IMPLEMENTATION OF TEXTUAL INFORMATION ENCRYPTION USING 128, 192 AND 256 BITS ADVANCED ENCRYPTION STANDARD ALGORITHM

Abikoye Oluwakemi Christiana [1], Garba Qudus Adeshola [1], Akande Noah Oluwatobi [2]

[1] University of Ilorin - Nigeria, Department of Computer Science
[2] Landmark University - Nigeria, Department of Computer Science

Corresponding author: Akande Noah Oluwatobi, akande.oluwatobi@gmail.com

ABSTRACT: Individual or organizational sensitive data and information need to be prevented from unauthorized access. Several reports of middle man attacks and intrusion in form of hacking have been reported over the years when data are sent over wired or wireless media. One of the ways to prevent these is to either secure the medium through which information is being sent or to put measures in place to secure the actual information being sent. Though implementing both options could be expensive, however, implementing any of the option could still be effective. This could be achieved by making the data being sent meaningless to a third party but meaningful to the intending recipient. Cryptography is a proven way to achieve this. It entails using certain mathematical techniques to make a data unreadable and meaningless to any other person except the actual recipient who has the key needed to decrypt the encrypted data. Rijndael algorithm; which is the Advanced Encryption Standard (AES) for implementing cryptography is presented in this paper. It is a symmetric approach to implementing cryptography as it uses the same key for encryption and decryption. A detailed analysis of its implementation using its various 128, 192 and 256 bits key lengths is presented in this paper. The encryption and decryption process is made more secured by sending the encrypted text to the recipients email while the decryption key is sent to the recipient's phone number. This is done to prevent instances where the recipients email is hacked.

KEYWORDS: *Advanced Encryption Standard, Cryptography, Encryption, Decryption, Rijndael algorithm.*

1. INTRODUCTION

An increase in the use of internet for data and information communication has increased the security concerns of its users. Measures are expected to be put in place to ensure the continuous confidentiality, guarantee integrity and availability of sensitive information such as credit card numbers, patient medical records, bank transactions, students' academic results e.t.c. being transferred over wired or wireless internet medium or being stored. A total security approach will entail securing the data being transferred or stored and securing the medium though which they are being transferred or stored. Though, implementing both approaches will always come with a cost, any attempt at fully implementing any of the two can still guarantee the safety of data being transferred. Cryptography; which attempts to make data being transferred meaningless to a third party but meaningful to the intending benefactors is a leading approach at securing data. This can be achieved in many ways using several algorithms. One of these algorithms is Advanced Encryption Standard (AES) algorithm aptly called Rijndael algorithm.

AES uses a uniquely generated key to encrypt as well as decrypt data; therefore, it is called a symmetric data encryption approach. In AES encryption, the data to be encrypted and the encryption key to be used are of different lengths; this could be 128, 192 or 256 bits ([KR16]). Furthermore, AES employs an iterative encryption process in encrypting a data; this varies depending on the data length being used. The number of iterations called could be 10, 12 and 14 for bits 128, 192 and 256 bits respectively. Also, every AES algorithm implementation carries out byte transformations in four steps which are: substituting the byte using a substitution table (S-box), using different offsets to shift rows of the state array, manipulating the data within each column of the state array and adding a round key to the state array ([PDP12]).

Due to the complexities involved in implementing AES in 192 and 256 bits, most literature had focused on the implementation of AES algorithm using 128 bits. However, this works reports the implementation of AES data encryption algorithm using 128, 192 and 256 bits.

2. LITERATURE REVIEW

Several implementations of AES algorithms have been reported in literature. To start with, a comprehensive comparative analysis of top five cryptographic algorithms such as Data Encryption Standard (DES), Triple DES, AES, Blowfish and

Rivest, Shamir and Adelman (RSA) was carried out by ([P+16]). 25KB, 50KB, 1MB, 2MB, 3MB file sizes were used to measure encryption time, decryption time, memory used, avalanche effect and entropy of the cryptographic algorithms considered. Results obtained revealed that RSA requires the highest encryption time while blowfish requires the least encryption time across all the file sizes considered. However, AES achieved a minimal difference in encryption time compared to other algorithms. Furthermore, all the decryption time of all the algorithms increases with time while that of AES seems to be uniform across the entire file sizes. Blowfish algorithm requires the least memory while RSA consumes the highest memory. However, AES and DES require minimum memory size. This evaluation revealed the strength and weaknesses of the encryption algorithms. This further justifies the reason for the wide implementation and acceptance of AES algorithm. Similarly, a comparative analysis of five different 128 bit key length AES implementations was reported by ([FA17]). How cipher key expansion was carried out and how cipher module was executed was varied in these implementations. Result obtained revealed that when AES encryption designs are carried out using parallelism, higher frequency which ultimately results in improved throughput is achieved. With a view to study the encryption and decryption speed of AES algorithm, Smekal, Frolka & Hajny ([SFH16]) implemented AES algorithm in VHSIC Hardware Description Language (VHDL) and was tested on Field Programmable Gate Array (FPGA) network card. When 987MB, 4934MB and 9868MB file sizes were used, encryption time were 5094Mbit/secs, 5173Mbit/secs and 5028Mbits/secs respectively. This showed that file size has an insignificant effect on AES encryption and decryption speed when tested on FPGA.

An attempt to improve the performance of AES algorithm was reported by Abdulazeez & Tahir, ([AT13]). The SubByte and ShiftRow transformations were merged during the encryption process while inverse SubByte and inverse ShiftRow were also merged for the decryption process. VHDL was used for the coding and FPGA was used for the simulation. For the encryption and decryption process, chip resources used were 9% at operating clock frequencies of 382.988MHz and 382.988MHz respectively. This showed that minimal hardware resources were used.

When AES algorithm is implemented on Central Processing Unit (CPU), it is conjectured that vast amount of time is still consumed especially when large data size is involved. However, Graphics Processing Unit (GPU) is known to possess a massive parallel processing power when compared to CPU. Therefore, AES algorithm was implemented on GPU by Awadallah, Abdelrahman & Fouad ([AAF17]). Its performance was evaluated on Pascal, Kepler and Maxwell GPU architectures with eight different parallel schemes. Result obtained showed that full optimization was achieved with higher input block sizes as the throughput of the proposed method increases as the block size increases. Also, with Maxwell and Pascal GPUs, 128 and 256 thread block size produces the best performance while 512 thread block size achieved the best throughput for Kepler GPU.

In contrary to the widely accepted key size with corresponding key length, Kumar & Rana ([KR16]) introduced 32obits key size and conducted 16 iterations for the encryption and decryption process. However, results obtained revealed that when 15, 42, 90, 157, 303, 412Kb file sizes were experimented with; more time was consumed across all the file sizes when compared to the actual AES algorithm. Result obtained also showed that as the file sizes increase, execution time also increases.

256 AES key length was implemented by Nti, Gymfi & Nyarko ([NGN17]). The implementation was evaluated by using wireshark hacking tool to intercept and retrieve the encrypted messages sent over an unsecured network. The tool successfully obtained the sent encrypted file but scrambled letters which were not meaningful were displayed.

Literature reviewed so far have implemented 128bits AES key lengths with the expected 10 iterations, however, literature that have implemented 192 and 256 key lengths are hard to come by. As a result of this, this research focuses on the implementation of 128, 192 and 256 AES key lengths for textual data encryption and decryption.

3. RESEARCH METHODOLOGY

The implementation of 128, 192, 256 AES key length is discussed in this session. As required, 10, 12 and 14 rounds were carried out for the 128, 192, 256 AES key lengths respectively. The AES encryption entails the following:

a) Determining the set of round keys from the cipher key: Round Keys are generated from the Cipher key using AES key schedule. It entails turning the key bits into round keys that can be used by the cipher. AES block and key length determines the number of round Keys to be generated. 128, 192 and 256 bits Key length requires 11, 13 and 15 round keys respectively.

b) Generate a new state array by computing an XOR operation between the plaintext (starting state array) and the initial round key.

c) Performing “N” rounds of state manipulation: This is the major encryption task where states are

manipulated. This consists of four (4) different transformations which are SubByte, ShiftRow, Mix Column and AddRoundKey.
d) Copying the final state array out as the encrypted text.

3.1. AES State Transformations

The following AES state transformations will be carried out:

a)Sub Bytes
This is a non-linear substitution process where each byte in the state is replaced with another as specified in the substitution box called s-box. This entails computing the multiplicative inverse of the byte using Galois field inverse operation (2^8) and then applying a fixed affine transformation on the data.

b)Shift Row
This entails computing n-1 left shift operation on the byte alternatingly, where n is the number of row. Row 0 of the byte remain untouched while the first, second and third rows of the byte will be shifted once to the left. With this arrangement, the column of the output byte will contain bytes from each columns of the input state.

c)Mix Columns
This is a column by column linear transformation of the input byte. Each column of the byte is considered to be a four-term polynomial over the Galois field and is then multiplied by a fixed polynomial c(x) which is given as $3x^3 + x^2 + x + 2$. Such that:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$
$$(0 \le c \le N_b)$$

where N_b is the block size.
The polynomial is multiplied by a fixed polynomial and the result is taken in modulo $x^4 + 1$. The fixed polynomial is represented as; c(x), where the c(x) is given as $3x^3+1x^2+1x+2$.

d)Add Round Key
For the add round key transformation, bitwise XOR operation is performed between the present state data and the round key. The key selector block is used to choose the round key using the value of the expanded key and the current round.

3.2. The Proposed System Model

The AES encryption and decryption were carried put using the following Algorithm:

3.2.1. AES Encryption Process

Given:
Plain text: I Like This Book
Cipher key: Garba Abdulqudus
The encryption process will entail the following:
a) Generating the round keys
Step 1: converting the plain text to Hex (128bit): 49, 20, 4C, 69, 6B, 65, 20, 54, 68, 69, 73, 20, 46, 6F, 6F, 6B.
Step 2: converting the cipher text to Hex (128bits): 47, 61, 72, 62, 61, 20, 41, 62, 64, 75, 6C, 71, 75, 64, 75, 73.
Step 3: Generating the round key by breaking down the cipher text into words produces:
W_0 = 47, 61, 72, 62 W_2 = 64, 75, 6C, 71
W_1 = 61, 20, 41, 62 W_3 = 75, 64, 75, 73
Then left shift row operation is performed on W_3 to produce a temporary word G (W_3): 64, 75, 73, 75
Step 4: Performing byte substitution of G (W_3) using S-box produces 43, 9D, 8F, 9D
Step 5: Add round constant (Rcon1) to G(W_3) for the first round i.e. 01, 00, 00, 00. This produces new word NG (W_3) : 42, 9D, 8F, 9D.
Step 6: compute XOR of NG (W_3) and W_0 to produce new byte W_4 i.e. 42, 9D, 8F, 9D ⊕ 47, 61, 72, 62. This produces 05, FC, FD, FF
Step 6: compute XOR of W_4 and W_1 to produce new byte W_5 i.e. 05, FC, FD, FF ⊕ 61, 20, 41, 62. This produces 64, DC, BC, 9D
Step 7: compute XOR of W_5 and W_1 to produce new byte W_6 i.e. 64, DC, BC, 9D ⊕ 64, 75, 6C, 71. This produces 00, A9, D0, EC.
Step 8: compute XOR of W_6 and W_1 to produce new byte W_7 i.e. 00, A9, D0, EC ⊕ 75, 64, 75, 73. This produces 75, CD, A5, 9F.
Step 9: merge bytes W_4, W_5, W_6, and W_7 together to produce the round key (RKey1) for first iteration.
RKey1 = 05, FC, FD, FF, 64, DC, BC, 9D, 00, A9, D0, EC, 75, CD, A5, 9F.
This could be converted to a matrix form, such that Rkey1 becomes:

$$\begin{pmatrix} 05 & 64 & 00 & 75 \\ FC & DC & A9 & CD \\ FD & BC & D0 & A5 \\ FF & 9D & EC & 9F \end{pmatrix}$$

Repeating steps 3 to 9 for the 10 rounds required by 128bits key length produces the following round keys:
Rkey2: BA, FA, 26, 62, DE, 26, 9A, FF, DE, 8F, 4A, 13, AB, 42, EF, 8C
Rkey3: 92, 25, 42, 00, 4C, 03, D8, FF, 92, 8C, 92, EC, 39, CE, 7D, 60
Rkey4: 11, DA, 92, 12, 5D, D9, 4A, ED, CF, 55, D8, 01, F6, 9B, A5, 61
RKey5: 15, DC, 7D, 50, 48, 05, 37, BD, 87, 50, EF, BC, 71, CB, 4A, DD
RKey6: 2A, OA, BC, F3, 62, OF, 8B, 4E, E5, 5F, 64, F2, 94, 94, 2E, 2F
RKey7: 48, 3B, A9, D1, 2A, 34, 80, 9F, CF, 6B, E4, 6D, 5B, FF, CA, 42
RKey8: DE, 4F, 85, E8, F4, 7B, 05, 77, 3B, 10, E1, 1A, 60, EF, 2B, 58
RKey9: 1A, BE, EF, 38, EE, C5, EA, 4F, D5, D5, 0B, 55, B5, 3A, 20, 0D
RKey10: AC, 09, 38, CD, 42, CC, D2, 82, 97, 19, D9, D7, 22, 23, F9, DA
b) Compute a new state array by performing XOR operation between the initial cipher key in step 2 and the plain text in step 1.

$$\begin{pmatrix} 49 & 6B & 68 & 42 \\ 20 & 65 & 69 & 6F \\ 4C & 20 & 73 & 6F \\ 69 & 54 & 20 & 6B \end{pmatrix} \oplus \begin{pmatrix} 47 & 61 & 64 & 75 \\ 61 & 20 & 75 & 64 \\ 72 & 41 & 6C & 75 \\ 62 & 62 & 71 & 73 \end{pmatrix} = \begin{pmatrix} 0E & 0A & 0C & 37 \\ 41 & 45 & 1C & 0B \\ 3E & 61 & 1F & 1A \\ 0B & 36 & 51 & 18 \end{pmatrix}$$

Using the new state array, state manipulations in 10 rounds will be computed as required for AES 128key lengths. This will also involve sub byte, shift row, mix column, and add round key process.
ROUND 1
Step 1: using the sub byte of the new state array, substitute its entries with corresponding entries in the AES S-box. This produces:

$$\begin{pmatrix} AB & 67 & 6F & 9A \\ 83 & 6E & 2C & 2B \\ B2 & EF & C0 & A2 \\ 2B & 05 & D1 & AD \end{pmatrix}$$

Step 2: Perform 0, 1, 2, 3 bytes left shift row operation on the sub byte in step1. This produces:

$$\begin{pmatrix} AB & 67 & 6F & 9A \\ 6E & 2C & 2B & 83 \\ C0 & A2 & B2 & EF \\ AD & 2B & 05 & D1 \end{pmatrix}$$

Step 3: perform a mix column matrix by multiplying state array in step 2 by a fixed matrix

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

specified in the AES encryption process. This produces a new state matrix:

$$\begin{pmatrix} 92 & 32 & 14 & 8F \\ 80 & E8 & F1 & 7C \\ B3 & 69 & 36 & 74 \\ 09 & 70 & 32 & A0 \end{pmatrix}$$

Step 4: perform add round key operation by XORing the round key for iteration 1 i.e. RKey1 with the new state matrix produced in step 3. i.e.

$$\begin{pmatrix} 92 & 32 & 14 & 8F \\ 80 & E8 & F1 & 7C \\ B3 & 69 & 36 & 74 \\ 09 & 70 & 32 & A0 \end{pmatrix} \oplus \begin{pmatrix} 05 & 64 & 00 & 75 \\ FC & DC & A9 & CD \\ FD & BC & D0 & A5 \\ FF & 9D & EC & 9F \end{pmatrix}$$

this produces a new state matrix:

$$\begin{pmatrix} 97 & 56 & 14 & FA \\ 7C & 35 & 58 & B1 \\ 4E & D5 & E6 & D1 \\ F6 & ED & DE & 3F \end{pmatrix}$$

Step 5: repeat steps 1-4 to generate the new state matrix for the remaining 9 iterations.
The matrices generated are:
Round 2: EB, B9, A0, 13, 65, 43, FA, E7, C8, 3A, F9, 4B, DF, 57, F3, DF.
Round 3: 72, D6, CF, 6A, 3D, 3F, 88, 5F, C0, 7D, 5D, 47, 90, 3C, 2D, 39.
Round 4: 40, B7, 09, CF, D3, 6A, 3F, A7, C3, 68, CC, 30, 73, FB, 5B, 58
Round 5: 88, CE, BD, 9D, F8, 28, B3, 64, 97, 3F, 28, 78, E5, 96, 8E, 83.
Round 6: FB,75,68,C2,38,F1,6F,5D,7C,21,4D,1D, 6A,4B,6B,C8.
Round 7: 58,9B,F9,4B,62,76,EB,5E,28,85,19,33,EF, 9B, E9,22.
Round 8: 05, A1, CB, 87, B4, 75, CA, 66, AE, E4, 00, 7B, D3, FB, 77, 42.
Round 9: 3F, 7D, AB, 3A, B6, 89, C6, 71, EB, 3D, 8B, 14, 7A, 85, 7F, A3.
Round 10:D9,AE,05,C7,0C,EB,00,02,7E,8E,BB,

74,F8,DC,66,20.

Results obtained at round 10 is the required cipher text for the plain text "I Like This Book".

For the 192 and 256bits, the following algorithm as used in 128 bits was implemented:

i. Convert the plain text to Hex (192 bits or 256 bits depending on the key length)
ii. Convert the cipher text to Hex (192 bits or 256 bits depending on the key length)
iii. Generate round keys by breaking down the converted cipher text into words
iv. Perform left shift row operation on the rightmost word generated in (iii) so that rows 1,2,3,4 are shifted by 0,1,2,3 bytes respectively.
v. Performing byte substitution of the word generated in (iv) using S-box
vi. Add the round constant of the current iteration n to the word generated in (v) to produce a new word
vii. Compute XOR of the new word in (vi) with the remaining word bytes in (iii)
viii. Repeat steps (vi) to (vii) till the last word in (iii) has be XORed
ix. Merge the words generated in (viii) together to produce the round key (RKey1) for the first iteration.
x. Compute a new state array by performing XOR operation between the initial cipher key in (ii) and the plain text in (i).
xi. Using the sub byte of the new state array in (x), substitute its entries with corresponding entries in the AE S S-box.
xii. Compute a left shift of the rows in the modified states produced in (xi) cyclically so that rows 1,2,3,4 are shifted by 0,1,2,3 bytes respectively.
xiii. Perform a mix column matrix by multiplying shifted array by a fixed matrix

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

xiv. Perform add round key operation by XORing the round key iteration n with the new state matrix produced in the previous iteration n-1
xv. Steps (xiii) to (xiv) are repeated 9, 11 more times As required for 192 and 256 key lengths respectively.
xvi. The result of the last iteration is the cipher text of the input plain text

3.2.2. AES Decryption Process

The decryption process is a reverse of the encryption process discussed above. The encrypted messages when decrypted are expected to produce the same text as the ones encrypted. The decryption algorithm for each bit key length uses the same round number, round key and performs all the round transformations similar to the encryption algorithm but in opposite way using inverse function of each transformation (InvSubByte, InvShiftRows, InvMixColumn) by reversing all the steps taken in the encryption. XOR Round key doesn't take an inverse function here because XORing twice gives the original value. InvSubByte works the same way as SubBytes but uses a different table that returns the original value. InvShiftRows involves rotating left instead of right and the InvMixColumn uses a different constant matrix to multiply the columns.

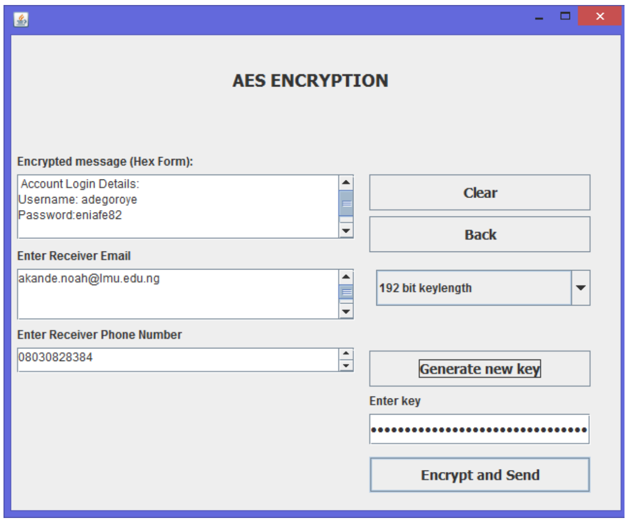
The steps involved in the decryption algorithm are as follows:

i. Examine the cipher text and divide it into a group of 4 bytes.
ii. Process each of the bytes of the cipher text in n rounds where n can be 10, 12, 14 depending on the selected bit key length
iii. The cipher text is XORed with the initial round key "n" to produce a new state
iv. Perform left row shift on the new state generated in (iii) where rows 1,2,3,4 are shifted by 0,1,2,3 bytes respectively.
v. Substitute each single byte in the state matrix produced in (iv) by the inverse value of the S-box.
vi. XOR the result in step (v) with the current round key to generate a new state.
vii. Perform an inverse mix column matrix multiplication on the new state produced in (vi)
viii. Substitute each single byte in the state produced in (vii) with corresponding inverse value in the S-box.
ix. Steps iii to vii are repeated 7, 9, 11 more times depending the key selected using the output of the preceding round.
x. A final round which involves XORing the round key 0 with the last output in (ix).
xi. The output in (x) is the original plain text that was encrypted.

4. RESULTS AND DISCUSSION

The AES 128, 192 and 256 encryption and decryption algorithms discussed in section 3 were implemented using Java programming language on Netbeans Integrated Development Environment (IDE) 7.0. The Netbeans IDE was run on a windows 8.0 operating system with 2.0GHz processor and 2GB RAM.

As illustrated in Figure 1, users are expected to enter a plain text to be encrypted inside the text box then provide receiver's email address and phone number. The preferred encryption key length is expected to be selected after which the generate new key button is expected to be clicked to generate an encryption key that will be sent to the receiver's phone number while the encrypted message will be sent to the receiver's email address.



Account Login Details:
Username: adegoroye
Password:eniafe82

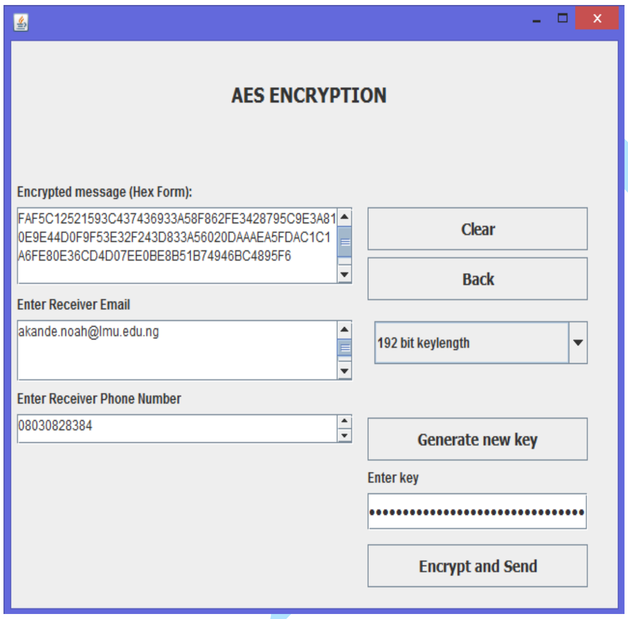
akande.noah@lmu.edu.ng

08030828384

192 bit keylength

Figure 1: Encryption Interface

Figure 2 shows the encrypted message that will be sent to the receiver's email address while the encrpted key will be sent to the receiver's phone number.



FAF5C12521593C437436933A58F862FE3428795C9E3A81
0E9E44D0F9F53E32F243D833A56020DAAAEA5FDAC1C1
A6FE80E36CD4D07EE0BE8B51B74946BC4895F6

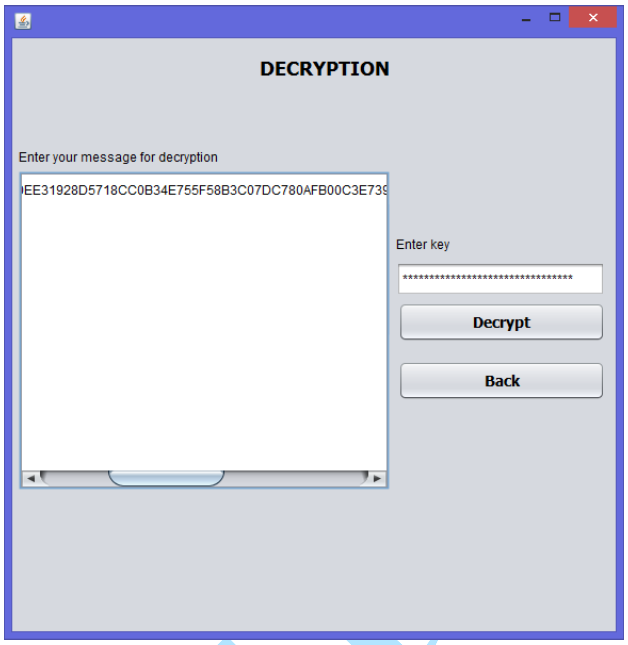
akande.noah@lmu.edu.ng

08030828384

192 bit keylength

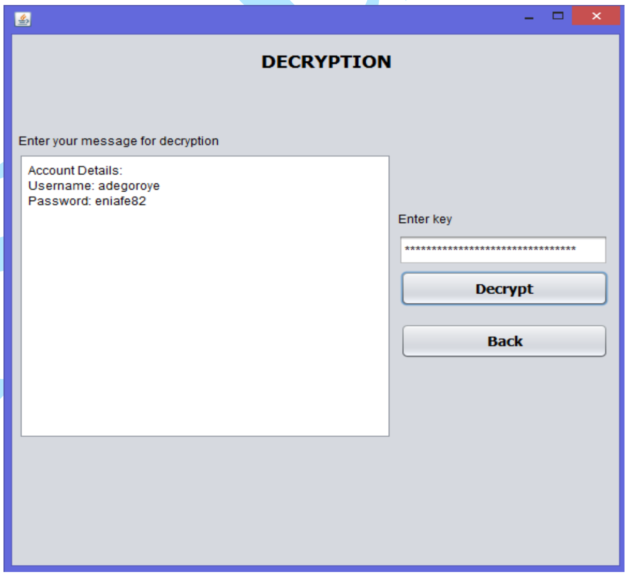
Figure 2: Encryption Interface Showing the Encrypted Message

For the decryption process, the receiver is expected to paste the encrypted message received through the email inside the decryption textbox while the decryption key is expected to be provided as shown in Figure 3 in order to decrypt the encrypted message as shown in Figure 4.



EE31928D5718CC0B34E755F58B3C07DC780AFB00C3E739

Figure 3: Decryption Interface



Account Details:
Username: adegoroye
Password: eniafe82

Figure 4: Decryption Interface Showing the Decrypted Message

With this, middle man atack has been prevented and the message has been secured. Should an intruder gain access to the reciever's email account, the encryption key which doubles as the decryption key that was sent to the reciever's phone will be needed to decrypt the encrypted message. The only way through which this process could be undermined is when an intruder gain access to both the receiver's email and phone. Depending on the interest of the sender, any of the AES key lengths can be used to encrypt the message i.e. the message can be encrypted using 128, 192, 256 AES key lengths.

5. CONCLUSION AND FUTURE WORK

AES encryption process using 128, 192 and 256 key lengths has been implemented. Results obtained have demonstrated that AES encryption technique could be used to effectively make a textual information meaningless to intruders while it will be meaningful to the target receiver. In the future, a comparative analysis to measure the performance of these key lengths using the text length, encryption and decryption time, as well as memory consumption will be carried out. Also, the work will be extended to image encryption while the effect of the file size on the key lengths will also be studied.

REFERENCES

[AT13] **Abdulazeez A. M., Tahir A. S.** - *Design and Implementation of Advanced Encryption Standard Security Algorithm using FPGA*. Int. J. of Computers & Technology, 4(9), 1988–1993, 2013. Retrieved from http://cirworld.com/journals/index.php/ijct/article/view/5119

[AAF17] **Awadallah A., Abdelrahman A. A., Fouad M. M.** - *High Performance CUDA AES Implementation : A Quantitative Performance Analysis Approach*. In Computing Conference London, UK (pp. 1–9), 2017.

[FA17] **Farooq U., Aslam M. F.** - *Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA*. Journal of King Saud University - Computer and Information Sciences, 29(3), 295–302, 2017. https://doi.org/10.1016/j.jksuci.2016.01.004

[KR16] **Kumar P., Rana S. B.** - *Development of modified AES algorithm for data security*. Optik, 127(4), 2341–2345, 2016. https://doi.org/10.1016/j.ijleo.2015.11.188

[NGN17] **Nti I. K., Gymfi E., Nyarko O.** - *Advancements in Technology Implementation of Advanced Encryption Standard Algorithm with Key Length of 256 Bits for Preventing Data Loss in an Organization*, 8(2), 2017. https://doi.org/10.4172/0976-4860.1000183

[PDP12] **Pitchaiah M., Daniel P., Praveen** - *Implementation of Advanced Encryption Standard Algorithm*. International Journal of Scientific & Engineering Research, 3(3), 1–6, 2012.

[P+16] **Patil P., Narayankar P., Narayan D. G., Meena S. M.** - *A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish*. Procedia Computer Science, 78 (December 2015), 617–624, 2016. https://doi.org/10.1016/j.procs.2016.02.108

[SFH16] **Smekal D., Frolka J., Hajny J.** - *Acceleration of AES Encryption Algorithm Using Field Programmable Gate Arrays*. IFAC-PapersOnLine, 49(25), 384–389, 2016. https://doi.org/10.1016/j.ifacol.2016.12.075