The Advanced Encryption Standard Block Cipher "Different Modes"

BICS- Sean ACHTATOU

University Of Luxembourg

E-mail : sean.achtatou@hotmail.be

Abstract:
This project will provide a new way to perceive the security of the Advanced Encryption Standard already known. Some modes of the Advanced Encryption Standard will be approached and analyzed all along the project making a new idea about the real efficient of the Advanced Encryption Standard in everyday life. This project will mainly be focused on the Encryption side, beside the Decryption will need to be approached as well for a good reason afterwards.

1.INTRODUCTION

Since always, the mankind has secrets and try to keep it safe and protected. In history a lot of manners has been invented to manage it. But today with the actual technologies the mankind has created the Advanced Encryption Standard.
The Advanced Encryption Standard[AES] first named ''Rijndael Cipher", is one among the Block Cipher the most used and secured today. It has been invented in 1997 by two Belgians Cryptographers, Joan Deamen and Vincent Rijmen, that gave their name to it, explaining "Rijndael". The main task of the Advanced Encryption Standard is to encrypt our information; messages, pictures or more generally your plaintext, which you wish to keep secret or share in secure to someone else. There exist already a lot of ways to encrypt, but the Advanced Encryption Standard still is the best option. In 1999, it contests at the NSIT (National Security of International Technologies) versus others Block Cipher and has been chosen has the most secured one. Since 2001, this Block Cipher is being used by a lot of company, commercial, business, military, and the NSA (National Security Agency) as well.
The Advanced Encryption Standard Block Cipher is based on a powerful combination of different algorithms, each important as each other. It is supported by the use of a Binary Key which can be of different length and will be injected in the Advanced Encryption Standard Algorithm. The Advanced Encryption Standard has the particularity to work as a Symmetric Encryption System, simply means that the same Binary Key that is kept secret is used to Encrypt and Decrypt the data. If it was an Asymmetric Encryption System, a Public Key and a secret Key would be required.
But still the Advanced Encryption Standard must use different mode of operation. It is a little algorithm that helps to increase the level of security, without this step, it would be better to directly give the information in clear. The goal of this project is to fully understand the running of the Advanced Encryption Standard Block Cipher with different mode in its details. The work is focused around the implementation of the Advanced Encryption Standard Block Cipher in a language allowing to test different aspects of the algorithms, deleting, adding some parts and see how it acts at the end. Some results can really turn out surprising about the real security of the Advanced Encryption Standard Block Cipher.
But still none program can be 100% secure.

2.PROJECT DESCRIPTION

2.1. DOMAIN

The Advanced Encryption Standard Block Cipher concerns one of the most studied and complex computing domain of the worldwide, the security. That said, it is still a vast domain to works with. There are some sub-domains of the security this project carried; at first the Cryptography; meaning it "hides and writes" it is a discipline of the Cryptology (Cryptography and Cryptanalyze). This sub-domain helps to protect the information and assure the confidentiality (the information will not be recovered by an unauthorized entity), authenticity (the information will be the real one) and integrity (the information will arrive completed without any loss or modification) of it by the usage of a Random Binary Key, symmetric or asymmetric[Figure25].
The second sub-domain is the Block Cipher; it is where we could say the magic happens, where the Plaintext will be Encrypted to result in an Encrypted Plaintext,,

more named Cyphertext. Such as the cryptography, it assures the confidentiality and the recovery of the Plaintext from a Cyphertext. The use of a Block cipher is one way to secure the information, there exist indeed several others cipher, one of the easiest and best known is the Morse Cipher.
One last sub-domain is the computer science, indeed by the use of an implementation of this Block cipher, we are entering the data processing section. Providing a way to analyses the algorithm of the Advanced Encryption Standard Block Cipher and elaborate the research on the subject.

2.2. OBJECTIVES

The main objective is to understand how works each parts of the algorithm of the Advanced Encryption Standard Block Cipher, then put them all together to get a clear overview of how the Advanced Encryption Standard Block Cipher is emulating.
Thereafter, the implementation of the Advanced Encryption Standard algorithm on a computer is needed. The implementation of this algorithm will helps to accomplish the research and objective pursued on the subject. Indeed, to fully know the operation of the Advanced Encryption Standard, the need to manipulate the algorithm to discern each steps is mandatory. So, once the implementation will be done, the objective is to see how the Advanced Encryption Standard Block Cipher would react when its algorithm is being modified. We will work with the result obtained after adding, deleting or changing the chronology of the algorithm what happens to the Cyphertext.
As well, the same will be done for the different modes of operations enumerated earlier that can be used such as ECB(ElectronicCodeBook), CBC(CipherBlockChaining) and CTR(Counter)[Figure1], and explain the difference that can be observed at the end using one or another. To have a clear vision of what is happening, the Plaintext will be tracked all along the program. Each result collected will be reported, and concluded by the fact that the algorithm has been implement this way for a good reason and therefore why we can trust, or not, the Advanced Encryption Standard Block Cipher for this mode.

2.3. CONSTRAINTS

Like for any project some constraints must be considered at some point, especially when this is the first constraint that could be encountered is the comprehension of the different domains mentioned upper above. For the cryptography, is quite easy to figure out how it works because of its simplicity, it just explains the need of using a Binary Key in the security. Still, the Block Cipher of the Advanced Encryption Standard was a little bit harder yet.
That brought us to the second constraint. The consideration of the mathematical formulas contained in each parts of the algorithms of the Advanced Encryption Standard Block Cipher. Nonetheless some parts were actually rudimentary, several were really hard to discern and understand at first sight because of the mathematical formulas such as XOR or Mix Columns. It can take some time but it is possible to quickly override this difficulty.
Finally, the last constraint and complement of the last one, is the implementation of the algorithm of the Advanced Encryption Standard Block Cipher. Indeed, specially when a beginner in programming, it was quite difficult to choose how to do it. We first have to choose between several programming languages such as C or Python, but the implementation wasn’t a cake walk. It must be the part that took the most of the time in this project, and since it is what we were working with we didn’t have the possibility to fail to this part.

3.BACKGROUND

3.1. SCIENTIFIC

For the scientific part, there will be a justification about the work with the implementation of the algorithm of the Advanced Encryption Standard Block Cipher, and further an explanation as well on how each operation works together to form a good security system. Then, we will introduce the reason about the purpose of each modification done in the implementation of the Advanced Encryption Standard and the outcome to the collection of our results.
First part, as said below, it was the first time we were working in the domain of the security. The Advanced Encryption Standard Block Cipher belongs to a part of the security that is hard to get because it uses Binary Encryption and matrices. Not being used to manipulate binary element in matrices, it was difficult to work with it. Therefore, the need to do an implementation of the Advanced Encryption Standard Block Cipher Algorithm was recommended. Moreover, being a beginner in programming didn’t help as well. It was not an easy

task to implement it in a language.
The Advanced Encryption Standard algorithm is based on a chain of mathematical formulas for each function that results in the security it has. The plaintext is in the first place introduced in a 4x4 Matrix, each case corresponding to 8 Bits, each matrix can provide 16 Bytes of information[Figure18]. First to first, the Key expansion step[Figure2-19]. It is a simple step which allow us to get more rounds of our 128 Bits Main Random key put in a 4x4 matrix, it will extend the Main key depending on its length(128Bits-10Rounds/192-12Rounds/256-15Rounds), into another 128 Bits Random key, which will be needed to increase the security all through the algorithm. Then, the second step is the Initial Round[Figure6-21]. This step contains the operation AddRoundKey. The AddRoundKey is using the Main Random Key of 128 Bits, to XOR each case with the 128 Bits Plaintext put as well in a 4x4 Matrix. The result will give a new 4x4 Matrix for the Plaintext. Next, the third step is the Round[Figure3-4-5-6-21-22]. Using a 128 Bits Random Key, we have 10 Random Key, we need to do n-1 Rounds of the following functions in order: Sub Bytes, Shift Rows, Mix Columns and AddRoundKey. The Sub Bytes: this function will pass each case of the 128 Bits Plaintext Matrix in a Substitute Box table[Figure7], depending of the actual value, this will give us a new value for each of the case of the 128 Bits Plaintext Matrix. The Shift Rows: this will for each rows of the 128 Bits Plaintext Matrix (expect the first one) move on the left according to the number of rows we are in. The Mix Columns: this is the operation that must be the most difficult to understand in the Advanced Encryption Standard, fortunately, the task has been easier since there exist as well tables (Mul2, Mul3)[Figure7-8] like the Substitute Box, avoiding us having to re-do the algorithm ourself. And the AddRoundKey: as the Initial Round it will XOR each case of the 128 Bits Plaintext Matrix but with the expansion of the 128 Bits Random Key depending on the rounds we are in. All these steps are repeated 9 times; each time provides to increase the security of the encryption. We could try to do more rounds but it wouldn't increase the security anymore and would therefore be useless. Finally, after the Rounds, the 3 operations: Sub Bytes, Shift Rows and AddRoundKey will be done a last time, to provides a bonus security. And the Cyphertext of the Plaintext will be given.
In second part, we mainly worked on the modification of the Advanced Encryption Standard Block Cipher Algorithm. Indeed, the justification is that it was the that the Advanced Encryption Standard Block Cipher algorithm was structured like this for a good reason. Each time we were modifying the algorithm, we were obtaining different results that could corrupted the security of the message. In fact, as first task, the comprehension of the Advanced Encryption Standard Block Cipher itself was already pretty good to us, but most of the people stops researching after arriving to their goal, and we wanted to go further than this. By modifying the elements of the algorithm, we have not only opened the possibility to be able to create our own Block Cipher, but to create a better algorithm than the Advanced Encryption Standard itself. But for now this is too more ambitious to think about doing that.

3.2. TECHNICAL

Here will be the talk about the necessity of using the language Python for this project, and the different possibilities acquired thanks to the use of different operation on the Advanced Encryption Standard.
The use of Python really helps to manipulate and read the code. Indeed, it is a language called as "High Level" because it is near the mankind language. In contrary using another language such as C, would have increase the difficulty to read and code the algorithm of the Advanced Encryption Standard. The goal of this project being to understand and master the Advanced Encryption Standard, there is not point complicating the task.
For the different modes of operation used in this project, we have the ECB mode (Electronic Code Book): the use of this mode shows the inconvenient that could be encountered with it; first, it is working with independent block ciphers, which means in a case there is an error propagation in a block, this last will not be propagated on the other's blocks, which is good; but this way of encrypting alternates as well the global security of the data encrypted. In fact, tends to use data more than 128 Bits, will occur in a repetition at the end of the encryption and the data could therefore be easily read by an expert cryptographer. Next we are using the CBC mode(cipher-block-chaining): This mode, in contrary of the ECB, is using blocks that are dependent of each other's. Explicitly, each 128 Bits Cypher text block after the Advanced Encryption Standard algorithm will be XOR with our next 128 Bits Plaintext before going in the Advanced Encryption Standard Block Cipher in turn. The inconvenient consequently encountered is the error propagation, because of this dependence of the blocks the error

will be propagated in each Blocks. But this mode has as well as particularity, a little step is added at the start of the encryption to increase the security, which is the “Initialization Vector”. This Initialization Vector is a random 128 Bits key, only used once, that is XOR with our first original 128 Bits plaintext before the Advanced Encryption Standard algorithm. Finally, we have the CTR mode. This mode will help us to compare with the ECB mode, because both of them are using independent Block Ciphers. The particularity of the CTR is that it is using a Counter for each of his 128 Bits Block Cipher. This Counter will pass in the Advanced Encryption Standard Algorithm then XOR with the 128 Bits Plaintext Block and the result of the first Block Cipher will be given. Then the Counter will add one to all its 8 Bits case, and pass again in the Advanced Encryption Standard algorithm and XOR with the next plaintext.
Those 3 modes will be the mains this project will work with.

4.BSP - THE ADVANCED ENCRYPTION STANDARD [128 BITS/THREE MODES]

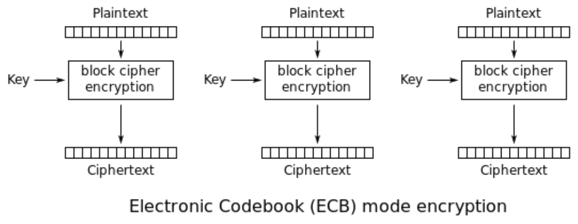
4.1. Requirements

There are several requirements that need to be satisfied in this project. The main requirement of this project is to first deliver a new wide idea about the conception of the real security image of the Advanced Encryption Standard or any other security system in the world. This project should allow the user to apprehend the easy insecurity of the transmission of information that can happen if the system is not correctly mastered or used in the best way possible. And consequently this will show the way to go to avoid having the information compromised, to provides the most secure algorithm of encryption as the Advanced Encryption Standard has. Indeed, numerous are the existing encryption system on the run today, but each of these has their particular issues. Those last being less known, for now, the Advanced Encryption Standard is considered as the one the most used and known by the mankind to secure information. But most of the people are still naïve and thinks this is a system 100 % secured, principally people which are not in the computer science domain or are in their third age. But this project, proves as for any program, that it is not as much secured as you would think.
This project should as well allow the user to be able to understand perfectly how works the Advanced Encryption Standard algorithm for each mode shown in its overall. The user should know each parts of each algorithm, and consequently being capable to compare after, by himself, the algorithm of any security system to others security system and notice the difference between those. This comprehension of the algorithm will help the user to not be easily fooled about any system swearing providing a 100 % secured system, and explain to himself why it couldn’t be possible.
By using a programming language such as Python, the user should be able to understand easily each lines written in the project at first. However, the use of this language should provide to the user a new type of programming language he didn’t know; otherwise if the user is used to it, it would help him to master it; either, if the user has never used any programming language, this is an easy way to begin with a comprehensible way of coding. Indeed, as Python can be compared to like reading English words, it is easy for anyone to understand after some efforts any code of Python. As well, by implementing along the algorithm of the Advanced Encryption Standard in Python, the user should be able, if motivated, to re-invent his own method to encrypt information. He could also ameliorate an existing one or doing a wide new one, but providing of course a high security algorithm as the Advanced Encryption Standard does.
However, this project provides the main idea of the security or insecurity of the Advanced Encryption Standard Algorithm around those three modes, there exist indeed several others study on this subject providing more deep researches on each parts and others modes than those three worked with. Consequently, some parts might not be exploited in this project, but the user could be able to find out, by himself, new results by modifying the code by himself.
This project is, as you could have noticed, mainly focused on the comparison of the result got after the Encryption of data with the Advanced Encryption Standard in the three different mode of operation. This project will in fact not considerate much about the Decryption of the Advanced Encryption Standard in any mode. Indeed, the Decryption of the Advanced Encryption Standard, only being the inverse operation, it is not worth talking about it in this project. Nevertheless, the project will still contain the different inverse operation of each mode in the algorithm studied, so the user can still have an idea on how works the inverse function of the Advanced Encryption Standard if he has some difficulties to get an idea

on how it would be done.
The project is focused on only those three modes for the Advanced Encryption Standard: ECB, CBC and CTR. The choice to choose those three is because there exist always a common and different point between them. Indeed, the two modes, ECB and CTR, use both independent block (Recall: each matrix pass independently in the Advanced Encryption Standard) while the mode CBC use dependent block. The two modes, CBC and CTR, are using an Initialization Vector (Recall: random 128 Bits matrix) while ECB don't at all.
This project will compare the different results obtained with those three different modes of operation each time. But still has said below there still exist more than those three modes for the Advanced Encryption Standard. Indeed, others modes such as PCBC, OFB or CFB are also used for the Advanced Encryption Standard, there are quite similar to the others three but still got very different way to be implemented providing better or worst results.
The actor exploiting these deliverables will mainly be the Advanced Encryption Standard, the project will work around this algorithm.

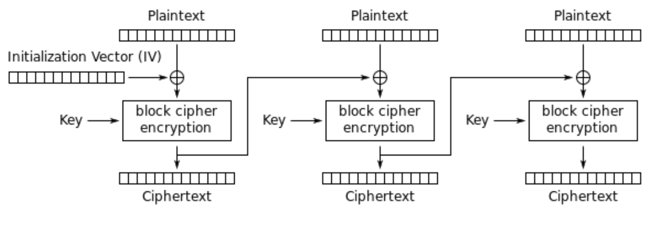
4.2. Design

The design here will give some explanation about the whole structure of the Advanced Encryption Standard and the three different modes, in addition this will show how to use each step and mode to manage to create a strong project to be able to compare the security of the Advanced Encryption Standard later. As reminder, the project is designed on those three modes ECB, CBC and CTR.
First at first, each mode will automatically have above the code, the different fixed box enumerated below such as Substitute Box, for the Sub Bytes step, and Mul2/ Mul3 for the Mix Columns step.
Begin with the ECB [Electronic Code Book] mode: the first thing first to do is to create several 4x4 matrix of 128 Bits which will contain the information to encrypt. For this project each letter of sentence of data will be put in the last case of each 128 bits' matrix, while the others 15 cases will be set to the value 0, thereby for each mode it will be sure to not only have one Block Cipher per mode. After this, a Main Random 128 Bits Key will need to be created and put as well in a 4x4 128 Bits Matrix. Since the Main Random Key is of length 128 Bits, the matrix of the Main Random Key will need to be expanded to create 10 others matrix of Random 128 Bits key that will be used in the Advanced Encryption Standard Algorithm. Next, each one of the Plaintext 128 Bits matrix will be called in a function (i.e. AESEncrypt()) containing the Advanced Encryption Standard algorithm, each Block Cipher are called one by one since there are independent. This function will call as variables: the 128 Bits matrix, the Main Random key and the expansion of the Main Random Key. Each matrix will pass one by one in the several steps enumerated upper[scientific]: Initial Round (AddRoundKey with the main random key [Recall: XOR case[i] of the matrix with the case[i] of the main random key); Rounds [1-9] (Sub Bytes with the substitute box [Recall: each case[i] will pass in a fix tab], Shift Rows, Mix Columns (Mul2, Mul3), AddRoundKey[Recall: XOR case[i] of the matrix with the case[i] of the main random key[i]); finally, the Sub Bytes, Shift Rows and AddRoundKey function will be done again. The output will be the Cypher text of the first matrix, then the others matrix will be called too with the same parameters except it will be for the second 128 Bits Matrix.



Source : Wikipedia - Advanced Encryption Standard Modes - ECB

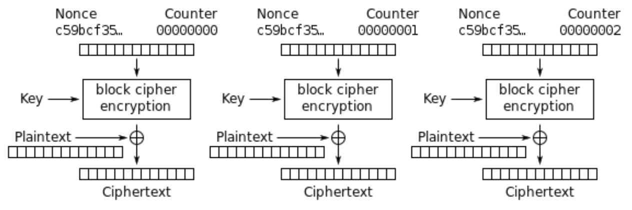
In second, the mode CBC[Cipher-Block-Chaining] is constructed: such as the ECB mode, a 4x4 matrix of 128 Bits will be created and will contains the information to encrypt. The matrix will as well contain only one letter in the last case while the others cases will be set to 0. Then, as ECB, a Random Key of 128 Bits will be created and put in 4x4 a matrix and later expanded to 10 others 128 Bits Random key. Until there, all is the same, here comes the difference to the ECB, an Initialization Vector(IV) of 128 Bits will be created and put in a 128 Bits Matrix of 4x4. This IV will be used once before going to the function of the Advanced Encryption Standard, the first 128 Bits matrix of the plaintext and the Initialization Vector will XOR themselves to give a new matrix. Next, the new first Matrix obtained by that XOR will pass in the Advanced Encryption Standard function, the function will still have as parameters the 128 Bits Matrix XORed, the Main Random Key and the 10 Matrix expansion of the Main Random Key. At the exit, as the ECB, the Cyphertext is the result for the first Plaintext Matrix. But, the particularity of the CBC is that is it using dependent block, so the Cypher text obtained will be XORed with the next matrix of plaintext (i.e. Cypher text[i] XOR Plaintext[i+1]) then the resulted matrix will pass in the Advanced Encryption Standard function as turn, and so on until the end. So the Block Cipher are clearly dependent to each other's.



Cipher Block Chaining (CBC) mode encryption

Source : Wikipedia - Advanced Encryption Standard Modes - CBC

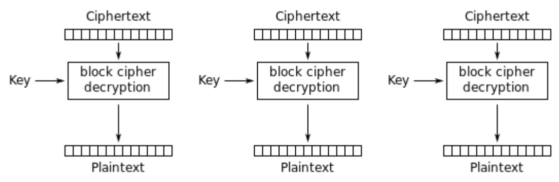
The last mode is CTR[Counter]: it creates as well as the two others modes, multiples Block Ciphers of 128 Bits containing the data and a Main Random Key with its Expansion, it is similar to ECB because it is based on independent Matrix. But there is a big difference about what is going in the Advanced Encryption Standard Algorithm. Indeed, on the contrary of the CBC, it won't create an Initialization Vector but a Counter. Indeed, a Random 64 Bits Matrix will be created and each time concatenate to a 64 Bits Counter that will give a 128 Bits Matrix. The Counter will add one to itself each time a 128 Bits Matrix is created (picture), the number of matrix created will correspond to the number of 128 Bits Matrix Plaintext created. At this point, it won't be each of the 128 Bits Plaintext Matrix that will pass in the Advanced Encryption Standard Algorithm but those 128 Bits Matrix Counter. Indeed, each one of the 128 Bits Counter Matrix will pass in the function with parameters: the 128 Bits Counter Matrix, the main random key and the expansion of the main random key. The result obtained is not yet the final result, it will need to be XORed to the matrix corresponding to the number of the Counter (i.e. Matrix Result of Counter[I] XORed with Matrix Plaintext[i]), and the result obtained is the Cypher Text for the first Matrix.



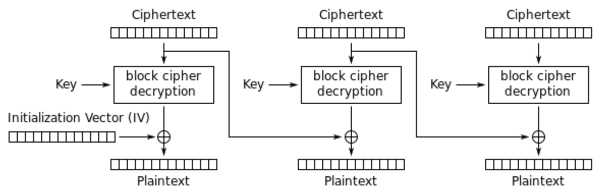
Counter (CTR) mode encryption

Source : Wikipedia - Advanced Encryption Standard Modes - CTR

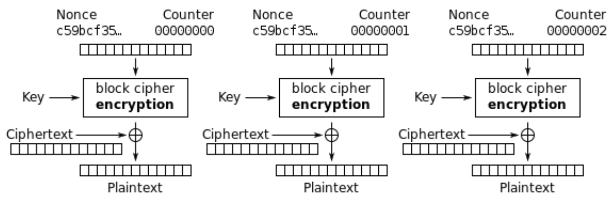
However, if the idea come to you to want to decrypt the Cypher text for each of the modes, here is a quick summary on how you are supposed to proceed to have the inverse operation. Mainly, all you would have to do is rewrite each step backwards. So it would give for the Advanced Encryption Standard Algorithm: AddRoundKey, Shift Rows, Sub Bytes (last random key); Rounds [AddRoundKey (take the random key backwards 9->1), Mix Columns (the new tab for the inverse are Mul9, Mul11, Mul14), Sub Bytes] 9->1; AddRoundKey (the Main Random Key). The XOR operation will give the inverse automatically, it is one property of this function. Still with Python, an issue has been encountered with the inverse function of the Mix Columns, the result should give as said below the inverse, but in some way, Python seems to have difficulty to manage this task, therefore the result is totally different to what is expected. To counter this difficulty, you might be tended to write yourself the algorithm to create the tab, but the problem seems to come from the Exploitation System used (i.e. Python has been designed for Mac, Windows might have difficulties with the libraries)For the different modes, it is the same principle: for ECB, just pass the Cypher Text in the Inverse Advanced Encryption Standard; for CBC, you will need to pass the result in the Inverse Advanced Encryption Standard then XOR the result with the matrix of the Cypher Text before, then pass again the result in the Inverse Advanced Encryption Standard; for the CTR, you will just need to XOR the Result obtained with the Cypher Text of the Counter after the pass of the Advanced Encryption Standard.



Electronic Codebook (ECB) mode decryption



Cipher Block Chaining (CBC) mode decryption



Counter (CTR) mode decryption

4.3. Production

In the production each step will be considerate individually in one of each of the mode: ECB, CBC and CTR. The Algorithm of each of these mode and the Advanced Encryption Standard will be teared apart and deleted for some parts on the different step to see the change got after on the result. But in the beginning, the results obtained for the three modes with the completed Advanced Encryption Standard Algorithm will be analyzed to see what kind of encryption is produced with the data. At the end, each of the three mode will be compared between them to see if there is any difference or similarities that could be observed.

First to first, the mode for the ECB. This mode, as said below in the Design, is based on an encryption by independent block, is not using any Initialization Vector and pass each Matrix one by one containing the data in the Advanced Encryption Standard, still this is where it is not good at all about this mode. Indeed, after inputting the data (“Hello World”) in the code and executing the mode, the result obtained afterward is really strange and surprising[Figure9]. In fact, we can clearly observe that some parts are like being repeated in the final result in Cypher text. To see what is going on, the clear plaintext and the scripted cypher text are being compared, and observe that the parts repeated are in fact simply the same letters. In practice if the data “Hello World” would be input, for the letter “l”, the cypher text would repeat exactly the same line three times. This is what would be called a big security ward in computer science. Since then, the mode ECB is really not good for Encrypting about this particular reason, because any expert Cryptographer or a simple intelligent person would very easily interpret your data and be able to decrypt it since most of the words have at least two same letters. Said that, the security for this mode can be still attained at two conditions: that the ECB is only used when your data doesn’t reach more than one 128 Bits Matrix, meaning your data couldn’t pass 16 characters; or possibly that your data doesn’t count the same character more than once. Indeed, when only one matrix is being computed, the data is perfectly secured, the same is observed with word that are not using twice the same letter. But it is difficult to find a bunch of words without having twice the same letter. Since today the data sent are usually bigger than 16 characters, this mode is not advisor at all for big data. To have a clear idea of what it would give you and that the insecurity is real, the data input can be a picture[Figure15], indeed there is no difference since a picture is bunch of data reinterpreted. After inputting the data of any picture, the picture can be observed as slightly blurry on the side but still readable after all[Figure16], that really show how bad is the ECB mode.

In second, there is the CBC mode. This mode, as said below in the Design, is this time based on an Encryption by using dependent blocks for the data, it is as well generating an Initialization Vector helping the security to already increase at the start before to pass as ECB the data in the Advanced Encryption Standard Algorithm. This mode is beside really different from the ECB in all parts, except for the Advanced Encryption Standard Algorithm. This is the first mode from now which could be considered as quite secured to use. Indeed, after inputting the data (“Hello World”) in the code and executing the mode, the result obtained is really good looking and nicely randomly scripted as we would expect from a scripting mode[Figure11]. With this mode, the observable result doesn’t give any repeated line at all at the end. Even after comparing the plaintext and the cypher text, there is no possibilities for anyone to interpret any line of the clear data with the result. Since then, this mode could be considered as efficient at first sight for this reason. But, there is always a ward. Indeed, even if the scripted data is looking really good, it can happen that some bits can be compromised after executing and bring afterward errors to a block, why? Because computer science error can occur anytime, anywhere for any reason. As we know, the CBC is based on dependent block so those blocks are connected between them all along the mode and this can lead to a propagation error. Indeed, if the error occurs in the first block, all the next block will be affected as well, and the error will expand consequently until the end. Still, there exist some mode that can automatically take care of these error and get rid of it, but not in this case unfortunately. Anyway, this is the only ward that could be considered to happen to your data, in other way the CBC mode is highly recommended for encrypting any type of data. As for ECB, to have a clear idea of the real security this mode can bring to you, the same picture[Figure15] used for ECB has been used and input in the CBC mode. For result, the resulted picture is not slightly blurry as ECB would provide, but completely illegible or unrecognizable, this proving the efficient of the CBC.[Figure17]

And for the last but not least, there is the CTR mode. This mode, as said below in the Design, is based as well as the ECB mode on an Encryption using independent Blocks for the data, but this time it doesn't use an Initialization Vector such as CBC does, but will create multiples Blocks with a same 64 Bits random matrix concatenated with a Counter. The difference of the CTR with the two other modes, is observe from what is passing in the Advanced Encryption Standard Algorithm. The data is indeed input in several 128 Bits Block, but this is the Counter that is passing in the Advanced Encryption Standard Algorithm this time and not the data anymore. Besides that, it is quite similar to the CBC from the point of view of the results. Indeed, after inputting the data and executing the mode, the result seems to give satisfying result as good as the CBC mode does[Figure13]. Indeed, there is as well not sign of repetition or any issues leading to an insecurity coming from this mode when observing the output. Since then, this mode can be considered as good as the CBC. But the CTR has an advantage compared to it. As said for the CBC, some error propagation can occur from time to time, but here the CTR is using independent block, so consequently if an error would occur, this wouldn't spread to the other blocks of the mode. In conclusion, the CTR is better than CBC concerning the error propagation error, but could as well lead to the same time of security obtain with the CBC. To be absolutely sure of the security provided, a picture can as well be input in this mode, and it gives as well a perfectly scripted picture illegible and unrecognizable as CBC does[Figure17].

4.4. Assessment

Here will be presented the objectives attained with this project and those which could have been improved. This will mirror the requirements announced below and explain why it has or not be attained. For the global situation, the requirements as been indeed reach in some way. This project has provided for any user to observe the real difference and issues of each of the mode announced below: ECB, CBC and CTR. We could have observed that some modes bring some advantages beside other don't or according to the way you use it properly or not. Each of the mode has been explained well, with all the parts contained each of these and explaining how to use it or code it all along. The user should have in mind how works perfectly each mode, and be capable to construct it back by himself.

The only partial bad thing is the Decryption part for each of the modes. Indeed, normally a Decryption way, the inverse function, should have worked well for each of the mode. But a problem occurred when using the inverse function for the Mix Columns, the need to use the inverse fix table, it was giving different result from what we were expecting providing a false Decryption[Figure10-12]. Still, the CTR is the only mode providing a correct decryption, indeed it is simply because the algorithm doesn't need to pass again by the Advanced Encryption Standard Algorithm, but just need to be XORed with the result to obtain back the data. A solution to this was to create ourselves the inverse fixed tab for the inverse table for the Mix Columns, since the Encryption table was giving the result expected, the problem was coming from this side. Still it didn't give any good result either. The solution to this is probably because the programming language Python has been designed for Linux. Since this project has been done on a Windows device, it would explain that Python has difficulties to interpret normally the code. In the design part was said: not talking for the Decryption part. But it was absolutely necessary to talk about the issue with the Decryption. Indeed, at first the Decryption part was just the inverse function, and the need to talk about it was not necessary, but having encountered a problem of compiling, it is better to warn the user about what would happen if he was using Windows. The user shouldn't have discovered it by himself, otherwise he would have doubt about this project and wouldn't trust it anymore because he would fell a little under unformatted.
But as well, this project brought some knowledge in the programming language python. Indeed, not a lot of people are interested in security and this project provided a way to discover how would be coded a basic security algorithm. Python being really simple, this is the best way to learn to implement with the people, without those being confused about the writing of the code or the anarchic structure we could have encountered with another language more complicated.

5.CONCLUSION

This project has been successively realized and bring what the user was expecting for. Add to that, the user has the possibilities to re-work the project, create new security mode and evaluating himself the good or bad of any security system in its globalist. Beside the but avoid at all cost using ECB. But still, as advice, don’t trust any encrypting algorithm of anyone so far. If a code has been written, it is possible that it can be broken by any Hacker, no application is 100% secured.

References

[1] Wikipedia, help to discerning all the different of the Advanced Encryption Standard : https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

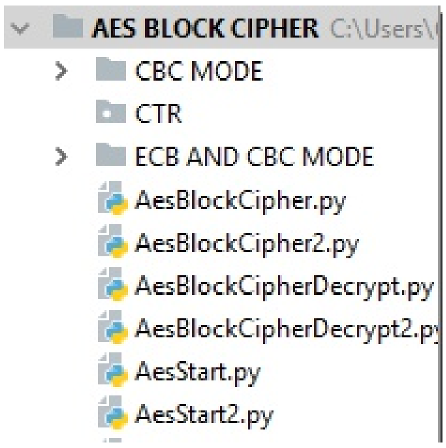
[2] Wikipedia, explanation of the working of the Advanced Encryption Standard (Steps, Structure):https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard

[3] Web page that help discerning each step of the Advanced Encryption Standard individually and how they were working and what they were bringing to the algorithm: http://aescryptography.blogspot.be

[4][5][6][7][8]Multiples of outdoor works from others students or University research around the world to apprehend efficiently the Advanced Encryption Standard : https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf
http://www.montefiore.ulg.ac.be/~dumont/pdf/crypto.pdf
http://www.emse.fr/~dutertre/documents/synth_AES128.pdf
http://www.math.wisc.edu/~boston/nover.pdf
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

[9] Helped to discern by a “FUN” way how was working each step of the Advanced Encryption Standard as well as the scientific explanation behind each step: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

APPENDIX



AES BLOCK CIPHER C:\Users\
CBC MODE
CTR
ECB AND CBC MODE
AesBlockCipher.py
AesBlockCipher2.py
AesBlockCipherDecrypt.py
AesBlockCipherDecrypt2.p
AesStart.py
AesStart2.py

Figure 1 : Structure for the implementation of the three modes [ECB/CBC/CTR] + the Encryption and Decryption of the Advanced Encryption Standard Block Cipher for each of the

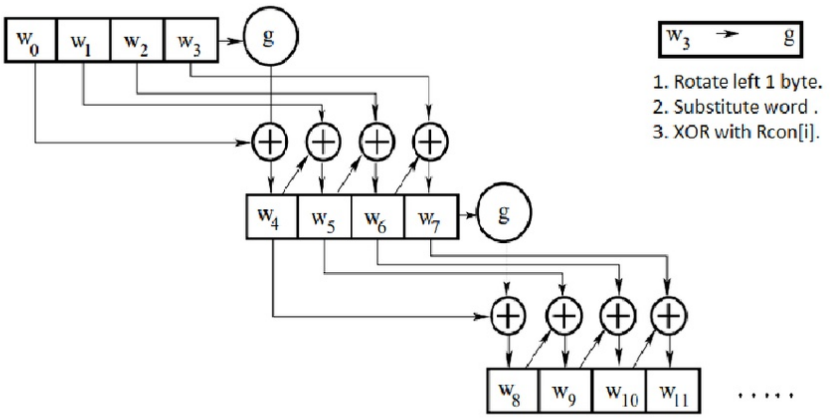


Figure 2 : Explanation for the Expansion of the Main Random Key
Source : Cornell University Electrical and Computer Engineering - ECE 5760 Final Project - AES Encrypter/Decrypter

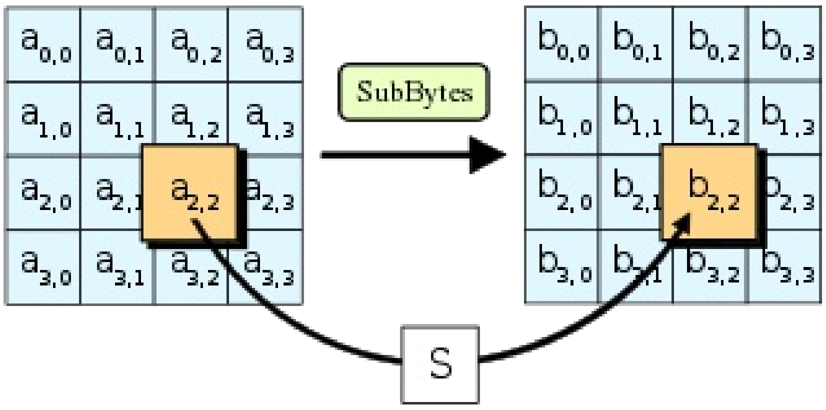

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

Figure 3 : Explanation of the function SubBytes for the Advanced Encryption Standard Algorithm
Source : aescryptography.blogspot.be, 2012.

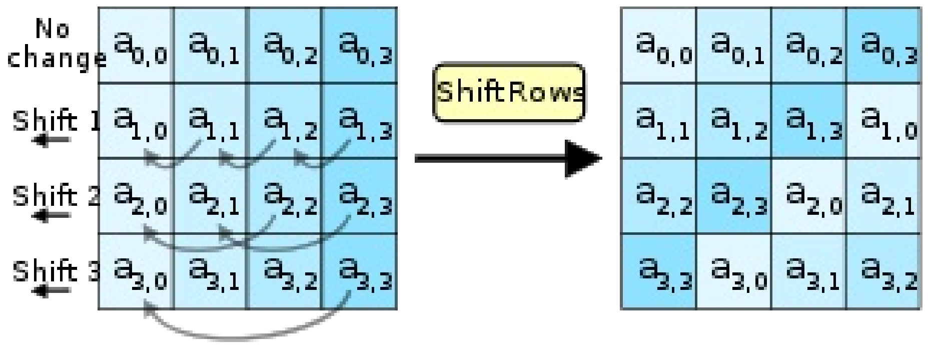

No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 4 : Explanation of the function ShiftRows for the Advanced Encryption Standard Algorithm
Source : aescryptography.blogspot.be, 2012.

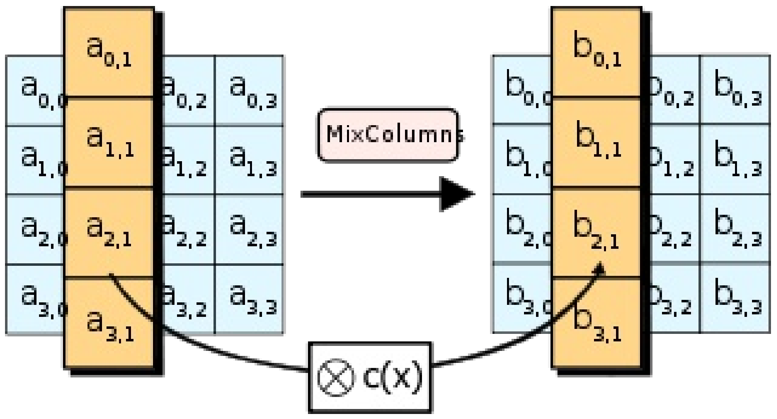

MixColumns
$\otimes c(x)$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 5 : Explanation of the function MixColumns for the Advanced Encryption Standard Algorithm
Source : aescryptography.blogspot.be, 2012.

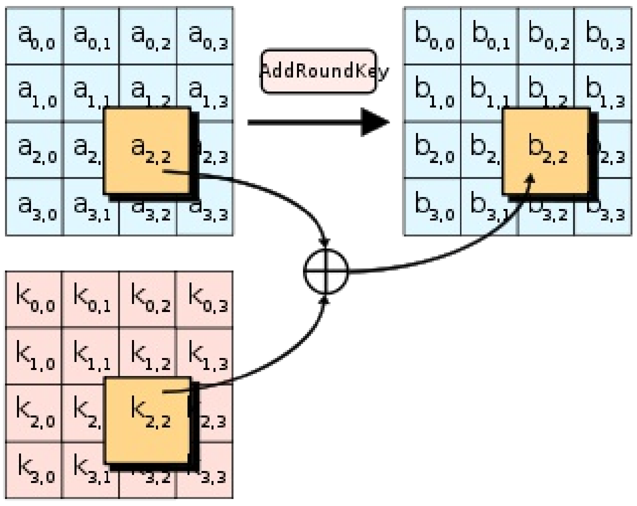
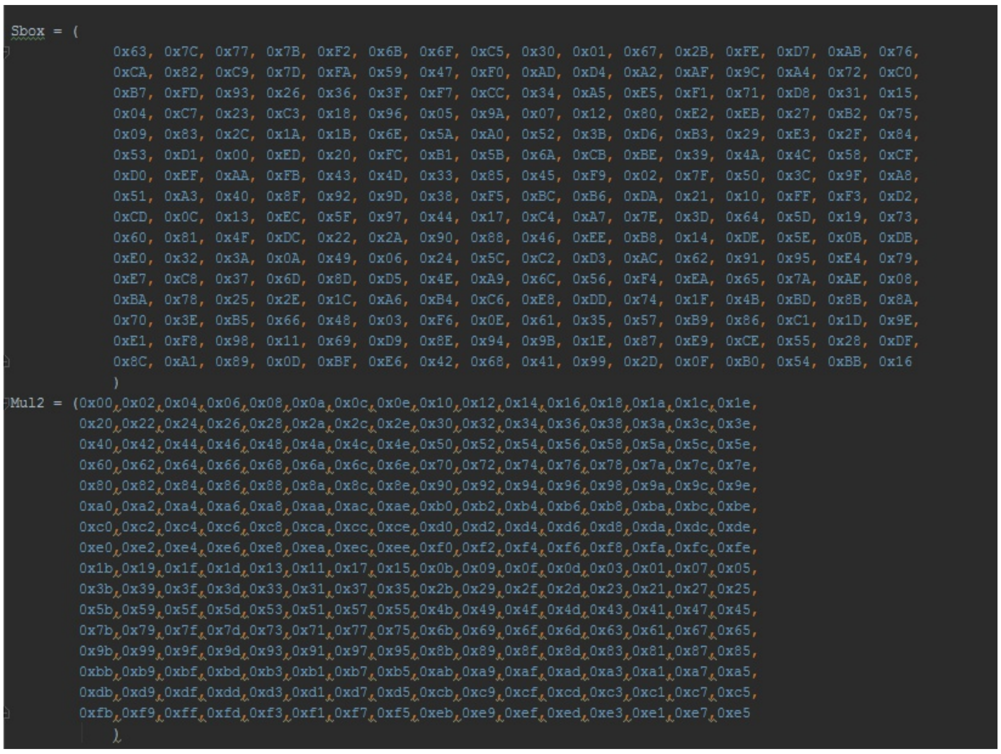
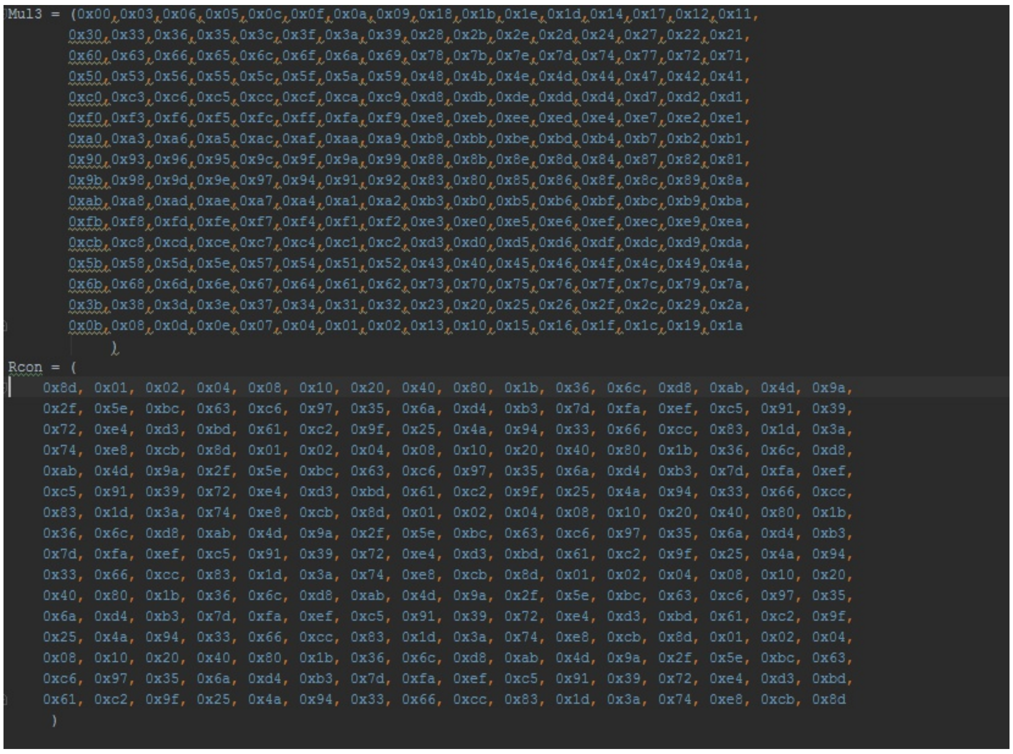

AddRoundKey
$\oplus$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$

Figure 6 : Explanation of the function AddRoundKey for the Advanced Encryption Standard Algorithm
Source : aescryptography.blogspot.be, 2012.



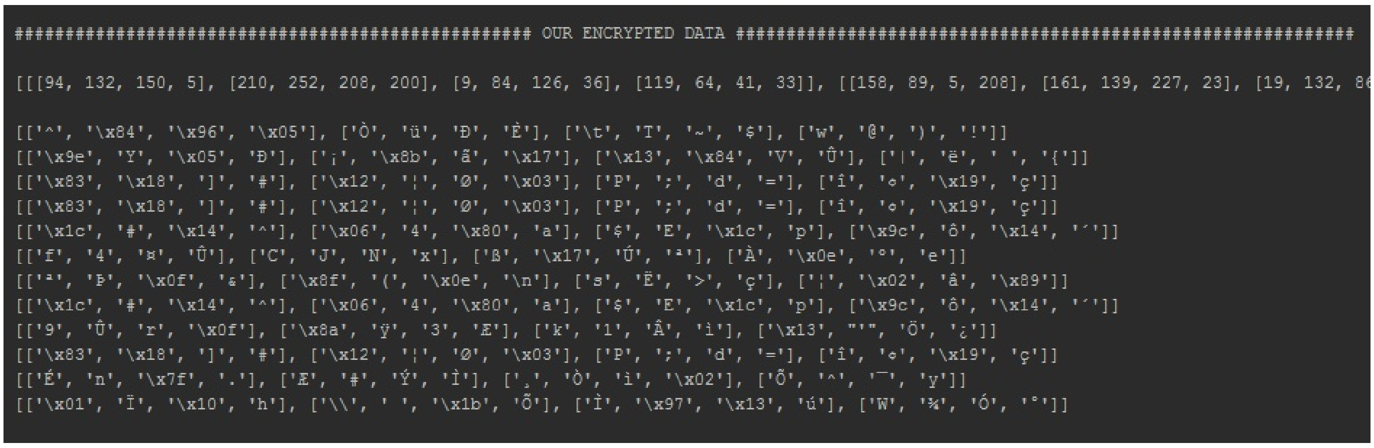
```
Sbox = (
            0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
            0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
            0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
            0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
            0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
            0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
            0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
            0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
            0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
            0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
            0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
            0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
            0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
            0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
            0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
            0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
            )
Mul2 = (0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
        0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
        0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
        0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
        0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
        0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
        0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
        0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
        0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
        0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
        0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
        0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
        0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
        0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
        0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
        0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
            ),
```

Figure 7 : Representation of the Substitute Box [Sbox] for the AddRoundKey step + representation of the Mul2 table for the MixColumns Step
Source : Wikipedia - Advanced Encryption Standard Encryption - Rinjdael MixColumns



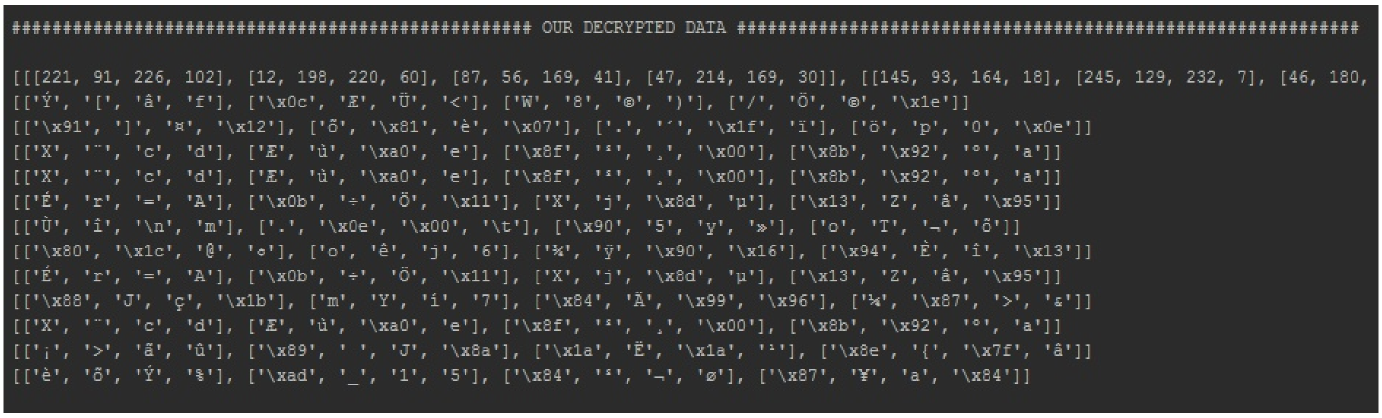
```
Mul3 = (0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
        0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
        0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
        0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
        0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
        0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
        0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
        0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
        0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
        0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
        0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
        0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
        0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
        0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
        0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
        0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
            ),
Rcon = (
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
 )
```

Figure 8 : Representation of the Mul3 table for the MixColumns step + representation of the Rcon table for the Expansion of the Main Random Key Step
Source : Wikipedia - Advanced Encryption Standard Encryption - Rinjdael MixColumns and Wikipedia - Rinjdael Key Schedule



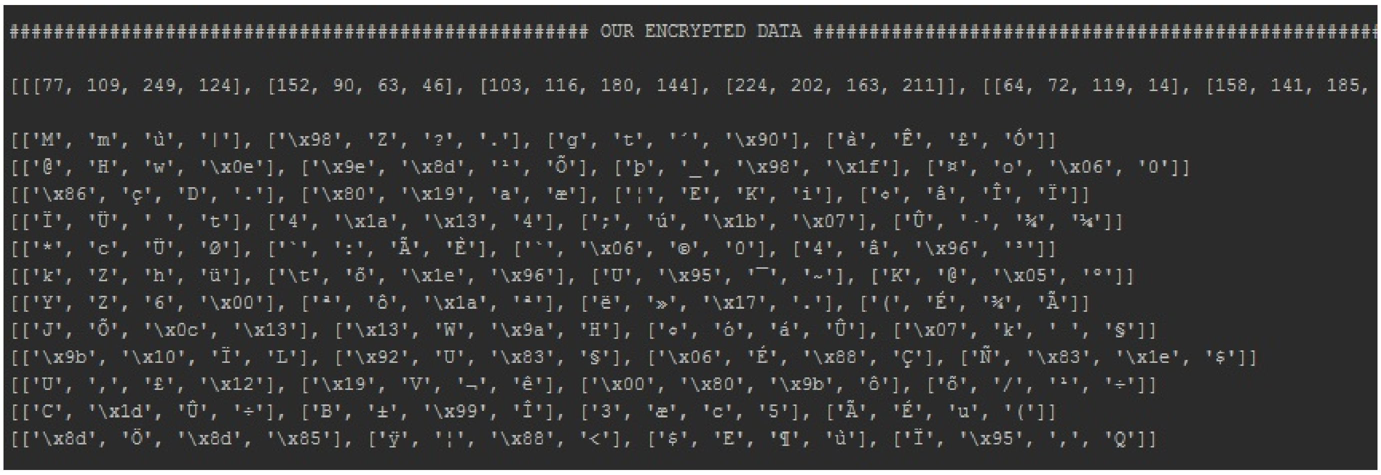
```
####################################################### OUR ENCRYPTED DATA ##################################################################

[[[94, 132, 150, 5], [210, 252, 208, 200], [9, 84, 126, 36], [119, 64, 41, 33]], [[158, 89, 5, 208], [161, 139, 227, 23], [19, 132, 8

[['^', '\x84', '\x96', '\x05'], ['Ò', 'ü', 'Ð', 'È'], ['\t', 'T', '~', '$'], ['w', '@', ')', '!']]
[['\x9e', 'Y', '\x05', 'Ð'], ['¡', '\x8b', 'ã', '\x17'], ['\x13', '\x84', 'V', 'Û'], ['|', 'ë', ' ', '{']]
[['\x83', '\x18', ']', '#'], ['\x12', '¦', 'Ø', '\x03'], ['P', ';', 'd', '='], ['î', '¤', '\x19', 'ç']]
[['\x83', '\x18', ']', '#'], ['\x12', '¦', 'Ø', '\x03'], ['P', ';', 'd', '='], ['î', '¤', '\x19', 'ç']]
[['\x1c', '#', '\x14', '^'], ['\x06', '4', '\x80', 'a'], ['$', 'E', '\x1c', 'p'], ['\x9c', 'ô', '\x14', '´']]
[['f', '4', '¤', 'Û'], ['C', 'J', 'N', 'x'], ['ß', '\x17', 'Ú', 'ª'], ['À', '\x0e', '°', 'e']]
[['ª', 'Þ', '\x0f', '&'], ['\x8f', '(', '\x0e', '\n'], ['s', 'Ë', '>', 'ç'], ['¦', '\x02', 'â', '\x89']]
[['\x1c', '#', '\x14', '^'], ['\x06', '4', '\x80', 'a'], ['$', 'E', '\x1c', 'p'], ['\x9c', 'ô', '\x14', '´']]
[['9', 'Û', 'r', '\x0f'], ['\x8a', 'ÿ', '3', 'Æ'], ['k', '1', 'Â', 'ì'], ['\x13', "'", 'Ö', '¿']]
[['\x83', '\x18', ']', '#'], ['\x12', '¦', 'Ø', '\x03'], ['P', ';', 'd', '='], ['î', '¤', '\x19', 'ç']]
[['É', 'n', '\x7f', '.'], ['Æ', '#', 'Ý', 'Ì'], ['¸', 'Ò', 'ì', '\x02'], ['Õ', '^', '¯', 'y']]
[['\x01', 'Ï', '\x10', 'h'], ['\\', ' ', '\x1b', 'Õ'], ['Ì', '\x97', '\x13', 'ú'], ['W', '¾', 'Ó', '°']]
```

Figure 9 : Result for the Encryption of the data for the ECB



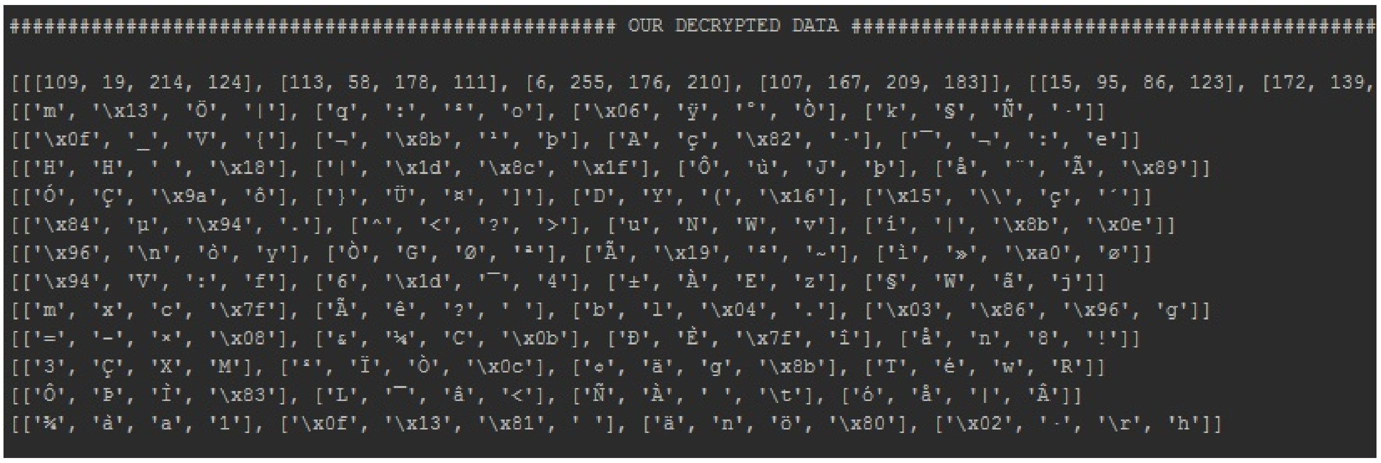
```
####################################################### OUR DECRYPTED DATA ##################################################################

[[[221, 91, 226, 102], [12, 198, 220, 60], [87, 56, 169, 41], [47, 214, 169, 30]], [[145, 93, 164, 18], [245, 129, 232, 7], [46, 180,
[['Ý', '[', 'â', 'f'], ['\x0c', 'Æ', 'Ü', '<'], ['W', '8', '©', ')'], ['/', 'Ö', '©', '\x1e']]
[['\x91', ']', '¤', '\x12'], ['õ', '\x81', 'è', '\x07'], ['.', '´', '\x1f', 'ï'], ['ö', 'p', '0', '\x0e']]
[['X', '¨', 'c', 'd'], ['Æ', 'ù', '\xa0', 'e'], ['\x8f', 'ª', '¸', '\x00'], ['\x8b', '\x92', '°', 'a']]
[['X', '¨', 'c', 'd'], ['Æ', 'ù', '\xa0', 'e'], ['\x8f', 'ª', '¸', '\x00'], ['\x8b', '\x92', '°', 'a']]
[['É', 'r', '=', 'A'], ['\x0b', '÷', 'Ö', '\x11'], ['X', 'j', '\x8d', 'µ'], ['\x13', 'Z', 'â', '\x95']]
[['Ù', 'î', '\n', 'm'], ['.', '\x0e', '\x00', '\t'], ['\x90', '5', 'y', '»'], ['o', 'T', '¬', 'õ']]
[['\x80', '\x1c', '@', '¤'], ['o', 'ê', 'j', '6'], ['¾', 'ÿ', '\x90', '\x16'], ['\x94', 'È', 'î', '\x13']]
[['É', 'r', '=', 'A'], ['\x0b', '÷', 'Ö', '\x11'], ['X', 'j', '\x8d', 'µ'], ['\x13', 'Z', 'â', '\x95']]
[['\x88', 'J', 'ç', '\x1b'], ['m', 'Y', 'í', '7'], ['\x84', 'Ä', '\x99', '\x96'], ['¼', '\x87', '>', '&']]
[['X', '¨', 'c', 'd'], ['Æ', 'ù', '\xa0', 'e'], ['\x8f', 'ª', '¸', '\x00'], ['\x8b', '\x92', '°', 'a']]
[['¡', '>', 'ã', 'û'], ['\x89', ' ', 'J', '\x8a'], ['\x1a', 'Ë', '\x1a', '¹'], ['\x8e', '{', '\x7f', 'â']]
[['è', 'õ', 'Ý', '%'], ['\xad', '_', '1', '5'], ['\x84', 'ª', '¬', 'ø'], ['\x87', '¥', 'a', '\x84']]
```

Figure 10 : Result [FALSE] for the Decryption of the data for the ECB



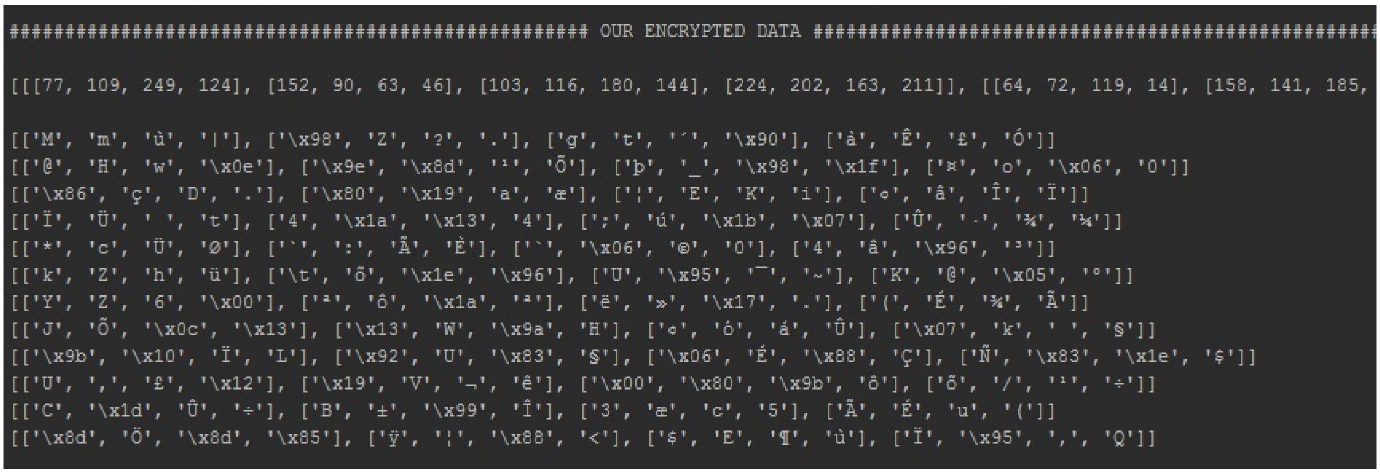
```
####################################################### OUR ENCRYPTED DATA ##################################################################

[[[77, 109, 249, 124], [152, 90, 63, 46], [103, 116, 180, 144], [224, 202, 163, 211]], [[64, 72, 119, 14], [158, 141, 185,

[['M', 'm', 'ù', '|'], ['\x98', 'Z', '?', '.'], ['g', 't', '´', '\x90'], ['à', 'Ê', '£', 'Ó']]
[['@', 'H', 'w', '\x0e'], ['\x9e', '\x8d', '¹', 'Õ'], ['þ', '_', '\x98', '\x1f'], ['¤', 'o', '\x06', '0']]
[['\x86', 'ç', 'D', '.'], ['\x80', '\x19', 'a', 'æ'], ['¦', 'E', 'K', 'i'], ['¤', 'â', 'Î', 'Ï']]
[['Ï', 'Ü', ' ', 't'], ['4', '\x1a', '\x13', '4'], [';', 'ú', '\x1b', '\x07'], ['Û', '·', '¾', '¼']]
[['*', 'c', 'Ü', 'Ø'], ['`', ':', 'Ã', 'È'], ['`', '\x06', '©', '0'], ['4', 'â', '\x96', '³']]
[['k', 'Z', 'h', 'ü'], ['\t', 'õ', '\x1e', '\x96'], ['U', '\x95', '¯', '~'], ['K', '@', '\x05', '°']]
[['Y', 'Z', '6', '\x00'], ['ª', 'ô', '\x1a', 'ª'], ['ë', '»', '\x17', '.'], ['(', 'É', '¾', 'Ã']]
[['J', 'Õ', '\x0c', '\x13'], ['\x13', 'W', '\x9a', 'H'], ['¤', 'ó', 'á', 'Û'], ['\x07', 'k', ' ', '§']]
[['\x9b', '\x10', 'Ï', 'L'], ['\x92', 'U', '\x83', '§'], ['\x06', 'É', '\x88', 'Ç'], ['Ñ', '\x83', '\x1e', '$']]
[['U', ',', '£', '\x12'], ['\x19', 'V', '¬', 'ê'], ['\x00', '\x80', '\x9b', 'ô'], ['õ', '/', '¹', '÷']]
[['C', '\x1d', 'Û', '÷'], ['B', '±', '\x99', 'Î'], ['3', 'æ', 'c', '5'], ['Ã', 'É', 'u', '(']]
[['\x8d', 'Ö', '\x8d', '\x85'], ['ÿ', '¦', '\x88', '<'], ['$', 'E', '¶', 'ù'], ['Ï', '\x95', ',', 'Q']]
```

Figure 11 : Result for the Encryption of the data for the CBC mode



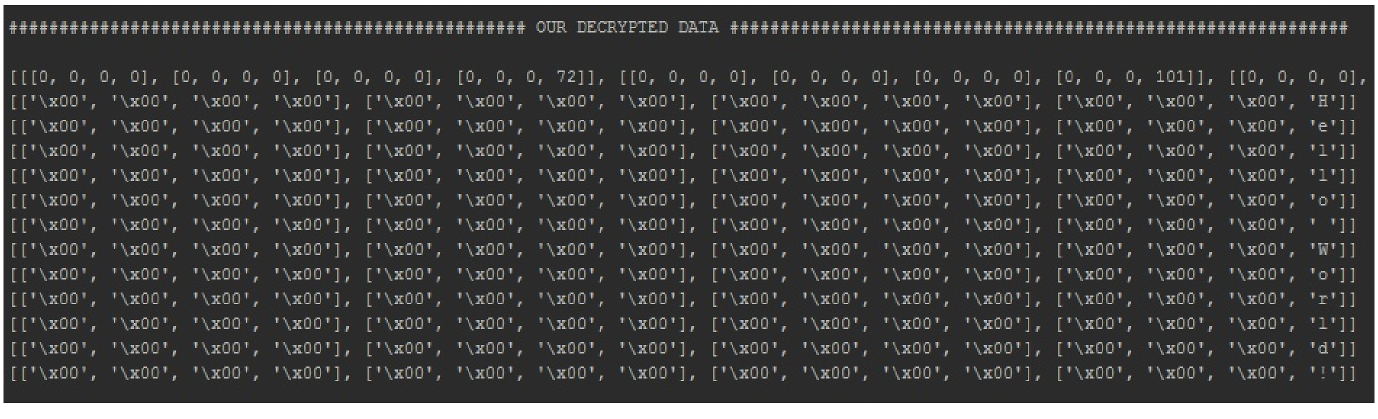
```
######################################################## OUR DECRYPTED DATA ########################################################

[[[109, 19, 214, 124], [113, 58, 178, 111], [6, 255, 176, 210], [107, 167, 209, 183]], [[15, 95, 86, 123], [172, 139,
[['m', '\x13', 'Ö', '|'], ['q', ':', 'ª', 'o'], ['\x06', 'ÿ', '°', 'Ò'], ['k', '§', 'Ñ', '·']]
[['\x0f', '_', 'V', '{'], ['¬', '\x8b', '¹', 'þ'], ['A', 'ç', '\x82', '·'], ['¯', '¬', ':', 'e']]
[['H', 'H', ' ', '\x18'], ['|', '\x1d', '\x8c', '\x1f'], ['Ô', 'ù', 'J', 'þ'], ['å', '¨', 'Ã', '\x89']]
[['Ó', 'Ç', '\x9a', 'ô'], ['}', 'Ü', '¤', ']'], ['D', 'Y', '(', '\x16'], ['\x15', '\\', 'ç', '´']]
[['\x84', 'µ', '\x94', '.'], ['^', '<', '?', '>'], ['u', 'N', 'W', 'v'], ['í', '|', '\x8b', '\x0e']]
[['\x96', '\n', 'ò', 'y'], ['Ò', 'G', 'Ø', 'ª'], ['Ã', '\x19', 'ª', '~'], ['ì', '»', '\xa0', 'ø']]
[['\x94', 'V', ':', 'f'], ['6', '\x1d', '¯', '4'], ['±', 'À', 'E', 'z'], ['§', 'W', 'ã', 'j']]
[['m', 'x', 'c', '\x7f'], ['Å', 'ê', '?', ' '], ['b', 'l', '\x04', '.'], ['\x03', '\x86', '\x96', 'g']]
[['=', '-', '×', '\x08'], ['&', '¼', 'C', '\x0b'], ['Ð', 'È', '\x7f', 'î'], ['å', 'n', '8', '!']]
[['3', 'Ç', 'X', 'M'], ['ª', 'Ï', 'Ò', '\x0c'], ['¢', 'ä', 'g', '\x8b'], ['T', 'é', 'w', 'R']]
[['Ô', 'Þ', 'Ì', '\x83'], ['L', '¯', 'â', '<'], ['Ñ', 'À', ' ', '\t'], ['ó', 'å', '|', 'Â']]
[['¾', 'à', 'a', '1'], ['\x0f', '\x13', '\x81', ' '], ['ä', 'n', 'ö', '\x80'], ['\x02', '·', '\r', 'h']]
```

Figure 12 : Result [FALSE] for the Decryption of the data for the CBC



```
###################################################### OUR ENCRYPTED DATA ########################################################

[[[77, 109, 249, 124], [152, 90, 63, 46], [103, 116, 180, 144], [224, 202, 163, 211]], [[64, 72, 119, 14], [158, 141, 185,

[['M', 'm', 'ù', '|'], ['\x98', 'Z', '?', '.'], ['g', 't', '´', '\x90'], ['à', 'Ê', '£', 'Ó']]
[['@', 'H', 'w', '\x0e'], ['\x9e', '\x8d', '¹', 'Õ'], ['þ', '_', '\x98', '\x1f'], ['¤', 'o', '\x06', '0']]
[['\x86', 'ç', 'D', '.'], ['\x80', '\x19', 'a', 'æ'], ['¦', 'E', 'K', 'i'], ['¢', 'â', 'Î', 'Ï']]
[['Ï', 'Ü', ' ', 't'], ['4', '\x1a', '\x13', '4'], [';', 'ú', '\x1b', '\x07'], ['Û', '·', '¾', '¼']]
[['*', 'c', 'Ü', 'Ø'], ['`', ':', 'Ã', 'È'], ['`', '\x06', '©', '0'], ['4', 'â', '\x96', '³']]
[['k', 'Z', 'h', 'ü'], ['\t', 'õ', '\x1e', '\x96'], ['U', '\x95', '¯', '~'], ['K', '@', '\x05', '°']]
[['Y', 'Z', '6', '\x00'], ['ª', 'ô', '\x1a', 'ª'], ['ë', '»', '\x17', '.'], ['(', 'É', '¾', 'Ã']]
[['J', 'Õ', '\x0c', '\x13'], ['\x13', 'W', '\x9a', 'H'], ['¢', 'ó', 'á', 'Û'], ['\x07', 'k', ' ', '§']]
[['\x9b', '\x10', 'Ï', 'L'], ['\x92', 'U', '\x83', '§'], ['\x06', 'É', '\x88', 'Ç'], ['Ñ', '\x83', '\x1e', '$']]
[['U', ',', '£', '\x12'], ['\x19', 'V', '¬', 'ê'], ['\x00', '\x80', '\x9b', 'ô'], ['õ', '/', '¹', '÷']]
[['C', '\x1d', 'Û', '÷'], ['B', '±', '\x99', 'Î'], ['3', 'æ', 'c', '5'], ['Ã', 'É', 'u', '(']]
[['\x8d', 'Ö', '\x8d', '\x85'], ['ÿ', '¦', '\x88', '<'], ['$', 'E', '¶', 'ù'], ['Ï', '\x95', ',', 'Q']]
```

Figure 13 : Result for the Encryption of the data for the CTR mode



```
################################################ OUR DECRYPTED DATA ##########################################################

[[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 72]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 101]], [[0, 0, 0, 0],
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'H']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'e']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'l']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'l']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'o']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', ' ']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'W']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'o']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'r']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'l']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', 'd']]
[['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '\x00'], ['\x00', '\x00', '\x00', '!']]
```

Figure 14 : Result for the Decryption of the data for the CTR mode

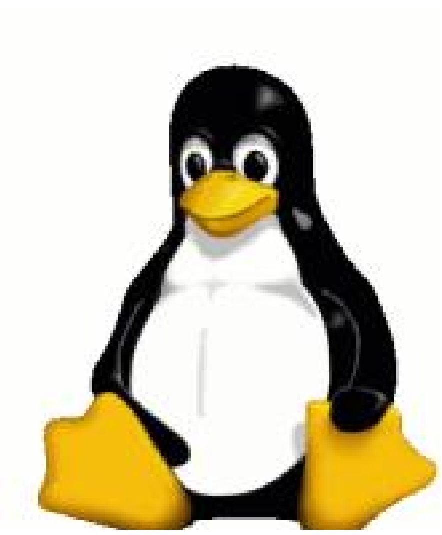


Figure 15 : Original picture containing the data tha will be Encrypted by the different modes. **Source :** Wikipedia - Advanced Encryption Standard Modes.

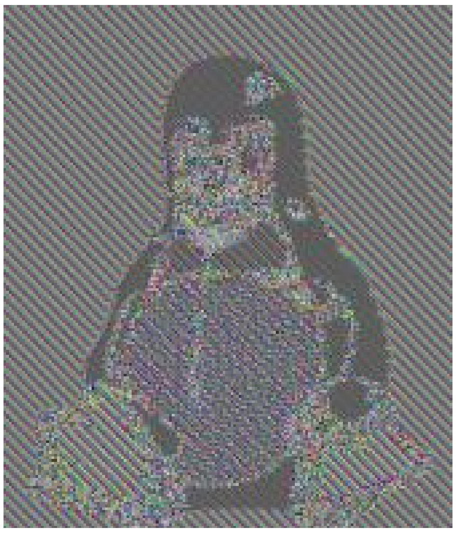


Figure 16 : Result of the Original Picture Encrypted with the ECB mode. **Source :** Wikipedia - Advanced Encryption Standard Modes.

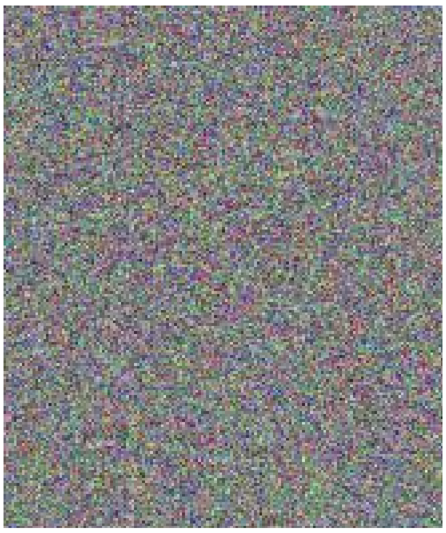
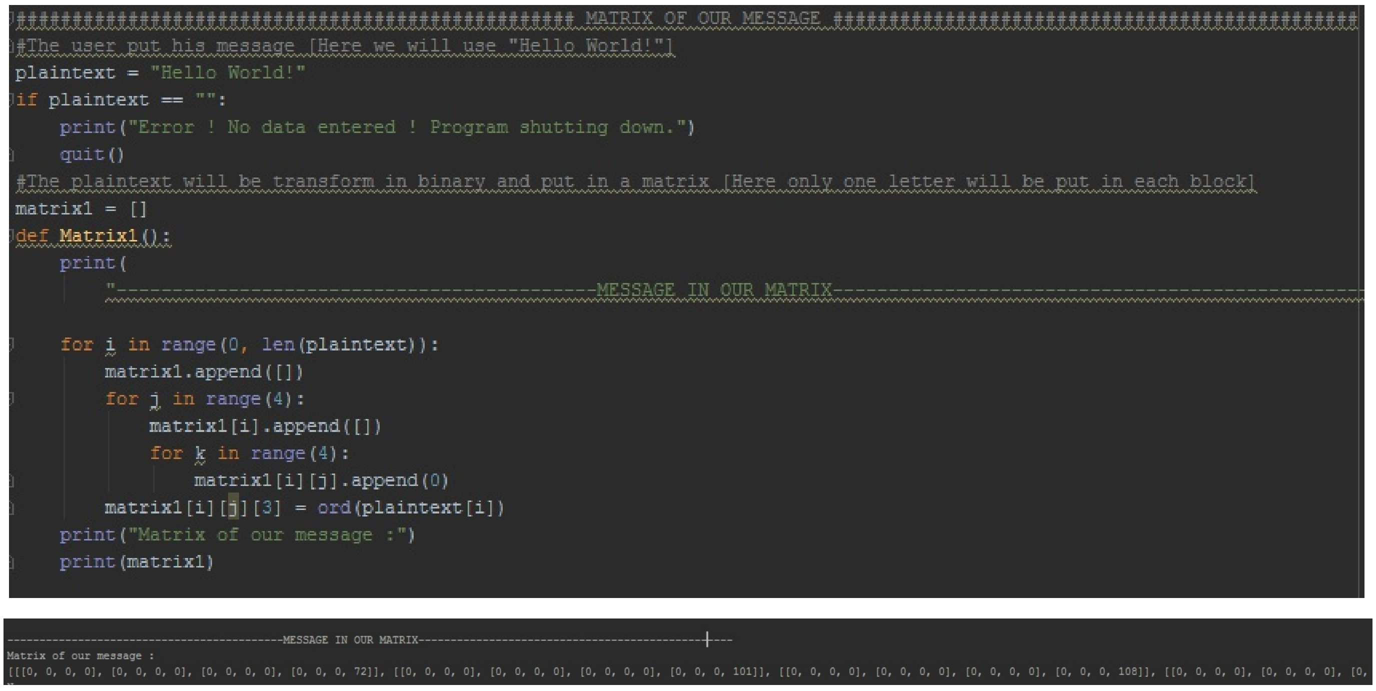


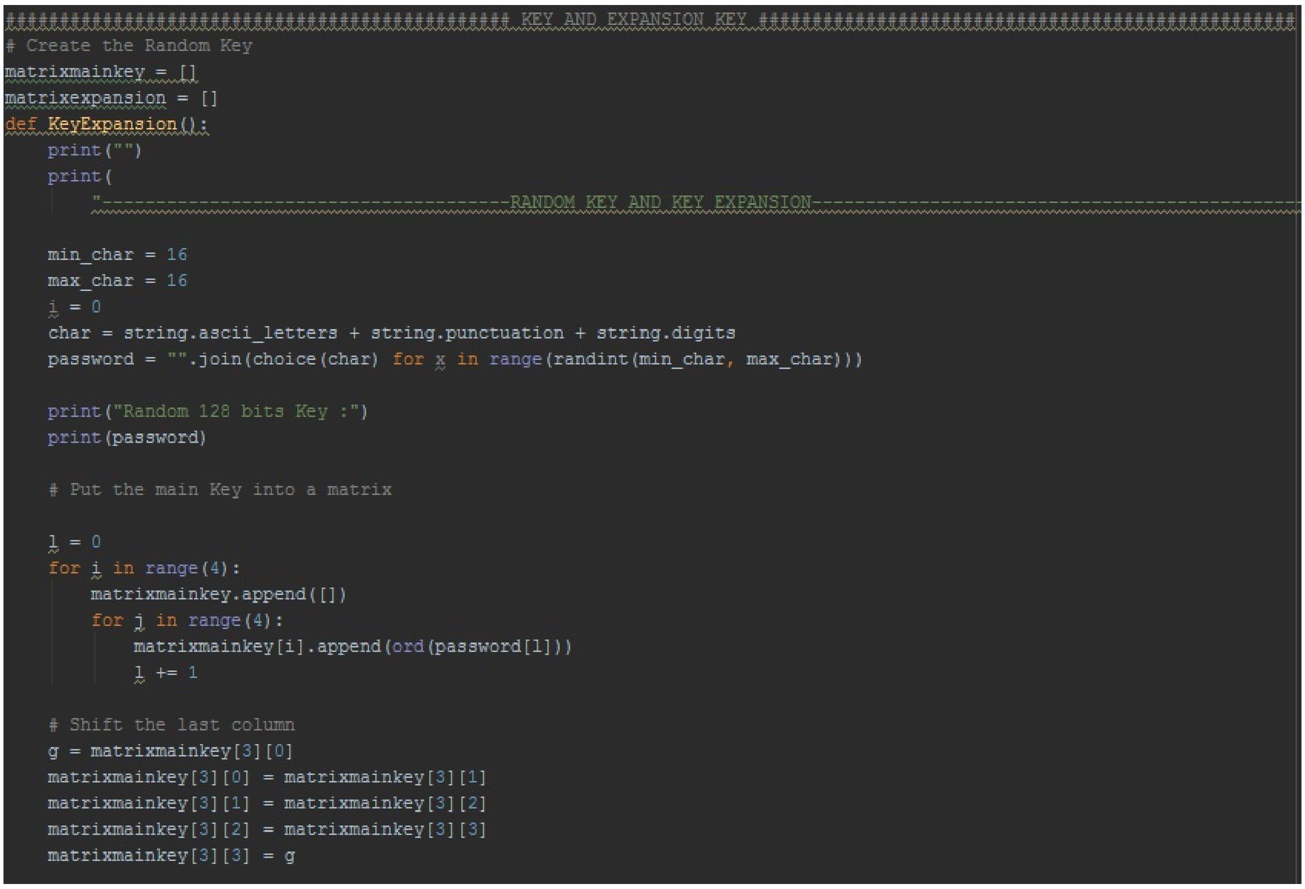
Figure 17 : Result of the Original Picture Encrypted with the CBC and CTR mode. **Source :** Wikipedia - Advanced Encryption Standard Modes.



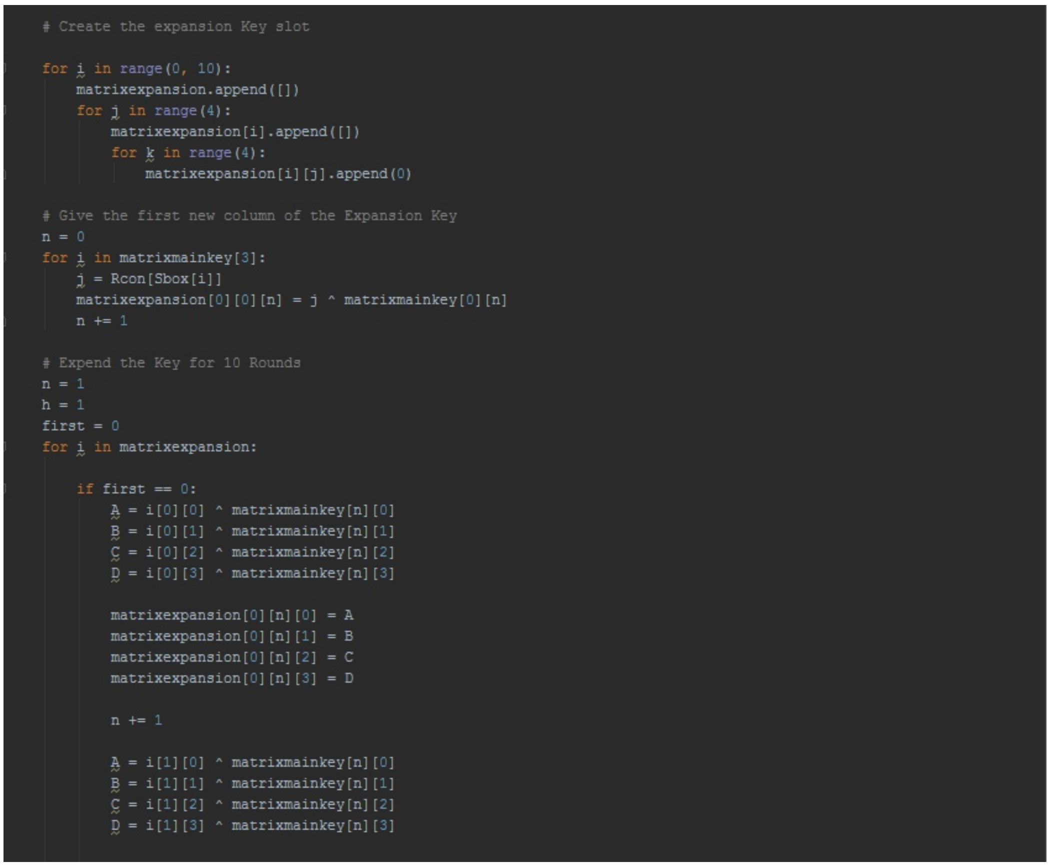
```
######################################################## MATRIX OF OUR MESSAGE ########################################################
#The user put his message [Here we will use "Hello World!"]
plaintext = "Hello World!"
if plaintext == "":
    print("Error ! No data entered ! Program shutting down.")
    quit()
#The plaintext will be transform in binary and put in a matrix [Here only one letter will be put in each block]
matrix1 = []
def Matrix1():
    print(
        "-------------------------------------------MESSAGE IN OUR MATRIX---------------------------------------------

    for i in range(0, len(plaintext)):
        matrix1.append([])
        for j in range(4):
            matrix1[i].append([])
            for k in range(4):
                matrix1[i][j].append(0)
        matrix1[i][j][3] = ord(plaintext[i])
    print("Matrix of our message :")
    print(matrix1)
```

```
------------------------------------------MESSAGE IN OUR MATRIX-------------------------------------------
Matrix of our message :
[[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 72]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 101]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 108]], [[0, 0, 0, 0], [0, 0, 0, 0], [0,
```

Figure 18 : Implementation of the Matrix containing the data + result

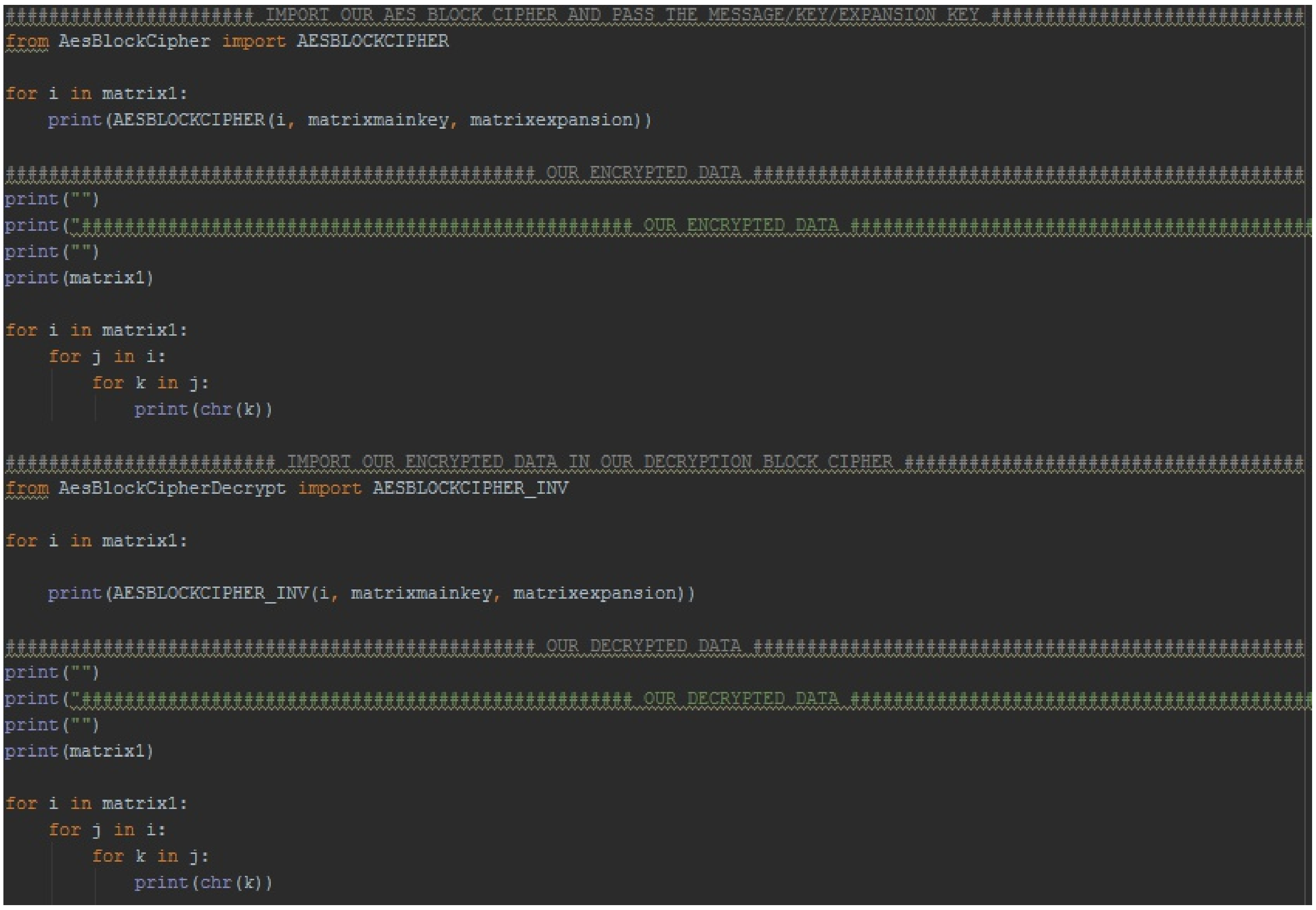


```
############################################### KEY AND EXPANSION KEY ####################################################
# Create the Random Key
matrixmainkey = []
matrixexpansion = []
def KeyExpansion():
    print("")
    print(
        "-------------------------------------RANDOM KEY AND KEY EXPANSION-----------------------------------------

    min_char = 16
    max_char = 16
    i = 0
    char = string.ascii_letters + string.punctuation + string.digits
    password = "".join(choice(char) for x in range(randint(min_char, max_char)))

    print("Random 128 bits Key :")
    print(password)

    # Put the main Key into a matrix

    l = 0
    for i in range(4):
        matrixmainkey.append([])
        for j in range(4):
            matrixmainkey[i].append(ord(password[l]))
            l += 1

    # Shift the last column
    g = matrixmainkey[3][0]
    matrixmainkey[3][0] = matrixmainkey[3][1]
    matrixmainkey[3][1] = matrixmainkey[3][2]
    matrixmainkey[3][2] = matrixmainkey[3][3]
    matrixmainkey[3][3] = g
```



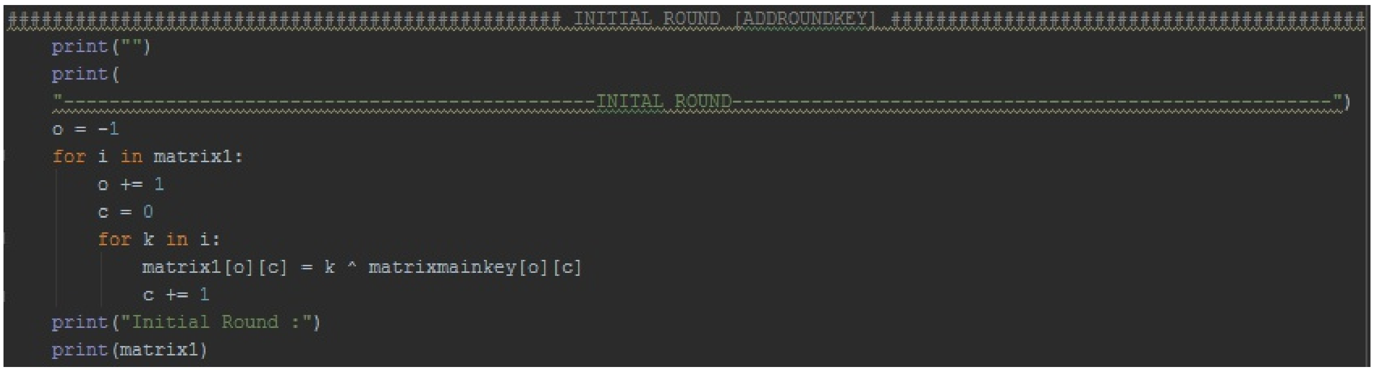
```
    # Create the expansion Key slot

    for i in range(0, 10):
        matrixexpansion.append([])
        for j in range(4):
            matrixexpansion[i].append([])
            for k in range(4):
                matrixexpansion[i][j].append(0)

    # Give the first new column of the Expansion Key
    n = 0
    for i in matrixmainkey[3]:
        j = Rcon[Sbox[i]]
        matrixexpansion[0][0][n] = j ^ matrixmainkey[0][n]
        n += 1

    # Expend the Key for 10 Rounds
    n = 1
    h = 1
    first = 0
    for i in matrixexpansion:

        if first == 0:
            A = i[0][0] ^ matrixmainkey[n][0]
            B = i[0][1] ^ matrixmainkey[n][1]
            C = i[0][2] ^ matrixmainkey[n][2]
            D = i[0][3] ^ matrixmainkey[n][3]

            matrixexpansion[0][n][0] = A
            matrixexpansion[0][n][1] = B
            matrixexpansion[0][n][2] = C
            matrixexpansion[0][n][3] = D

            n += 1

            A = i[1][0] ^ matrixmainkey[n][0]
            B = i[1][1] ^ matrixmainkey[n][1]
            C = i[1][2] ^ matrixmainkey[n][2]
            D = i[1][3] ^ matrixmainkey[n][3]
```

Figure 19 : Implementation of the Random 128 Bits Key + expansion of the Random 128 Bits Key



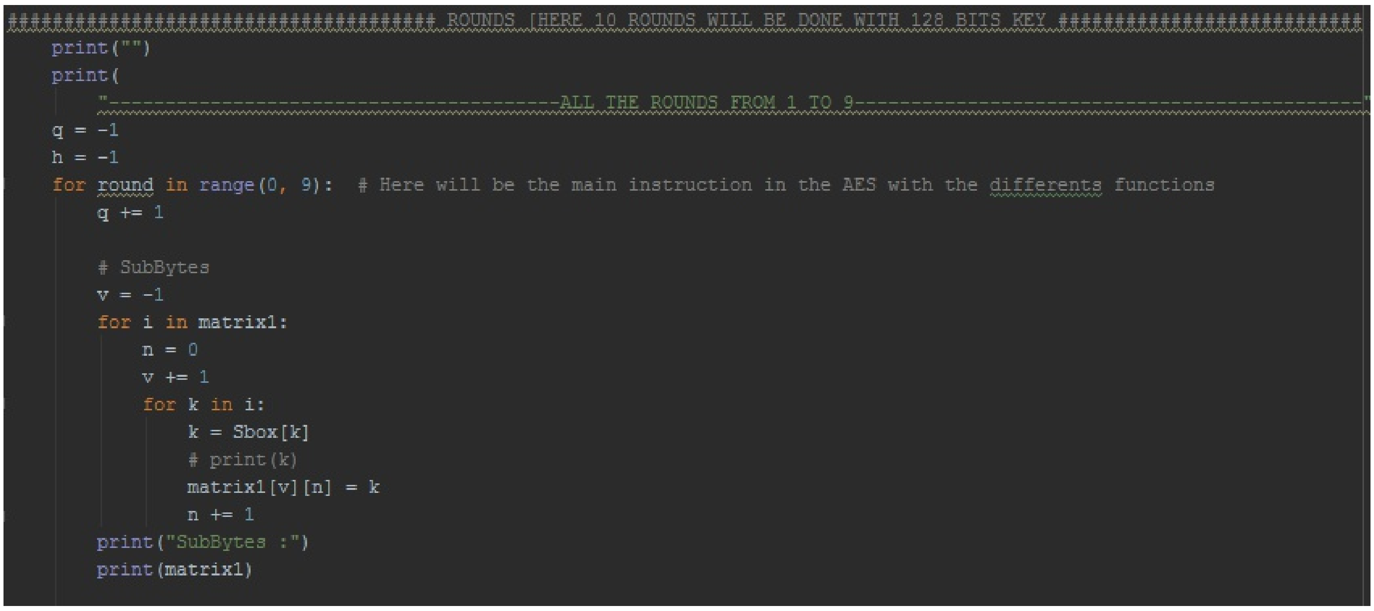
```python
####################### IMPORT OUR AES BLOCK CIPHER AND PASS THE MESSAGE/KEY/EXPANSION KEY #############################
from AesBlockCipher import AESBLOCKCIPHER

for i in matrix1:
    print(AESBLOCKCIPHER(i, matrixmainkey, matrixexpansion))

################################################# OUR ENCRYPTED DATA ###################################################
print("")
print("################################################### OUR ENCRYPTED DATA ###########################################
print("")
print(matrix1)

for i in matrix1:
    for j in i:
        for k in j:
            print(chr(k))

######################### IMPORT OUR ENCRYPTED DATA IN OUR DECRYPTION BLOCK CIPHER #####################################
from AesBlockCipherDecrypt import AESBLOCKCIPHER_INV

for i in matrix1:

    print(AESBLOCKCIPHER_INV(i, matrixmainkey, matrixexpansion))

################################################# OUR DECRYPTED DATA ###################################################
print("")
print("################################################### OUR DECRYPTED DATA ###########################################
print("")
print(matrix1)

for i in matrix1:
    for j in i:
        for k in j:
            print(chr(k))
```

Figure 20 : Implementation of the function AesBlockCipherEncrypt and AesBlock CipherDecrypt which the data will pass in.

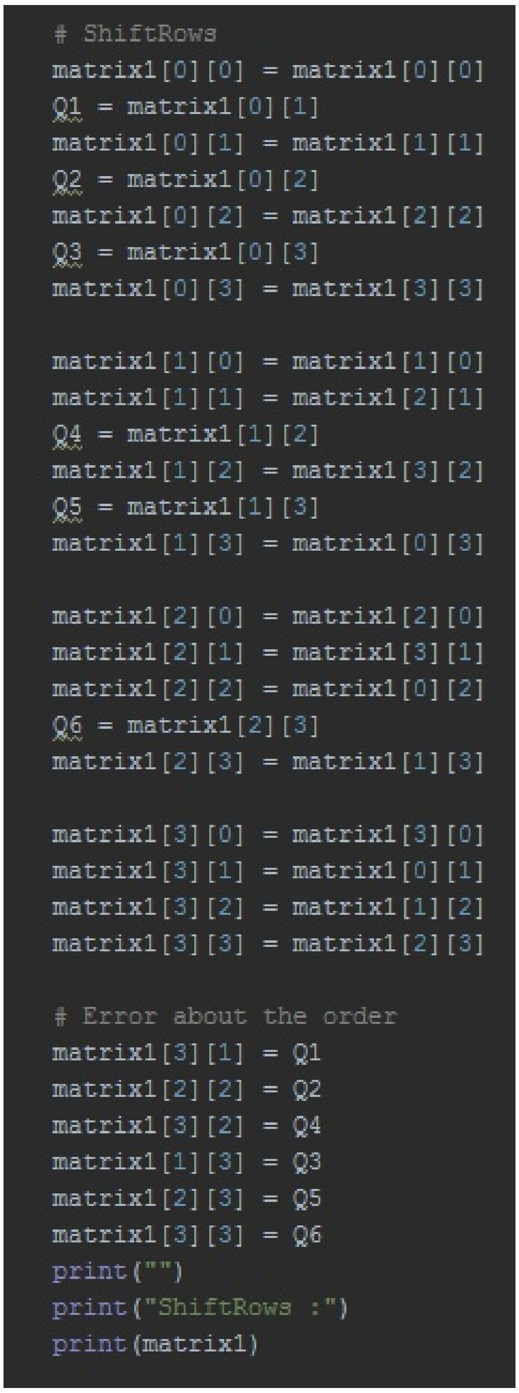


```python
################################################## INITIAL ROUND [ADDROUNDKEY] ###########################################
    print("")
    print(
    "------------------------------------------------INITAL ROUND------------------------------------------------------")
    o = -1
    for i in matrix1:
        o += 1
        c = 0
        for k in i:
            matrix1[o][c] = k ^ matrixmainkey[o][c]
            c += 1
    print("Initial Round :")
    print(matrix1)
```

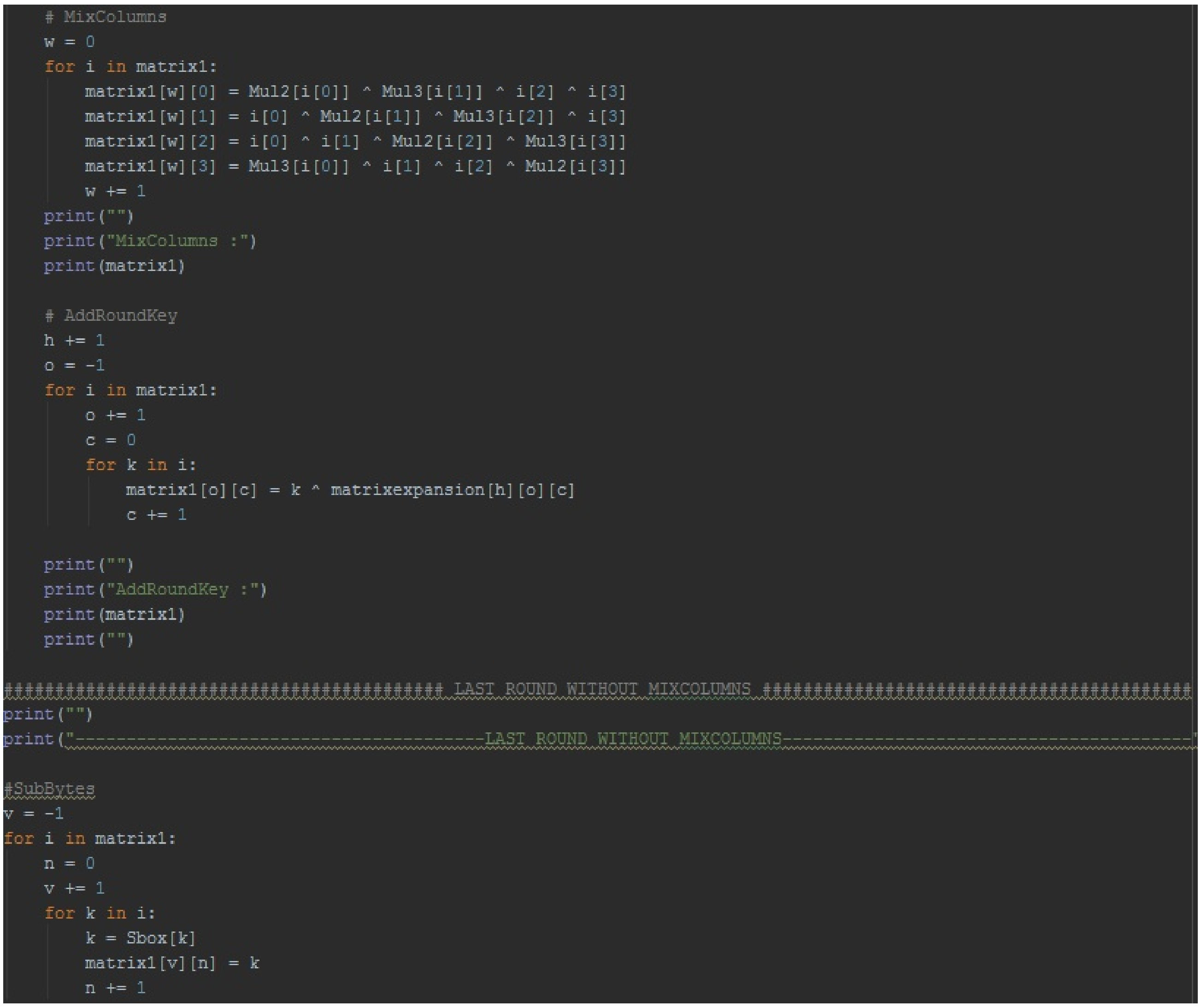
Figure 21 [Encryption] : Implementation of the function Inital Round[AddRoundKey]



```
######################################### ROUNDS [HERE 10 ROUNDS WILL BE DONE WITH 128 BITS KEY ##############################
    print("")
    print(
        "----------------------------------------ALL THE ROUNDS FROM 1 TO 9----------------------------------------------"
    q = -1
    h = -1
    for round in range(0, 9):  # Here will be the main instruction in the AES with the differents functions
        q += 1

        # SubBytes
        v = -1
        for i in matrix1:
            n = 0
            v += 1
            for k in i:
                k = Sbox[k]
                # print(k)
                matrix1[v][n] = k
                n += 1
        print("SubBytes :")
        print(matrix1)
```

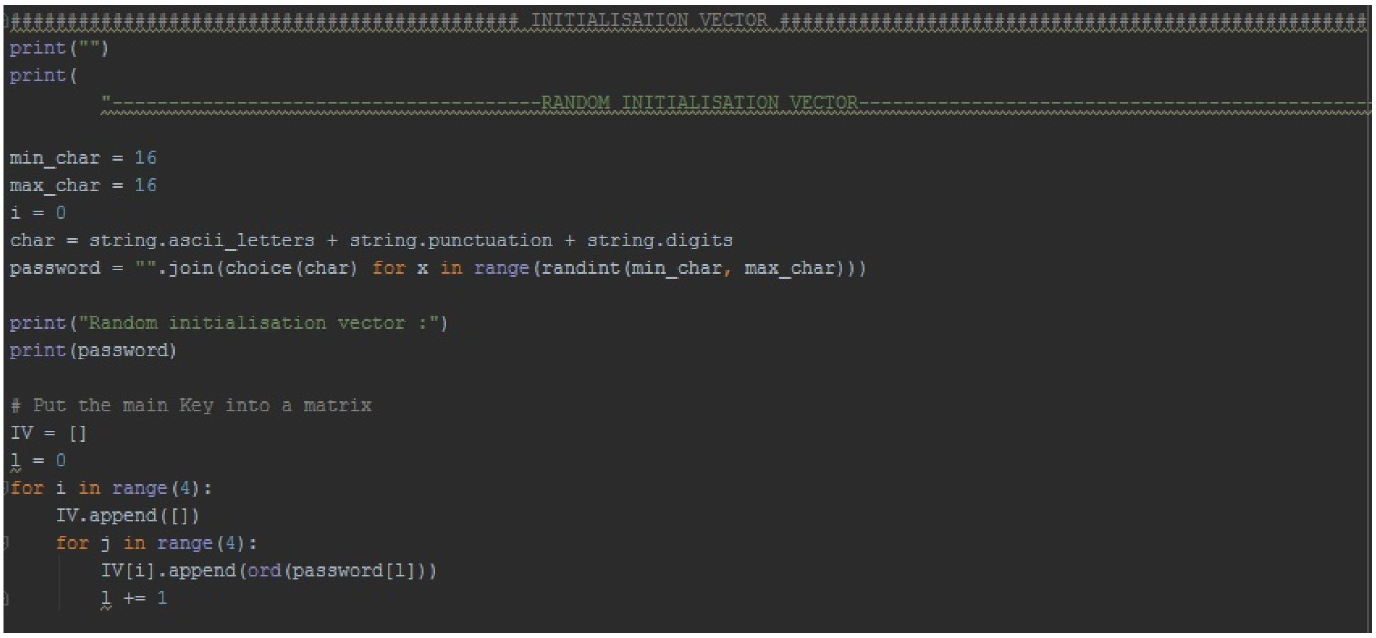


```
# ShiftRows
matrix1[0][0] = matrix1[0][0]
Q1 = matrix1[0][1]
matrix1[0][1] = matrix1[1][1]
Q2 = matrix1[0][2]
matrix1[0][2] = matrix1[2][2]
Q3 = matrix1[0][3]
matrix1[0][3] = matrix1[3][3]

matrix1[1][0] = matrix1[1][0]
matrix1[1][1] = matrix1[2][1]
Q4 = matrix1[1][2]
matrix1[1][2] = matrix1[3][2]
Q5 = matrix1[1][3]
matrix1[1][3] = matrix1[0][3]

matrix1[2][0] = matrix1[2][0]
matrix1[2][1] = matrix1[3][1]
matrix1[2][2] = matrix1[0][2]
Q6 = matrix1[2][3]
matrix1[2][3] = matrix1[1][3]

matrix1[3][0] = matrix1[3][0]
matrix1[3][1] = matrix1[0][1]
matrix1[3][2] = matrix1[1][2]
matrix1[3][3] = matrix1[2][3]

# Error about the order
matrix1[3][1] = Q1
matrix1[2][2] = Q2
matrix1[3][2] = Q4
matrix1[1][3] = Q3
matrix1[2][3] = Q5
matrix1[3][3] = Q6
print("")
print("ShiftRows :")
print(matrix1)
```



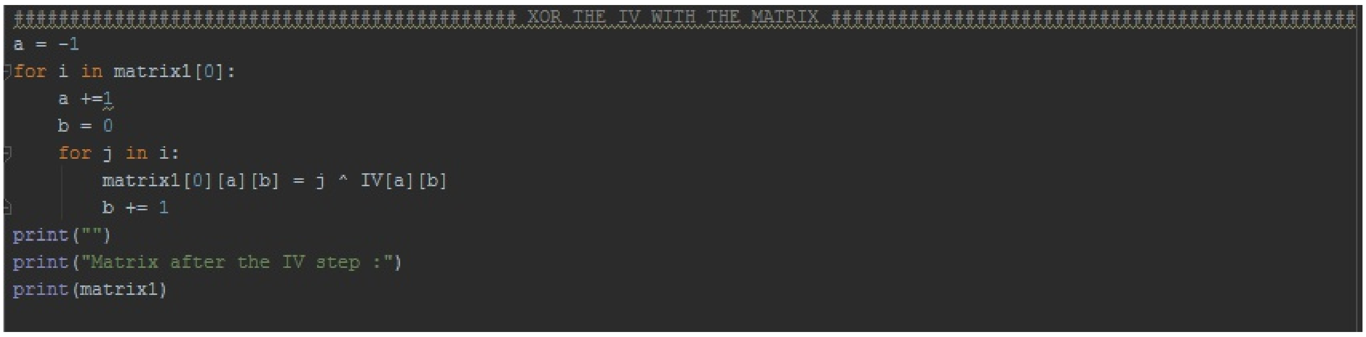
```
    # MixColumns
    w = 0
    for i in matrix1:
        matrix1[w][0] = Mul2[i[0]] ^ Mul3[i[1]] ^ i[2] ^ i[3]
        matrix1[w][1] = i[0] ^ Mul2[i[1]] ^ Mul3[i[2]] ^ i[3]
        matrix1[w][2] = i[0] ^ i[1] ^ Mul2[i[2]] ^ Mul3[i[3]]
        matrix1[w][3] = Mul3[i[0]] ^ i[1] ^ i[2] ^ Mul2[i[3]]
        w += 1
    print("")
    print("MixColumns :")
    print(matrix1)

    # AddRoundKey
    h += 1
    o = -1
    for i in matrix1:
        o += 1
        c = 0
        for k in i:
            matrix1[o][c] = k ^ matrixexpansion[h][o][c]
            c += 1

    print("")
    print("AddRoundKey :")
    print(matrix1)
    print("")

########################################### LAST ROUND WITHOUT MIXCOLUMNS #########################################
print("")
print("----------------------------------------LAST ROUND WITHOUT MIXCOLUMNS----------------------------------------"

#SubBytes
v = -1
for i in matrix1:
    n = 0
    v += 1
    for k in i:
        k = Sbox[k]
        matrix1[v][n] = k
        n += 1
```

Figure 22 [Encryption] : Implementation of all the functions for the Rounds [SubBytes/ShiftRows/MixColumns/AddRounsKey] + Implementation of the Last Round [SubBytes/ShiftRows/AddRoundkey]

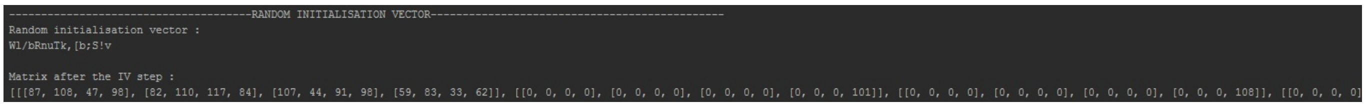


```
############################################### INITIALISATION VECTOR ########################################################
print("")
print(
        "------------------------------------RANDOM INITIALISATION VECTOR-----------------------------------------------

min_char = 16
max_char = 16
i = 0
char = string.ascii_letters + string.punctuation + string.digits
password = "".join(choice(char) for x in range(randint(min_char, max_char)))

print("Random initialisation vector :")
print(password)

# Put the main Key into a matrix
IV = []
l = 0
for i in range(4):
    IV.append([])
    for j in range(4):
        IV[i].append(ord(password[l]))
        l += 1
```

Figure 23 : Implementation of the Initialisation Vector for the CBC mode.

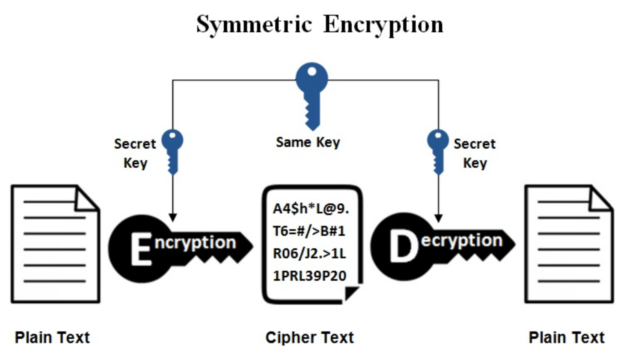


```
############################################### XOR THE IV WITH THE MATRIX ###############################################
a = -1
for i in matrix1[0]:
    a +=1
    b = 0
    for j in i:
        matrix1[0][a][b] = j ^ IV[a][b]
        b += 1
print("")
print("Matrix after the IV step :")
print(matrix1)
```



```
------------------------------------RANDOM INITIALISATION VECTOR----------------------------------------------
Random initialisation vector :
Wl/bRnuTk,[b;S!v

Matrix after the IV step :
[[[87, 108, 47, 98], [82, 110, 117, 84], [107, 44, 91, 98], [59, 83, 33, 62]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 101]], [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 108]], [[0, 0, 0, 0]
```

Figure 24 : Implementation of the XOR between the Initialisation Vector and the first Block Cipher of the data + Result obtained



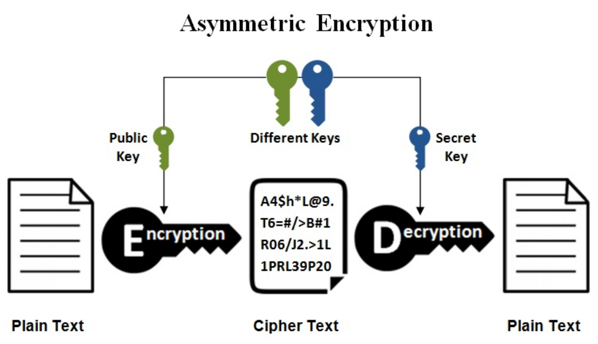


Figure 25 : Representation of the two modes of Encryption [Symmetric and Asymmetric]
Source : SSL2BUY.com, Symmetric vs. Asymmetric Encryption – What are differences?