image:

The used JPG image file has the following characteristics:

- Size: 67.2 Kbytes.
- Format: jpg.

	128-bit AES		512-bit AES		512-bit HLES	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
Execution 1	1617	1538	1818	2031	909	868
Execution 2	1528	1611	1753	1855	866	880
Execution 3	1588	1601	1750	1846	881	868
Mean	1577	1583	1773	1910	885	872

Table IV.10: Encryption / Decryption time of JPG image.

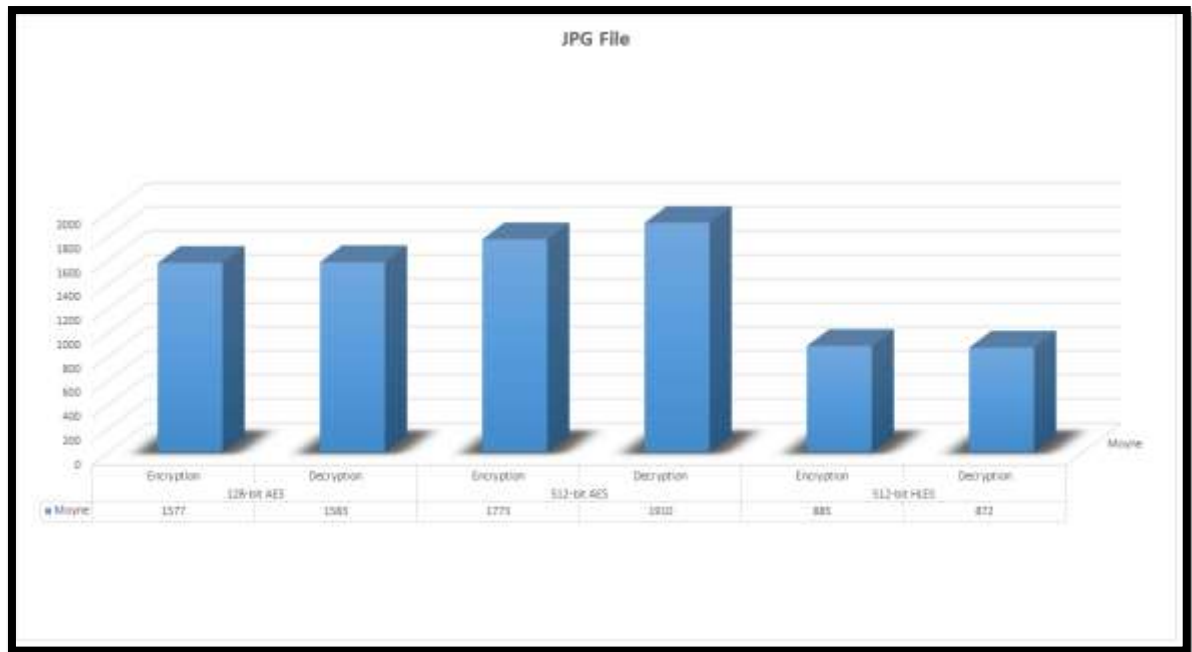


Figure IV.9: Encryption / Decryption time for JPG File.

PNG image:

The used PNG image file has the following characteristics:

- Size: 161 Kbytes.
- Format: png.

	128-bit AES		512-bit AES		512-bit HLES	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
Execution 1	5141	5244	5619	5255	3003	2803
Execution 2	5258	5427	5415	5475	2227	2151
Execution 3	5328	5428	5505	5697	3688	2650
Mean	5242	5366	5513	5475	2972	2534

Table IV.11: Encryption / Decryption time of PNG image.

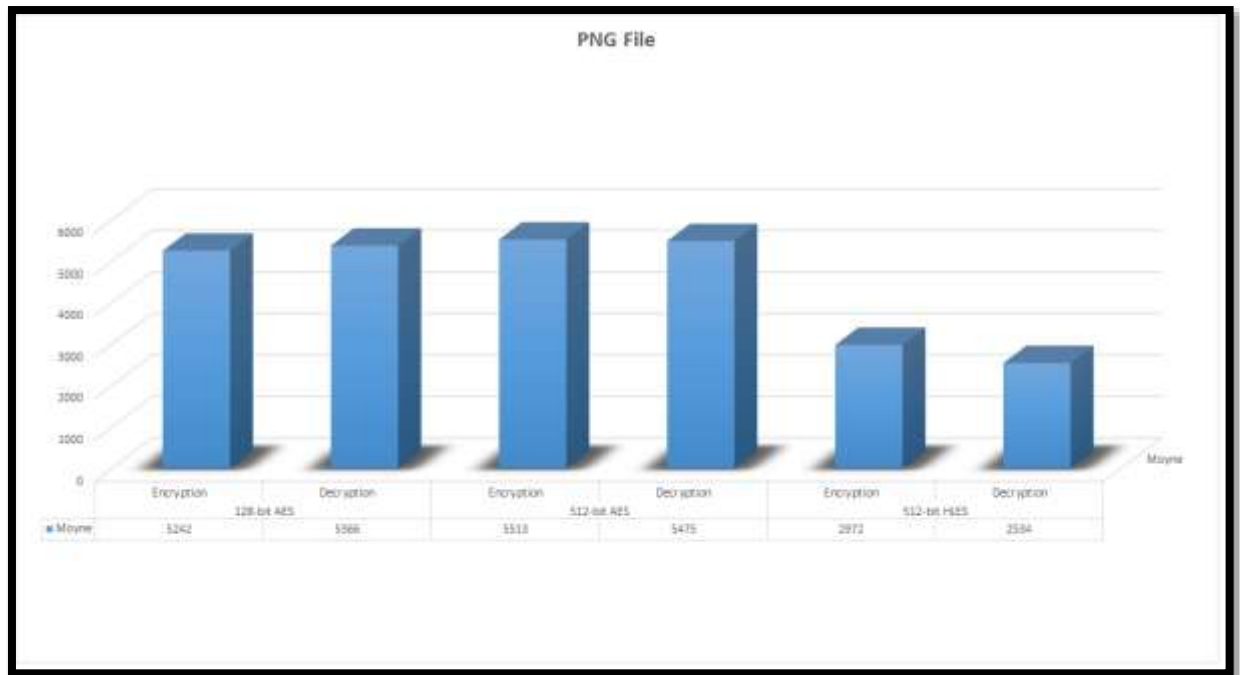


Figure IV.10: Encryption / Decryption time for PNG File.

MP3 sound file:

The used MP3 sound file has the following characteristics:

- Size: 105 Kbytes.
- Format: mp3.

	128-bit AES		512-bit AES		512-bit HLES	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
Execution 1	3975	3631	4704	5910	2520	2233
Execution 2	3670	3685	6652	6793	2321	2348
Execution 3	3694	3710	6772	7292	2387	2404
Mean	3778	3675	6042	6665	2409	2328

Table IV.12: Encryption / Decryption time of MP3 sound.

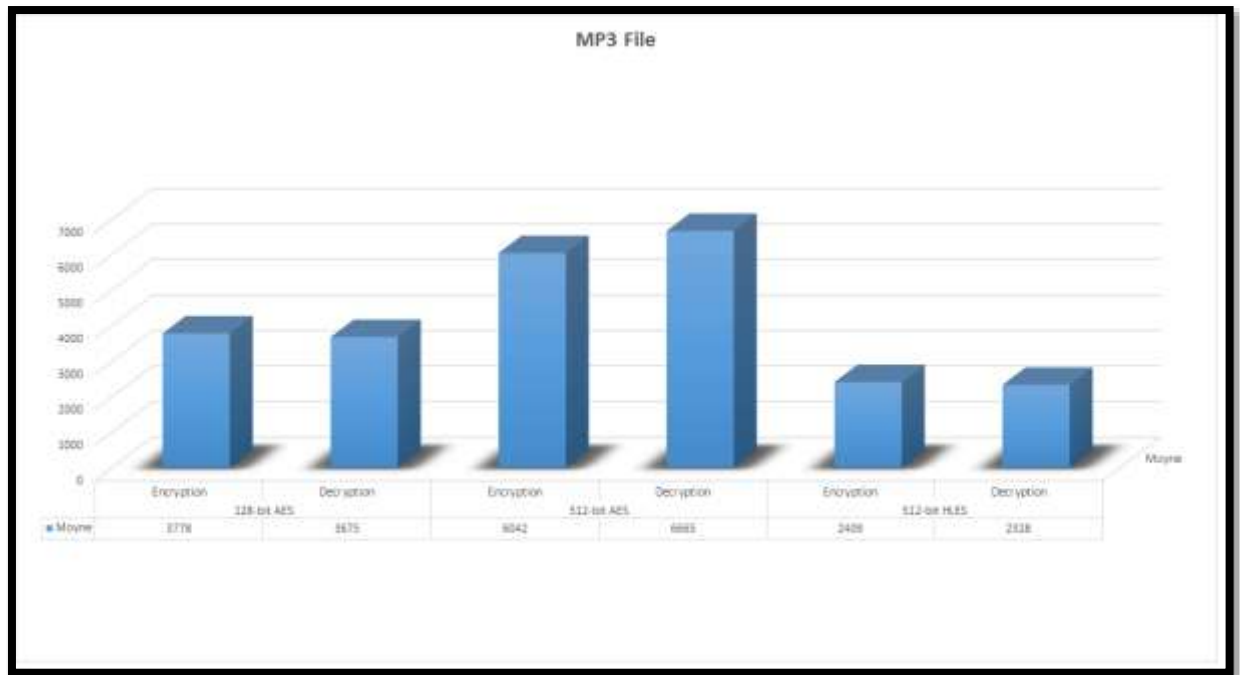


Figure IV.11: Encryption / Decryption time for MP3 File.

MP4 video file:

The used MP4 video file has the following characteristics:

- Size: 132 Kbytes.
- Format: mp4.

	128-bit AES		512-bit AES		512-bit HLES	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
Execution 1	5729	5011	8631	9540	3060	3144
Execution 2	5170	6414	10318	11288	3824	3936
Execution 3	5278	12185	5278	11890	3623	3974
Mean	5392	7870	8075	10906	3502	3684

Table IV.13: Encryption / Decryption time of MP4 video.

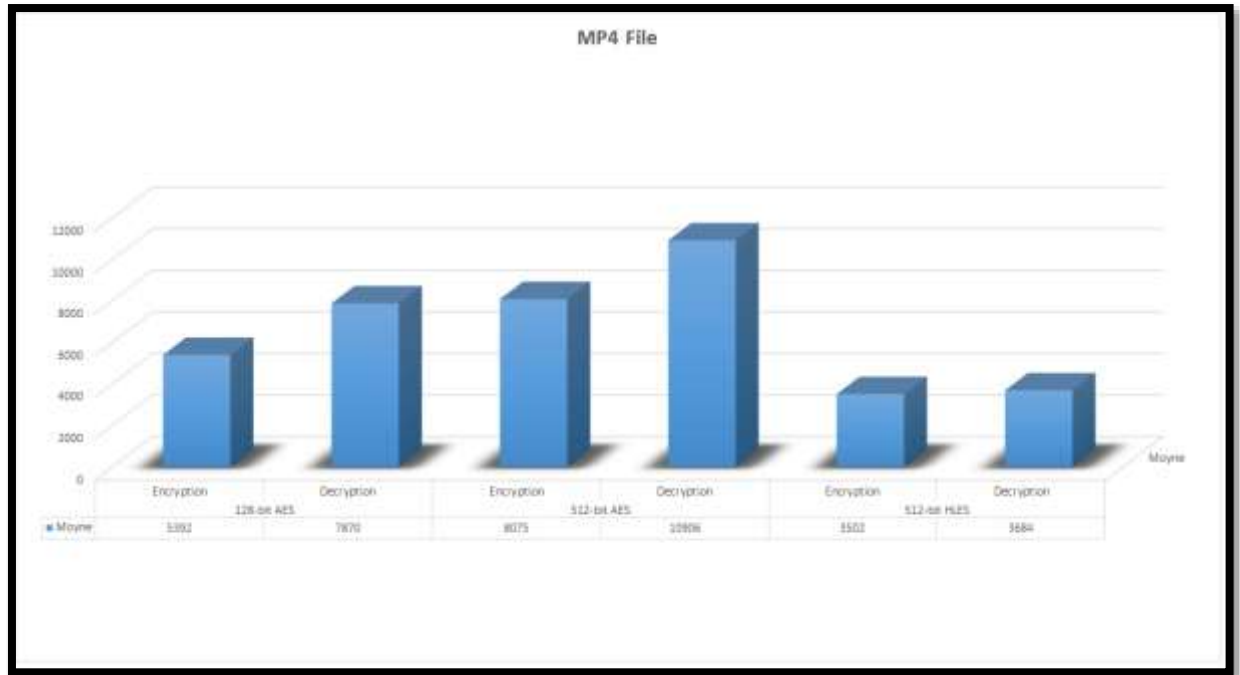


Figure IV.12: Encryption / Decryption time for MP4 File.

From the results obtained of the various tests, we can see that the proposed algorithm is much better than the other two AES versions in both of Encryption / Decryption process, especially when it comes to the big data files.

4.1.2. The encryption evolution vis-a-vis the file size

The encrypted file size is a parameter that has a great influence on the behavior of the encryption / decryption process of the three algorithms (128-bit AES, 512-bit AES and HLES). This parameter brings up a very high difference in the performance. With the linear growth of the file size, the encryption / decryption time of the two AES algorithms explodes exponentially. However, the encryption / decryption time of HLES algorithm is scaled linearly with the growth of the size of the test file with a small change. This observation emphasizes that the HLES

algorithm is much more efficient and effective than AES algorithm when it comes to the big size of data, means that more the data is bigger more the change in time becomes less. The following figure shows the change of time comparing with the size of data for the three algorithms.

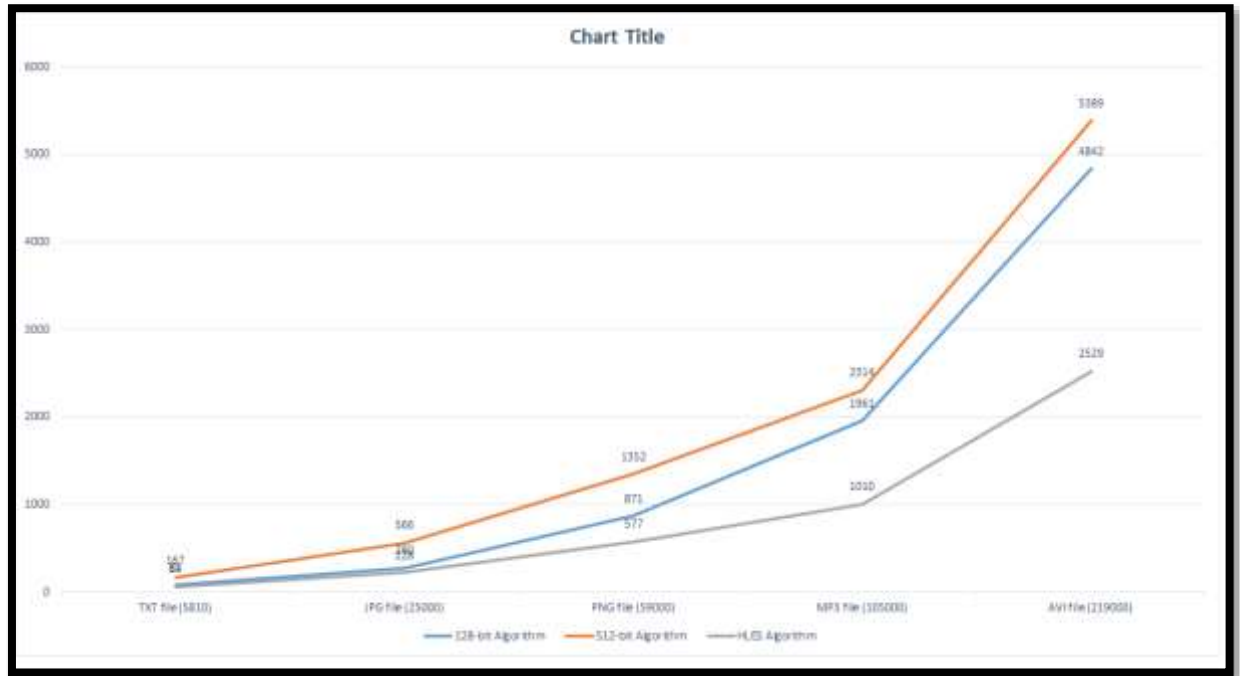


Figure IV.13: change of time comparing with the size of data for the three algorithms.

4.2. Criteria of the memory space

When it comes to the criteria of the memory space, the difference will be explained theoretically. The 128-bit AES has no relationship with this criterion because it is more than good thanks to its small data block size and key size. This criterion is applied on the 512-bit AES and HLES algorithms. There are two approaches to effect an ideal comparison on the memory space factor as explained in what follow:

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 ($512\text{-bit} \times 4$). While the area design of the proposed algorithm (HLES) allows to occupy only 1536-bits ($128\text{-bit} \times 3 \times 4$), which is less than the other one by 1/4.

The second approaches 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 data block that each function consume at the same 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. The following table shows the difference of memory space at one point of time for each algorithm:

	512-bit AES	512-bit HLES
Bit number	512 bit	128 or 256 bit

Table IV.14: difference of memory space at one point of time.

The proposed algorithm (HLES) uses memory space less than the other one that 512-bit AES uses, which makes it acceptable to be used in resource-limited systems.

4.3. Study of the method security

Among the major disadvantages of symmetric cryptography is that security cannot be proved theoretically. The best way to evaluate the security of a symmetric cryptographic algorithm is publishing it, because we cannot imagine (or evaluate) all attacks that exist in the world.

In our work, and for feasibility reasons, we could not perform sufficient safety testing of our methods. Indeed, the majority of modern attacks are statistical attacks, so the implementation of such an attack requires advanced statistical knowledge. In addition, all of these attacks depends on the algorithm, means that their implementations differ from one algorithm to another.

However, the choices made in the design (i.e. a block cipher design) will allow us to have a method with very acceptable safety, since we respected the realization of Shannon principle namely the confusion and diffusion [19]. Here is a practical evaluation of a brute force attack (exhaustive search) applied on keys of 128-bit size and 256-bit size in a machine that can test 1 million (10^6) key per millisecond:

Key size	Possible key number	Encryption time	Decryption time
128	$2^{128} = 3.4 \cdot 10^{38}$	$5.4 \cdot 10^{18}$ years	$5.0 \cdot 10^{18}$ years
256	$2^{256} = 11.56 \cdot 10^{76}$	$18.36 \cdot 10^{56}$ years	$18.36 \cdot 10^{56}$ years

Table IV.15: evolution of brute force attack for 128-bit and 256-bit key size.

The number of possible keys of 512-bit size is so huge and the Encryption / Decryption time to test all of them is almost being impossible to be counted.

5. Conclusion

In this chapter, we have seen the different classes and methods of our algorithm, and we conducted several tests in order to assess its performance, including its behavior when the parameters vary. Moreover, we presented the different results obtained from these tests followed by comments and practical and theoretical justifications. Those tests are performed to estimate the Encryption / Decryption time of our algorithm (with default settings). We also varied the algorithm parameters and followed the behavior of the memory consumption beside the criteria of time by making a comparison between our algorithm and equivalent implementation (under the same conditions) of the two current versions of the standard AES (128-bit AES and 512-bit AES).

The results obtained show that encryption / decryption time is linear with the increase in these parameters, which is acceptable in terms of complexity, and the memory space used in the whole process is much less than the 512-bit AES to be able to work with special characteristics of resource-limited systems.