Improving security of the Advanced Encryption Standard

Mark M. Lehet
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
lehet005@morris.umn.edu

ABSTRACT

In this day an age, information security is more important than ever due to the fact that everyone who has private information wants it to be secure. A useful method of making sure your information is secure is by encrypting it. Encryption is a privacy-protecting technology that essentially transforms your data so it cannot be read by an unintended recipient. A commonly used encryption algorithm is the Advanced Encryption Standard (AES), but this algorithm has security flaws that are not up to satisfactory security. This paper will provide a brief overview of the AES algorithm. We will then talk about some modifications to the AES done by researchers. The modifications aim to remove a flaw with encrypting multimedia and timing attacks. With each modification, they strengthen the security of the AES by removing the security flaws.

Keywords

1. INTRODUCTION

Today a lot of people use the internet to send and receive messages or store crucial information on their computer, and some of that information is deemed to be sensitive to the person or persons involved with the data. When important data is involved, the use of encryption is implemented to hide the information being sent or stored to keep the information safe from attacks. By using encryption, it makes it very difficult for the attacker to gain any information from an encrypted message. There are many different methods in which you can encrypt data, but the one we will be focusing on is the Advanced Encryption Standard (AES), a commonly used encryption algorithm. However, with the use of multimedia becoming a common place, there is a problem that occurs with the AES and multimedia, more specifically images, using what is known as ECB mode of operation. As you can see in Figure 1, this image has been encrypted by AES and there is an issue once the encrypted image has been completed, which is that the image still has noticeable edges that give away what the image was supposed to be hiding. The image pixels are still correlated between their adjacent pixels allowing for this intelligibility to occur. This way of encrypting is not practicable for real life use due to this flaw the poses a potential threat of people gaining access to an encrypted image and still being able to tell what the image is.

Another flaw that the AES faces is that it is inadequate at preventing what is know as timing attacks. The AES performs its sub processes with inconsistent times, creating an opening for attackers to potentially use the information of the time differences to help decrypt an encrypted message.

In Section 2 we discuss the background information of the AES algorithm, Section 3 we will discuss Mondal and Maitra's modificaiton to the AES, and Section 4 we will discuss the Fine Tuned AES. These modifications that are done to the AES each address a unique flaw in the security of the AES, and propose their modification as a solution to the issue. They both modify the AES by adding an additional cipher to strengthen the security.

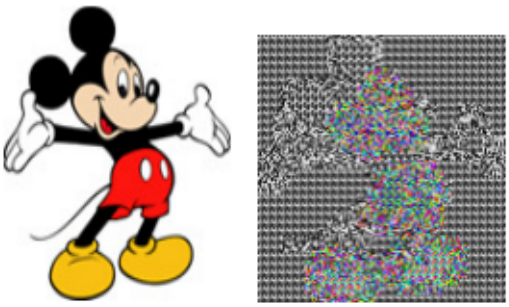

Figure 1: AES encryption with no modifications.

2. BACKGROUND

2.1 Advanced Encryption Standard (AES)

AES was formally known as Rijndael before it was established as a standard by the U.S. National Institute of Standards of Technology. It is an algorithm that is intended to encrypt data, which manipulates the data so the the encrypted message does not give any information about what was actually in the original message. The AES succeeds the

UMM CSci Senior Seminar Conference, December 2015 Morris, MN.

former encryption standard, the Data Encryption Standard (DES), due to the fact that DES was becoming inefficient in security. DES became insecure because computers and computing power became faster and the encrypted data by DES was easily able to be decrypted with brute force.

The AES is implemented as a symmetric key block cipher. A symmetric key in an encrypting algorithm is a parameter that is used to determine and manipulate the data during the encryption process, and is also used to decrypt the encrypted message. A symmetric block cipher is a fixed length string of bits that the algorithm operates on[6].

AES uses a block size of 128 bits and a key size of 128, 192, or 256 bits, which are referred to as AES-128, AES-192, and AES-256.[3] The block is put into a 4X4 array matrix which is referred to as the "state". The key sizes used in this cipher correspond to a certain amount of repetitions, or rounds, of transformations that will occur, that convert the input, which is called plaintext, into the encrypted output, the ciphertext. The rounds for each key size are as follows; 10 rounds for 128 bit keys, 12 rounds for 192 bit keys, and 14 rounds for 256 bit keys. Each round will perform multiple processes to transform the plaintext.

If the data that is going to be encrypted is less than the 128 bits, AES will pad it to make it equal to 128 bits. If the data is larger than 128 bits, it will pad the data to make it multiple of 128 bits, then perform a mode of operation. There are multiple modes of operations, and using the electronic codebook (ECB) mode of operation will result in the encrypted image with an outline (Figure 1) of the original image. This mode of operation is very fast compared to others, but does bring this flaw. The reason this flaw occurs is that it encrypts each block separately. If the blocks are identical, the resulting encrypted blocks will be identical, thus it does not hide any data patters. A more common and more secure mode of operation used with AES is cipher block chaining (CBC), where each plaintext is XORed with the previous encrypted block. It also implements an initialization vector, a fixed-size input, that is XORed with the initial plaintext before encryption and before each following block that is going to be encrypted. This method ensures each block is distinct, but does require more computing. Overall, a mode of operation generally tries to divide the data into multiple blocks of 128 bit states, and attempts to combine them once they are encrypted or as they are getting encrypted.

For the encryption process, AES performs two preliminary steps that are known as KeyExpansion and the InitialRound. The KeyExpansion uses the Rijndael key schedule, which in short terms, expands the key into a number of separate subkeys, that have the same size as the state, to be used in later rounds. It generates a certain amount of sub-keys depending on the key length being used. For a 128 bit key it generates 11 sub-keys, 192 generates 13, and 256 generates 15. Each key will be used throughout the encryption process, one will be used during the initial round and the rest in the rounds portion.

The InitialRound performs a step known as AddRoundKey. In this step, a subkey that was derived from the KeyExpansion step is combined with the state, combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

1. Once these two steps have been performed, we can start with our rounds. The first step in the rounds is the SubBytes operation. This operation takes each byte in the state matrix and replaces it with another according to a look-up table known as the S-Box. The S-Box is derived from the multiplicative inverse over Galois field 2^8, $GF(2^8)$, which is known to have non linearity properties. The S-Box is constructed by combining the $GF(2^8)$ with an invertible affine transformation, which helps prevent attacks that are done using basic algebraic properties. Looking at Figure 2, we can see that a byte from the original state is replace with an entry in the fixed S-Box table, denoting S-Box with S.[7]

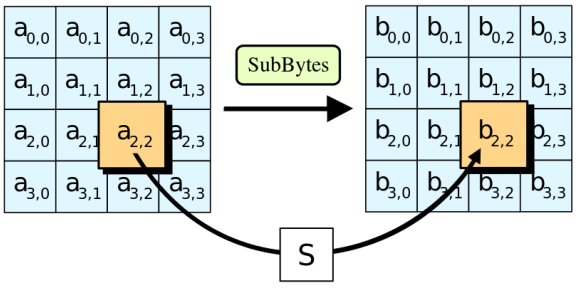

Figure 2: AES SubBytes step. S = S-box.

2. The second step is the ShiftRows step, which operates on the rows of the state. Each row byte is shifted by a given offset, and for AES 128 and 192 bit block sizes, the offsets are as follows; first row is left unchanged, the second row is cyclically left shifted by one, the third row is cyclically left shifted by two, and the fourth is cyclically left shifted by three. You can see the transformation for these block sizes in Figure 3. For 256 bit block size, the first is left unchanged like before, but the second, third, and fourth are cyclically left shifted 1, 3, and 4 bytes respectively.

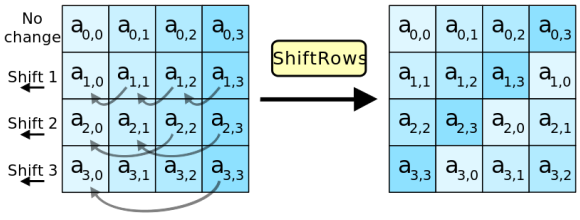

Figure 3: AES ShiftRows step.

3. The third step is the MixColumns step, where four bytes from the columns of the state are combined using an invertible linear transformation. The output will contain four bytes, where each output byte is affected from the input bytes. The columns are transformed using a fixed matrix

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Each column of the state is treated as a polynomial over $GF(2^8)$, and multiplied by a fixed polynomial c(x)

modulo $x^4 + 1$ given by

$$c(x) = \{03\}\, x^3 + \{01\}\, x^2 + \{01\}\, x + \{02\}$$

. Looking at Figure 4, each of the columns are multiplied by the fixed polynomial c(x). The coefficients are displayed in their hexadecimal equivalent of the binary representation.

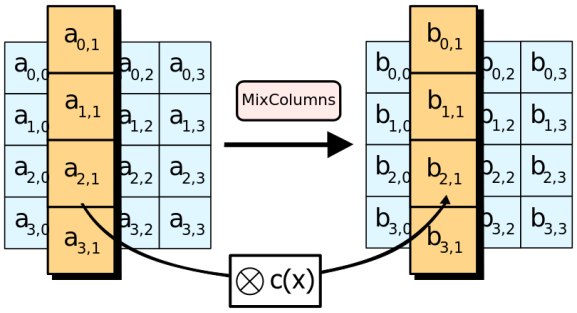

Figure 4: AES MixColumns Step.

4. The last step of a round is the AddRoundKey, looking at Figure 5, we see that each byte of the state is combined with a byte for the subkey for that round using bitwise XOR, denoted as âŁŢ.

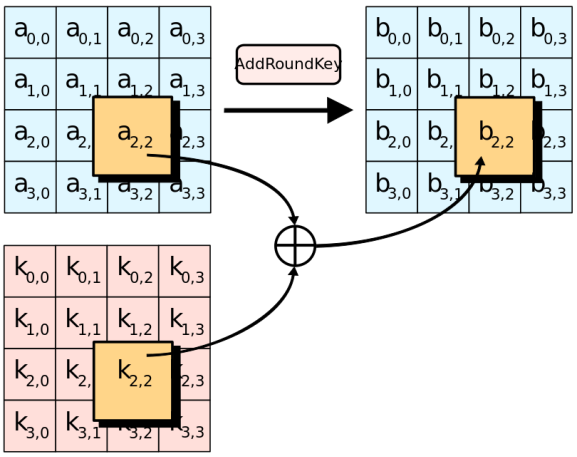

Figure 5: AES AddRoundKey Step.

There is one final step once the rounds are over, this step is known as the Final Round, in which it performs the SubBytes, ShiftRows, and AddRoundKey steps, but excludes the MixColumns step.

Now that we have an encrypted message, we need to be able to decrypt that message. The idea behind the decryption is pretty similar to the encryption process, except each step is done in reverse. The order in which each round is performed is reversed and the steps are done in the inverse. The steps that change are:

1. Inverse AddRoundKey uses the keys generated from the KeyExpansion, but reverses the order they are used.

2. Inverse ShiftRows cyclically rotates each byte to the right instead of left. The offsets are the same that are used in the encryption process.

3. Inverse SubBytes replaces each byte of the matrix with an inverse S-Box.

4. Inverse MixColumns each column is combined as before in the encryption process, but the columns are transformed using a different fixed matrix

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

and uses the fixed inverse polynomial c(x) modulo $x^4 + 1$ given by

$$c(x)^{-1} = \{11\}\, x^3 + \{13\}\, x^2 + \{09\}\, x + \{14\}$$

3. MONDAL AND MAITRA'S AES MODIFICATION

The first modification we are going to discuss was developed by Subijit Mondal and Subhashis Maitra. This modification is used as a purposeful way to remove the intelligible pixel formation of multimedia that the un-modified AES creates after encryption. The reason for the image issue is caused by the mode of operation used, ECB. They decide to use this mode of operation due to the fact that it is quicker than any other commonly used mode of operation. The modification to the AES described by Mondal and Maitra is proposed to be a more efficient and secure by randomizing the key values and by shifting the pixel values. The shifting of the pixel values is crucial to fixing the issue caused with the ECB mode of operation.

This modification is focused on the encryption and decryption of multimedia data such as images, videos, etc.. The modification is performed with two different algorithms, the sender side algorithm (encryption) and the receiver side algorithm (decryption). You can see the intelligible figure created by the AES in Figure 1, where a is the input image for encryption and b is the result after, and that the pixels still have some sort of correlation to each other. The main purpose of the modification is to break that correlation, that AES cannot do on its own, so the encrypted image no longer has any pixel correlation. [2]

3.1 Sender Side Algorithm

For the example used during this process, we will use a key length size that is 128 bits.

1. The sender side algorithm takes in an image file as an input, and as a first step procedure, it right shifts the pixels for the rows and columns a certain offset to break any correlation between the adjacent pixels, which creates a first level cipher. The first level cipher is still just the image, but with the pixels mixed up to the point where a pixel is not correlated to any of its original surrounding pixels.

2. We now will need to generate key values, which is done depending on random mouse positions on the user's screen. Using these mouse positions provides randomness for the keys. During the key generation, we will need to generate 16 byte values from each mouse position. The process takes 8 mouse positions, using the x and y coordinate pixel positions as values, and these

are used to create our key. Once the key value is generated, a Key Expansion routine takes place to generate the key schedule. The Key Expansion will then use the key created using the mouse positions to generate 11 128 bits sub-key arrays, and the first sub-key will be the initial key. The subsequent 10 sub-key arrays are used in the 10 rounds.

3. The next step is the AddRoundKey, where the sub-key is combined with the state. The sub-key is added by combing each byte of the state with the corresponding byte of the sub-key using bitwise XOR.

4. The process now moves to the SubByte step, where each of the 16 bytes in the state are substituted from the fixed table, S-Box.

5. The next step of the process is the Shiftrows process. This will take the state and left shift each row a certain position amount. The first is left unchanged, the second is shifted once, the third is shifted twice, and the fourth is shifted three times, which is the same as the original AES standard.

6. Now the MixColumns step will take the four bytes from each column and combine them using an invertible linear transform. The process from where the AddRoundKey begins repeats this whole process 9 more times to total 10 rounds.

7. Because we generate the key randomly with mouse positions, this will mean will we need to send this key with the cipher image. The process for doing this is by converting the key and cipher values from decimal numbers to binary numbers, then the key values are placed according to the prime numbers of the cipher values and the pixel value of that position of cipher image is shifted one position to the right.

Now we have our encrypted image which will be ready to be sent for decryption intended recipient.

3.2 Receiver Side Algorithm

1. Once a cipher image is received for decryption, the keys will be extracted along with the first level cipher. We start with the key expansion, creating the key schedule. It then creates us the 11 sub-key arrays needed, similar to before with the encryption process. In the next upcoming steps, the processes are similar to the AES's decryption process, and in that process each step is reversely performed.

2. The next step is the AddRoundKey Inverse, where the sub-key is added to the state, but in reverse. So instead of what would be the 'initial' sub-key, we would use what was used in the last round from encryption and use the 'initial' sub-key for the last round.

3. Next will by the SubByte Inverse step, where we will inverse the substitution from the S-Box.

4. Next is the Shiftrows Inverse step, where instead of shifting rows to the left, we will shift them to the right.

5. Now, like before, we will perform the MixColumns Inverse step.

6. After doing this for 9 more rounds, you will need to apply the first level cipher, but left shifting the pixels for the rows and columns. This will leave us with the original image.

3.3 Results Summary

This approach and experiments tested by the creators have shown to be reliable and proven to be more efficient with the application on images. A histogram analysis of the images before and after were conducted to see how the pixels in an image are distributed by plotting the number of pixels at each color intensity level. This is done to prove that there is no similarity between the original image and the encrypted image.

In Figure 6, we can see that the histogram of the original image (a) appears to have high spike intensities around several areas of the graph, implying that there are common color qualities, where as the histogram of the encrypted image (d) shows a more uniform intensity. This result shows that original image is significantly different from the encrypted image. These results also show that the encrypted image itself shows no clues for a statistical attack on the image.

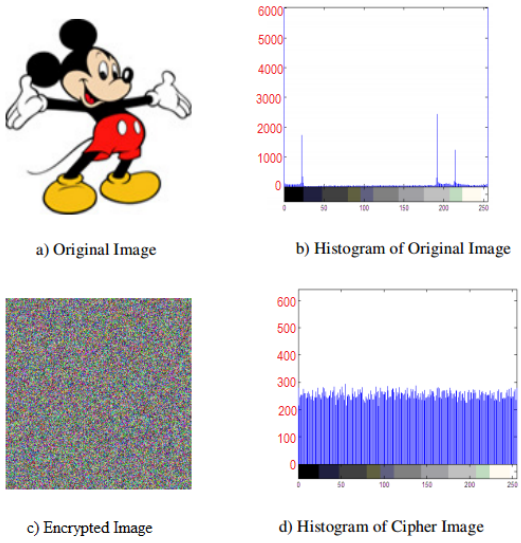

Figure 6: Histogram analysis of the Original Image and Encrypted Image.

This encryption process also bring along higher security due to its randomness in the key generating process. It randomly assigns key values depending on the mouse position on the screen, which allows randomness each time an image is encrypted and gives it a more reliable, secure, and practical use.

4. FINE TUNED AES

In this modification of the AES, the researchers Behnam et al. address the vulnerabilities that AES has against timing attacks, a subcategory of side channel attacks, and strengthen the security of the AES. The modification includes, in each round of the AES, a playfair cipher, that they have modified. It also changes the Final Round to include a MixColumn

step which helps prevent timing attacks. The researchers describe the modification to ensure the enhancement in safety and security of the AES.[5]

4.1 Playfair Cipher

A modification to the AES that they use is a modified playfair cipher, as mentioned before. A playfair cipher is a 5X5 matrix of letters constructed based on a keyword given by a user.[8] This matrix is created by adding the key to the matrix, omitting any duplicate letters. The rest of the matrix is then filled with the rest of the letters of the alphabet (the application of this usually have the letter "Q" omitted or "I" and "J" share the same space, to fit the entire alphabet).

E	X	A	M	P
L	B	C	D	F
G	H	I	K	N
O	Q	R	S	T
U	V	W	Y	Z

Figure 7: Example of playfair cipher.

For example, if we look at Figure 7, the key that was used to create this playfair cipher was "EXAMPLE". The word is placed in the matrix first, omitting the last "E" because it was already used in the key. The following letters are the rest of the alphabet, in order, omitting letters that have already been used in the key. To encrypt a message with this, you would take the message and break it up into groups of two letters, for example, "encryption" would be broken up as "EN CR YP TI ON". Now the message is broken up, each letter pair has rules that apply to encrypt the message on the matrix:

1. If both letters are the same, add an "X" (or a letter that is not used commonly part of repeated letter pairs) after the first letter, this will change the broken up message. For example, if the message is "hello", it would broken up as HE LX LO.

2. If the letters are found on the same row of the matrix, you will cyclically replace the letters to their immediate right in the cipher matrix.

3. If the letters are found on the same column of the matrix, you will cyclically replace the letters immediately below in the cipher matrix.

4. If the letters are found on a different row or column, you would replace the letters with a letter from the same row but at the other letter pairs column in the cipher matrix.

With these rules, we can create our encrypted message. Using the word "encryption", as before, for our message to encrypt. Looking at the broken up letter pairs, we see that they do not contain two of the same letters in one of the letter pairs, so we do not need to apply rule 1. Now we continue to encrypting the letter pairs, and the first letter pair we encrypt is "EN", and looking at Figure 7, we find that "E" and "N" fall on a different row and column. This means we will use rule 4, and the result of this will be "PG". The next letter pair would be "CR", and looking at Figure 7, we find that "C" and "R" fall on the same column, which means we will use rule 3, and the result of this will be "IW". Continuing using the rules 2, 3, and 4 and the playfair cipher matrix, the full encrypted message results in "PG IW ZM RN TG".

The modified playfair cipher used in the AES modification is slightly different as it is a 16X16 matrix instead of the 5X5. It uses a 16X16 matrix for the playfair cipher so that it can fit 256 (0-255) ASCII character codes. Each letter uses the ASCII character code in the cipher. The modified playfair cipher ensures that the key used is not duplicated in the encrypted matrix. It ensures this by making sure that the key used is not repeated with other ASCII characters that would coincide with the given characters.

4.2 Timing Attacks

A timing attack is a subcategory under a larger category of attacks know as side channel attacks, which are done to compromise a cryptography algorithm. A side channel attack is an attack that gains information from the physical implementation of a cryptography algorithm. In terms of a timing attack, the information that is being gained from the physical implementation is the time it takes for each logical operation that is being executed on a computer. The time differences between operations give access to an attacker to work backwards to what an input was[1]. In AES, the variables in times it takes to encrypt, specifically look-ups in the known S-Box table, and can be used to narrow down the possible values for a key[4].

4.3 Modification

The Fine Tuned AES addresses the issue of timing attacks that AES has. The issue falls where AES will release cruicial timing information between the rounds section and the final round section. The difference between the two is that there is no MixColumns step in the final round, and that difference allows some crucial timing information to be released that can be used to help decrypt the message. The modification to the AES implements a MixColumn step in the Final Round, this is a minor change, and is not needed in the original AES, but it ensures unified timing to between processes so that the time it takes between processes cannot be easily analyzed. This helps prevent timing attacks against the execution of the AES, by creating it a more consistent time algorithm. It also helps by with the addition of a delay in the faster operations to hid the timing difference.

The modified playfair cipher is implemented every round in the modified AES, starting with the Initial Round. The Key used in the AddRoundKey step is also used for as a key for the modified playfair cipher. The key is first used to generate the 16X16 playfair cipher matrix, then it is encrypted using the state as the "message". The state is treated as a single dimension array, and the array is split up into groups of two elements, which is similar to breaking you message, in the regular playfair cipher, into letter pairs. The encryption also follows the same rules as the normal playfair cipher; the only modification to these rules is that it does not implement the first rule, as it does not need it. The encrypted message from the modified playfair cipher is returned as the 4X4 state, which is used for the next round of AES.

4.4 Results Summary

As mentioned earlier under the modification section, the addition of a MixColumns step on the last round of AES ensures a consistent time for the algorithm. The timing attacks that analyze the time leakages from queries cannot gain as much information/secrets when analyzing these time leakages. The addition of it rectifies gaps, or areas where timing is different, to have a unified timing throughout the encryption process. The addition of this does bring an extra step to the encryption/decryption process, which does penalize the speed, although minor.

The addition of the modified playfair cipher is to increase the security of the encryption and decryption process. The algorithm is now more dependent on the keys it uses, so the encryption and decryption security are strengthened. This additional step, although fast if implemented correctly, does decrease the performance speed of the encryption/decryption process.

5. CONCLUSIONS

As we look at these solutions and modifications done to the AES, we see that they both find a unique issue related to the AES and remove the security flaw by implementing a modification that on slightly changes how the AES works. These modifications both find a unique issue, resolve the issue, and strengthen the security.

The modification by Mondal and Maitra remove the AES flaw of seeing an intelligible outline of an image that is encrypted when implemented with ECB mode of operation. By having this modification, AES can safely use ECB mode of operation, which is one of the fastest mode of operations to use. It also strengthens the security by adding randomness to the algorithm through the input of mouse positions as a way of generating a key.

The fine tuned AES by Behnam et al. seeks to remove the potential of timing attacks done to the encryption process by making the algorithm more consistent in its process timing. It also strengthens the AES by including a modified playfair cipher, which adds an extra step of encryption to the original AES algorithm. A down side to the fine tuned is that both of the modifications done to the AES decrease the performance speed.

It is hard to directly compare these two modifications to find out which one is better because they both fix a unique flaw and strengthen the AES.

Acknowledgments

Thank you to Kristen Lamberty for they great advice and feedback.

6. REFERENCES

[1] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3):17:1–17:29, Jan. 2009.

[2] S. Mondal and S. Maitra. Data security-modified aes algorithm and its applications. *SIGARCH Comput. Archit. News*, 42(2):1–8, Sept. 2014.

[3] N. I. of Standards and Technology. Announcing the advanced encryption standard (aes). November 2001. [Online; accessed 15-September-2016].

[4] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] B. Rahnama, A. Elci, and I. Eweoya. Fine tuning the advanced encryption standard (aes). In *Proceedings of the Fifth International Conference on Security of Information and Networks*, SIN '12, pages 205–209, New York, NY, USA, 2012. ACM.

[6] V. K. Rejani R. Deepu. Study of symmetric key cryptography algorithms. *International Journal of Computer Techniques (IJCT)*, pages 45–50, Mar - April 2015. [Online; accessed 15-September-2016].

[7] Wikipedia. Advanced encryption standard — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 15-September-2016].

[8] Wikipedia. Playfair cipher — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 15-September-2016].