

---

# **PROJECT REPORT**

**ON**

## **SMS Authentication Code Generated by using Advanced Encryption standard (AES)128 bits**

**Prepared by  
Tejas Palwe  
Software Engineering  
202190019**

**(Prepared under the guidance of Prof.Pavan Gedam)**

# Table of Contents

<b>1. Introduction .....</b>	<b>1</b>
1.1 Abstract .....	1
1.2 Introduction to Cryptography.....	1
1.3 Introduction to Advance Encryption standard .....	1
1.4 Project Scope.....	1
.....	1
<b>2. Advance Encryption Standard Description .....</b>	<b>2</b>
2.1 Description of the AES Algorithm.....	2
2.2 AES Operation .....	2
• Sub Bytes.....	2.2.1
• Shift Row.....	2.2.2
• Mix Column.....	2.2.3
• Add Round Key.....	2.2.4
2.3 The Key Expansion.....	2.3
2.4 AES Decryption.....	2.4
2.5 Flow Diagram.....	2.5
<b>3. Existing System.....</b>	<b>3</b>
3.1 Abstract.....	3.1
3.2 Introduction.....	3.2
3.3 Implementation.....	3.3
3.4 Disadvantage.....	3.4
<b>4. Proposed System.....</b>	<b>4</b>
4.1 Abstract.....	4.1
4.2 Introduction.....	4.2
4.3 Implementation.....	4.3
4.4 Advantages.....	4.4
4.5 Real Time Application.....	4.5
4.6 Screenshot.....	4.6
<b>5. References.....</b>	<b>5</b>
<b>6. Conclusion.....</b>	<b>6</b>

## Revision History

Name	Date	Reason For Changes	Version
Eddy Prasetyo Nugroho, Rizky Rachman Judhie Putra, Iman Muhamad Ramadhan	27/8/2016	SMS Authentication Code Generated by Advance Encryption Standard (AES) 256 bits Modification Algorithm and One Time Password (OTP) to Activate New Applicant Account	1.0
Tejas Palwe	30/4/2021	Adding more functionality to make it more secure and overcome drawback of existing system.	1.1

# 1. Introduction

## 1.1 Abstract

On October, 2, 2000, The National Institute of Standards and Technology (NIST) announced Rijndael as the new Advanced Encryption Standard (AES). The Predecessor to the AES was Data Encryption Standard (DES) which was considered to be insecure because of its vulnerability to brute force attacks. DES was a standard from 1977 and stayed until the mid 1990's. However, by the mid 1990s, it was clear that the DES's 56-bit key was no longer big enough to prevent attacks mounted on contemporary computers, which were thousands of times more powerful than those available when the DES was standardized. The AES is a 128 bit Symmetric block Cipher.

This thesis includes the complete step by step implementation of Advanced Encryption Technique, i.e. encrypting and decrypting 128 bit data using the AES and its modification for enhanced reliability and security. The encryption process consists of the combination of various classical techniques such as substitution, rearrangement and transformation encoding techniques. The encryption and decryption modules include the Key Expansion module which generates Key for all iterations. The modifications include the addition of an arithmetic operation and a route transposition cipher in the attacks iterative rounds. The Key expansion module is extended to double the number of iterative processing rounds in order to increase its immunity against unauthorized attacks

## 1.2 Introduction to cryptography

Cryptography is the science of information and communication security. Cryptography is the science of secret codes, enabling the confidentiality of communication through an insecure channel. It protects against unauthorized parties by preventing unauthorized alteration of use. It uses an cryptographic system to transform a plaintext into a cipher text, using most of the time a key. There exists certain cipher that doesn't need a key at all. An example is a simple Caesar-cipher that obscures text by replacing each letter with the letter thirteen places down in the alphabet. Since our alphabet has 26 characters, it is enough to encrypt the cipher text again to retrieve the original message

### 1.3 Introduction to the Advanced Encryption Standard.

The Advanced Encryption Standard, in the following referenced as AES, is the winner of the contest, held in 1997 by the US Government, after the Data Encryption Standard was found too weak because of its small key size and the technological advancements in processor power. Fifteen candidates were accepted in 1998 and based on public comments the pool was reduced to five finalists in 1999. In October 2000, one of these five algorithms was selected as the forthcoming standard: a slightly modified version of the Rijndael.

The Rijndael, whose name is based on the names of its two Belgian inventors, Joan Daemen and Vincent Rijmen, is a Block cipher, which means that it works on fixed-length group of bits, which are called *blocks*. It takes an input block of a certain size, usually 128, and produces a corresponding output block of the same size. The transformation requires a second input, which is the secret key. It is important to know that the secret key can be of any size (depending on the cipher used) and that AES uses three different key sizes: 128, 192 and 256 bits.

While AES supports only block sizes of 128 bits and key sizes of 128, 192 and 256 bits, the original Rijndael supports key and block sizes in any multiple of 32, with a minimum of 128 and a maximum of 256 bits.

### 1.4 Project Scope

- AES algorithm is most used for security purpose has it was most secure algorithm compared to DES and triple DES .
- In existing system it has been used for otp verification .we can used it for ATM money withdrawal purpose. we are can also make used of two factor or three factor authentication purpose. so that it can be more secure.
- As per record till now existing cyber attack would be able to break the AES algorithm as it was using great combination of key which was not easy to guess by attacker .
- SMS is the most common and major means of information exchange. This data can contain sensitive and vital information which needs to be protected. Which can be done using encryption. For this we have studied cryptographic algorithms.
- We devised that though asymmetric algorithm require 2 independent keys to encrypt and decrypt , it uses complex mathematical functions and is inefficient for small mobile devices. Hence we use symmetric algorithm for encryption. Also, among symmetric algorithms AES is the most efficient and resistive to brute force attack

## 2. Advanced Encryption Standard(AES)

### 2.1 Description Of AES

AES is an iterated block cipher with a fixed block size of 128 and a variable key Length. The different transformations operate on the intermediate results, called state. The state is a rectangular array of bytes and since the block size is 128 bits, which is 16 bytes, the rectangular array is of dimensions 4x4. (In the Rijndael version with variable block size, the row size is fixed to four and the number of columns varies. The number of columns is the block size divided by 32 and denoted Nb). The cipher key is similarly pictured as a rectangular array with four rows. The number of columns of the cipher key, denoted Nk, is equal to the key length divided by 32.

A state:

	a0,0		a0,1		a0,2		a0,3	
	a1,0		a1,1		a1,2		a1,3	
	a2,0		a2,1		a2,2		a2,3	
	a3,0		a3,1		a3,2		a3,3	

A key:

	k0,0		k0,1		k0,2		k0,3	
	k1,0		k1,1		k1,2		k1,3	
	k2,0		k2,1		k2,2		k2,3	
	k3,0		k3,1		k3,2		k3,3	

It is very *important* to know that the cipher input bytes are mapped onto the state bytes in the order a0,0, a1,0, a2,0, a3,0, a0,1, a1,1, a2,1, a3,1 ... and the bytes of the cipher key are mapped onto the array in the order k0,0, k1,0, k2,0, k3,0, k0,1, k1,1, k2,1, k3,1 ... At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order. AES uses a variable number of rounds, which are fixed: A key of size 128 has 10 rounds. A key of size 192 has 12 rounds. A key of size 256 has 14 rounds.

During each round, the following operations are applied on the state :-

1. **Sub Bytes** :- every byte in the state is replaced by another one, using the Rijndael S-Box.
2. **Shift Row** :- every row in the 4x4 array is shifted a certain amount to the left.
3. **Mix Column** :- a linear transformation on the columns of the state
4. **AddRoundKey** :- each byte of the state is combined with a round key.

## 2.2 AES operations: SubBytes, ShiftRow, MixColumn and AddRoundKey

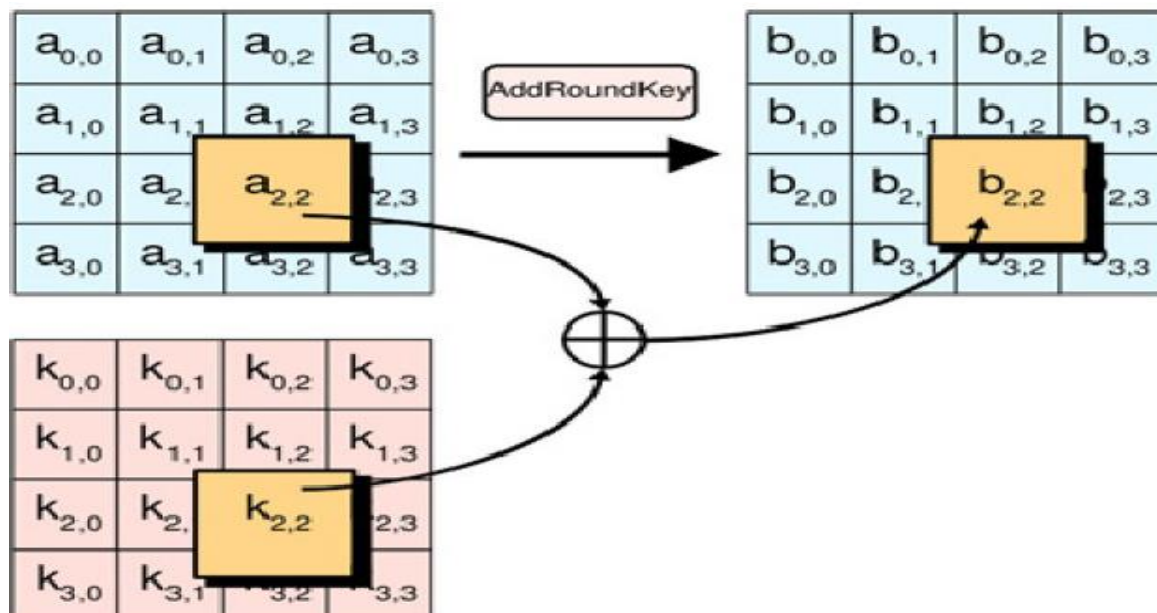
- **The Add RoundKey operation :-**

In this operation, a Round Key is applied to the state by a simple bitwise XOR. The Round Key is derived from the Cipher Key by the means of the key schedule. The Round Key length is equal to the block key length (=16 bytes).

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \text{ XOR } \begin{array}{|c|c|c|c|} \hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

where:  $b(i,j) = a(i,j) \text{ XOR } k(i,j)$

A graphical representation of this operation can be seen below:



- **The Shift Row operation**

In this operation, each row of the state is cyclically shifted to the left, depending on the row index.

**The 1st row is shifted 0 positions to the left.**

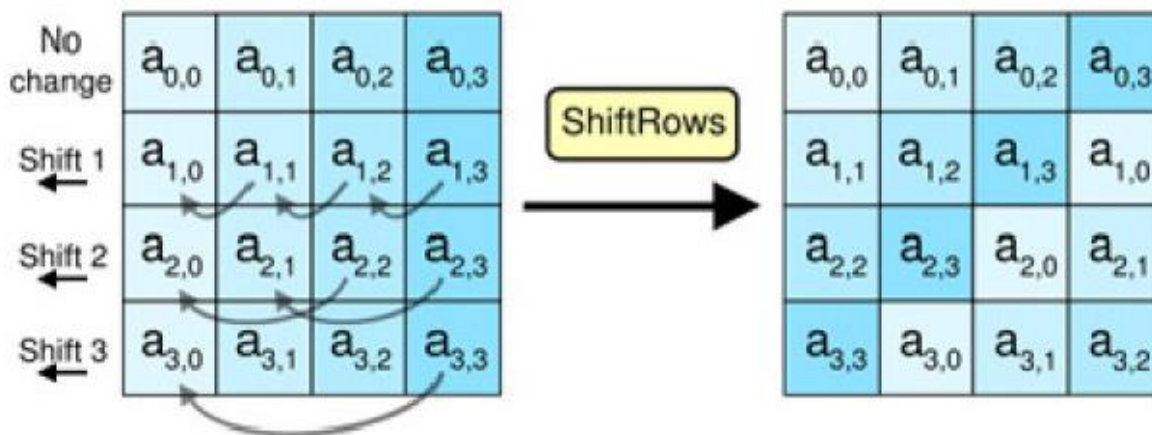
**The 2nd row is shifted 1 position to the left.**

**The 3rd row is shifted 2 positions to the left.**

**The 4th row is shifted 3 positions to the left.**

a <sub>0,0</sub>   a <sub>0,1</sub>   a <sub>0,2</sub>   a <sub>0,3</sub>		a <sub>0,0</sub>   a <sub>0,1</sub>   a <sub>0,2</sub>   a <sub>0,3</sub>
a <sub>1,0</sub>   a <sub>1,1</sub>   a <sub>1,2</sub>   a <sub>1,3</sub>	->	a <sub>1,1</sub>   a <sub>0,2</sub>   a <sub>1,3</sub>   a <sub>1,0</sub>
a <sub>2,0</sub>   a <sub>2,1</sub>   a <sub>2,2</sub>   a <sub>2,3</sub>		a <sub>2,2</sub>   a <sub>2,3</sub>   a <sub>2,0</sub>   a <sub>2,1</sub>
a <sub>3,0</sub>   a <sub>3,1</sub>   a <sub>3,2</sub>   a <sub>3,3</sub>		a <sub>3,3</sub>   a <sub>3,0</sub>   a <sub>3,1</sub>   a <sub>3,2</sub>

A graphical representation of this operation can be found below:



The inverse of **Shift Row** is the same cyclically shift but to the right. It will be needed later for decoding.

- **The SubBytes operation**

The **SubBytes** operation is a non-linear byte substitution, operating on each byte of the state independently. The **substitution table (S-Box)** is invertible and is constructed by the composition of two transformations. Since the S-Box is independent of any input, pre-calculated forms are used. Each byte of the state is then substituted by the value in the S-Box whose index corresponds to the value in the state:

$$a(i, j) = \text{SBox}[a(i, j)]$$

The inverse of SubBytes is the same operation, using the inversed S-Box, which is also precalculated.

	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	0f
0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
40	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16



- **The MixColumn operation**

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes.

During this operation, each column is transformed using a fixed matrix (matrix left-multiplied by column gives new value of column in the state)

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

**Predefine Matrix**

**State Array**

**New State Array**

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

- **Add Round Key**

The the forward add round key transformation called Add Round key the 128 bits of state are bitwise XORed with the 128 bits of the round key.

As Shown in Figure the operation is viewed as a column wise operation between the 4bytes Of a state column and word of the round key it can also be viewed as a byte level operation.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

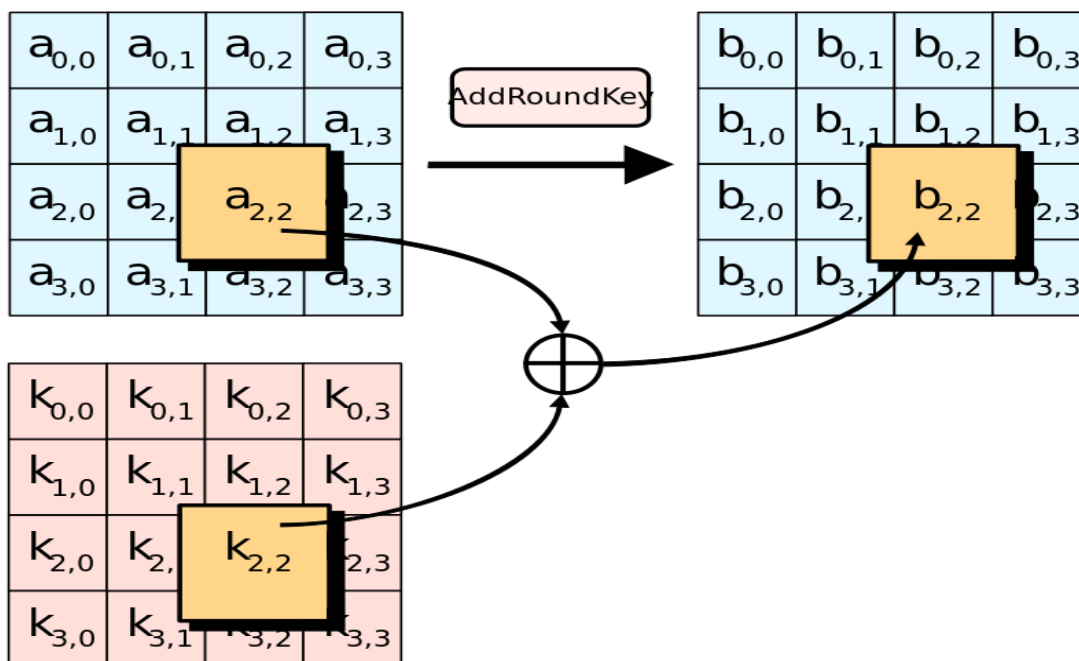
 $\oplus$ 

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

 $=$ 

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main key using Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.



## 2.3 The Key Expansion

. The AES key expansion algorithm takes as input a four-word (16 byte) key and produces a linear array of 44 words (176 bytes).

This is sufficient to provide a four-word round key for the initial Add Round Key stage and each of the 10 rounds of the cipher.

The pseudocode on the next page describes the expansion.

The key is copied into the first four words of the expanded key.

The remainder of the expanded key is filled in four words at a time.

Each added word  $w[i]$  depends on the immediately preceding word,  $w[i - 1]$ , and the word four positions back,  $w[i - 4]$ . In three out of four cases, a simple XOR is used.

For a word whose position in the  $w$  array is a multiple of 4, a more complex function is used. Figure the generation of the expanded key, using the symbol  $g$  to represent that complex function. The function  $g$  consists of the following subfunctions.

```

KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                   key[4*i+2],
                                   key[4*i+3]);

    for (i = 4; i < 44; i++)
    {
        temp = w[i - 1];
        if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                               ⊕ Rcon[i/4];

        w[i] = w[i-4] ⊕ temp
    }
}

```

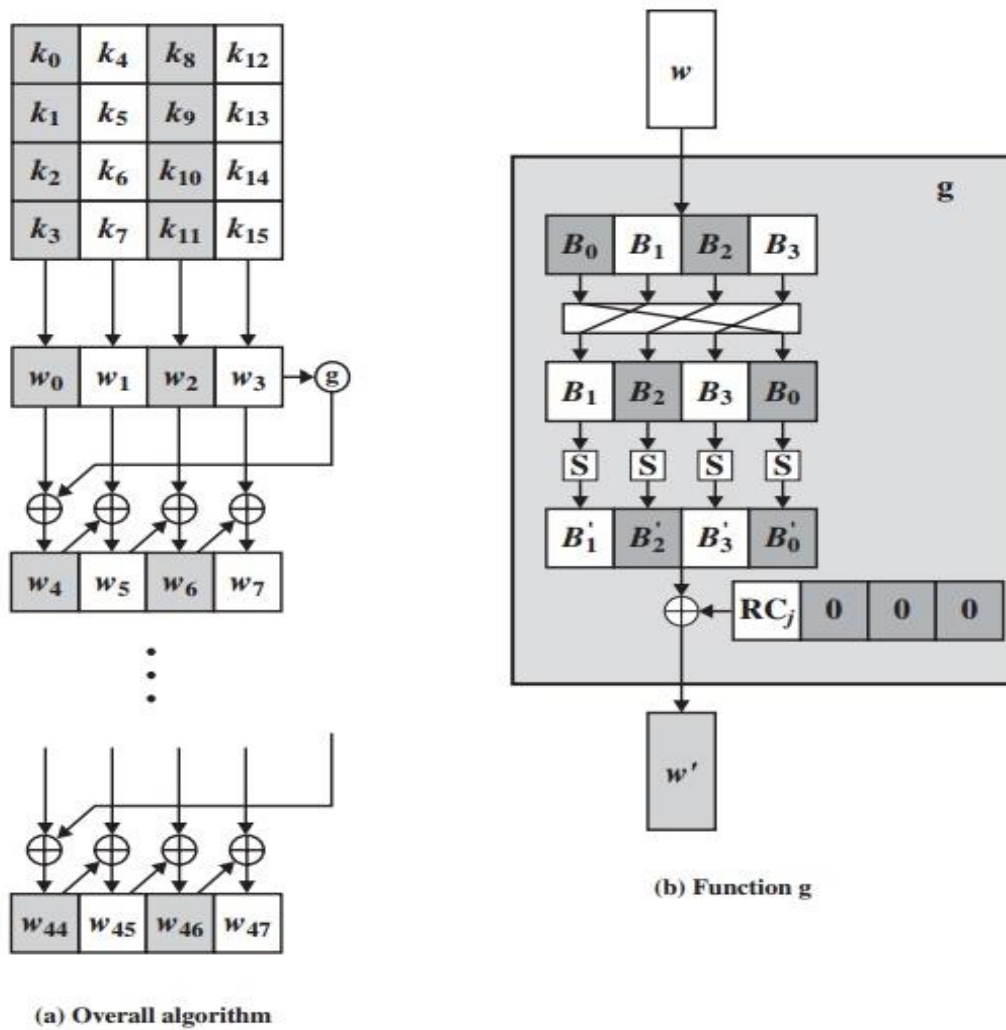


Figure 5.9 AES Key Expansion

- RotWord performs a one-byte circular left shift on a word. This means that an input word  $[B_0, B_1, B_2, B_3]$  is transformed into  $[B_1, B_2, B_3, B_0]$ .
- SubWord performs a byte substitution on each byte of its input word, using the S-box (Table 5.2a).
- The result of steps 1 and 2 is XORed with a round constant,  $Rcon[j]$ .

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as  $Rcon[j] = (RC[j], 0, 0, 0)$ , with  $RC[1] = 1$ ,  $RC[j] = 2 RC[j-1]$  and with multiplication defined over the field  $GF(2^8)$ . The values of  $RC[j]$  in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	$w[i-4]$	$w[i] = temp \oplus w[i-4]$
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

## 2.4 AES Decryption

To decrypt an AES-encrypted ciphertext, it is necessary to undo each stage of the encryption operation in the reverse order in which they were applied. The three stage of decryption are as follows:

Inverse Final Round

- AddRoundKey
- ShiftRows
- SubBytes

Inverse Main Round

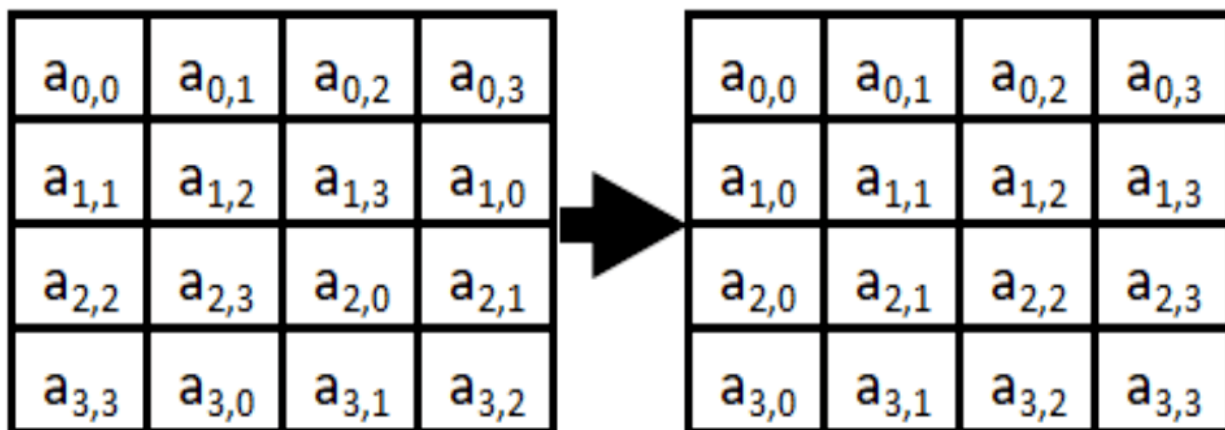
- AddRoundKey
- MixColumns
- ShiftRows
- SubBytes

Inverse Initial Round

- AddRoundKey

Of the four operations in AES encryption, only the AddRound Key operation is its own inverse (since it is an exclusive-or). To undo Add RoundKey, it is only necessary to expand the entire AES key schedule (identically to encryption) and then use the appropriate key in the exclusive-or.

The other three operations require an inverse operation to be defined and used. The first operation to be undone is Shift Rows. The Inverse Shift Rows operation is identical to the Shift Rows operation except that rotations are made to the right instead of to the left. This is illustrated in the Figure below.



The next operation to be undone is the SubBytes operation. The Inverse S-Box is shown in the Table below. It is read identically to the S-Box matrix.

	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	0f	
0	52	9	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	8	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	0	8c	bc	d3	0a	f7	e4	58	5	b8	b3	45	6
70	d0	2c	1e	8f	ca	3f	0f	2	c1	af	bd	3	1	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	7	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	4	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

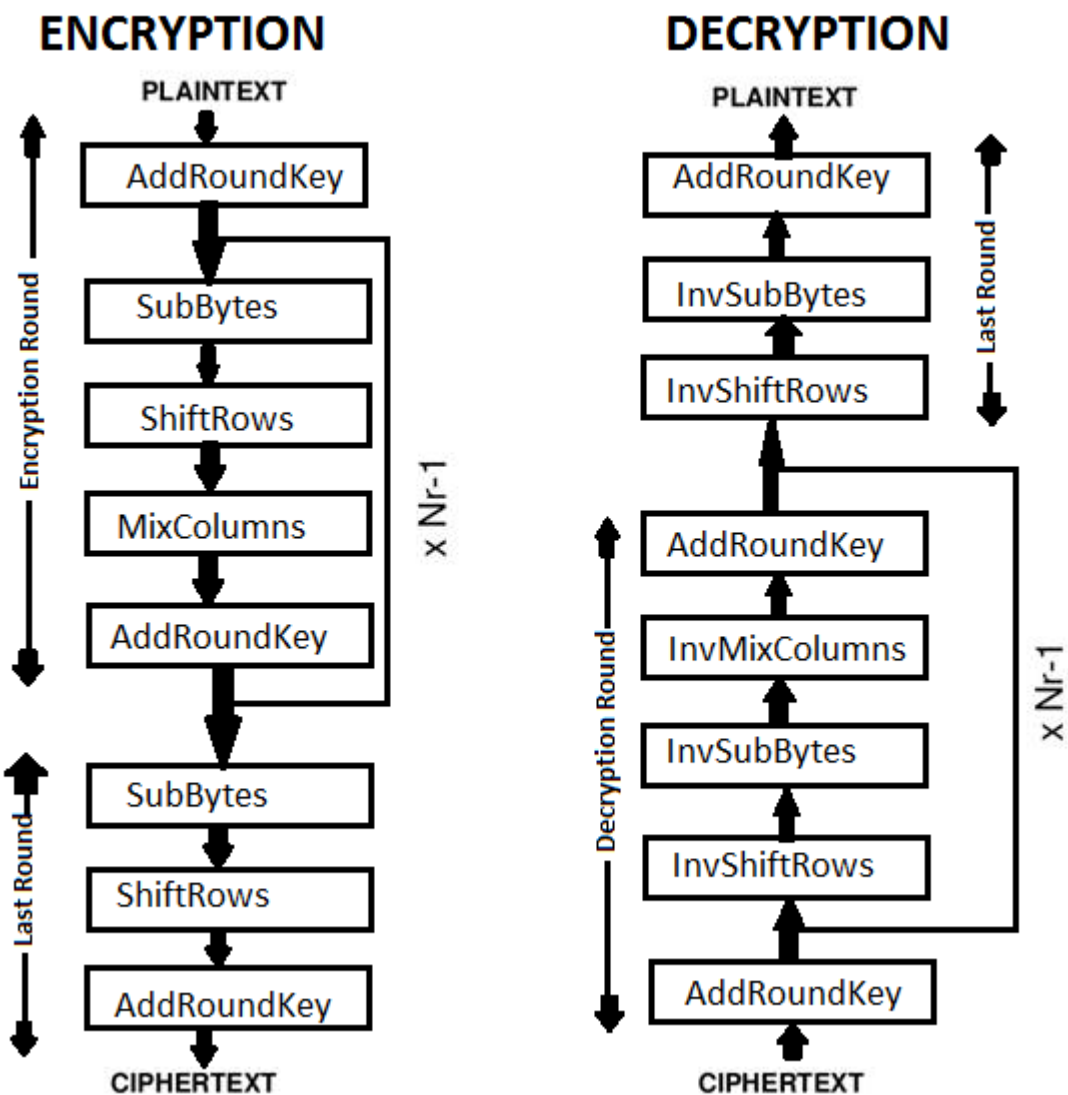
The last inverse operation to define is MixColumns. Like MixColumns, Inverse MixColumns can be defined as the matrix multiplication in Galois Field 28. This is illustrated in the Figure below.

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline 14 & 11 & 13 & 9 \\ \hline 9 & 14 & 11 & 13 \\ \hline 13 & 9 & 14 & 11 \\ \hline 11 & 13 & 9 & 14 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

The reason why this multiplication inverts the initial operation is because of how math works in the Galois Field 28, which we won't describe in detail in this tutorial. However, note that the specific values in both matrices are chosen in a way such that one multiplication is the inverse of the other in Galois Field 28.

AES ciphertexts are decrypted by following the order of operations explained at the beginning of this section, using the appropriate inverse operations and using round keys in reverse order.

## 2.5 Flow Digaram





### 3. Existing Sytem

#### 3.1 Abstract

Nowadays, it is highly necessary to give precautions aimed to prevent Netizen from creating fake accounts to do criminal offence. One of the means is to use authentication code where it contains activation message. This authentication code is generated through activation message combined with timestamp values which would further be used on One-time Password. Then, it would be encrypted using Advanced Encryption Standard cryptographic algorithm, the generated authentication code can only be used once within a limited time. Unfortunately though, in 2011 three researchers from several universities and Microsoft, Andrey Bogdanov from K.U. Leuven, Dimitri Khovratovich from Microsoft and Christian Reachberger from ENS Paris found a crack on AES encryption, thus requiring modification to enhance AES complexity in order to close the said crack. AES modification can be done on S-Box and ShiftRow, thus enabling their dynamics following the keys given. 256 bits AES is chosen because it contains more key combination and longer time to decrypt compared to other type of AES. After its modification is done, the next step would be tested with Avalanche Effect and Randomness Test, where good cryptographic algorithm would have Avalanche Effect values around 50% and able to pass 5 basic random tests in Randomness Test.

#### 3.2 Introduction

The ever increasing social media and online purchase activities in cyber world has created a phenomenon where there are certain individuals or interest groups use it to benefit themselves or disadvantage the others, by registering fake accounts or bots, thus enabling these irresponsible actors to do improper acts, such as deception, insult, defamation, or falsified support, even black campaign during presidential election.

This obviously harms and gives bad image toward the social media owner or online trading sites. The emergence of rogue and fake accounts. Therefore, preventive measures are necessary in order to minimize those rogue and fake accounts in the social media and online trading sites.

One of the ways is to enhance security measures in registration process by generating authentication code to verify and activate the account. Account activation can implement cryptographic algorithm within the activation message containing authentication code sent through SMS to the mobile phone's number to ensure whether it's bot. This authentication code is in the form of encrypted disposable message and only has rather short lifespan

### 3.3 Implementation

One time password concept originated from the need to do periodic password replacement in order for the password to stay safe and not misused, password replacement is done automatically.

The generated passwords cannot be reused (non-reusable), so it's useless if someone finds their password, because the password will expire. Each password is used only once.

OTP provides immunity and security from possible password sniffing attacks, even if someone peek your password, they still cannot get access to your account

Here is how One Time Password works and the example in table II:

Taking the value of timestamp when OTP is generated.

Combining value of OTP unit time with a secret message to be encrypted

Encrypt the combined result of the value of the unit of OTP time with a secret message

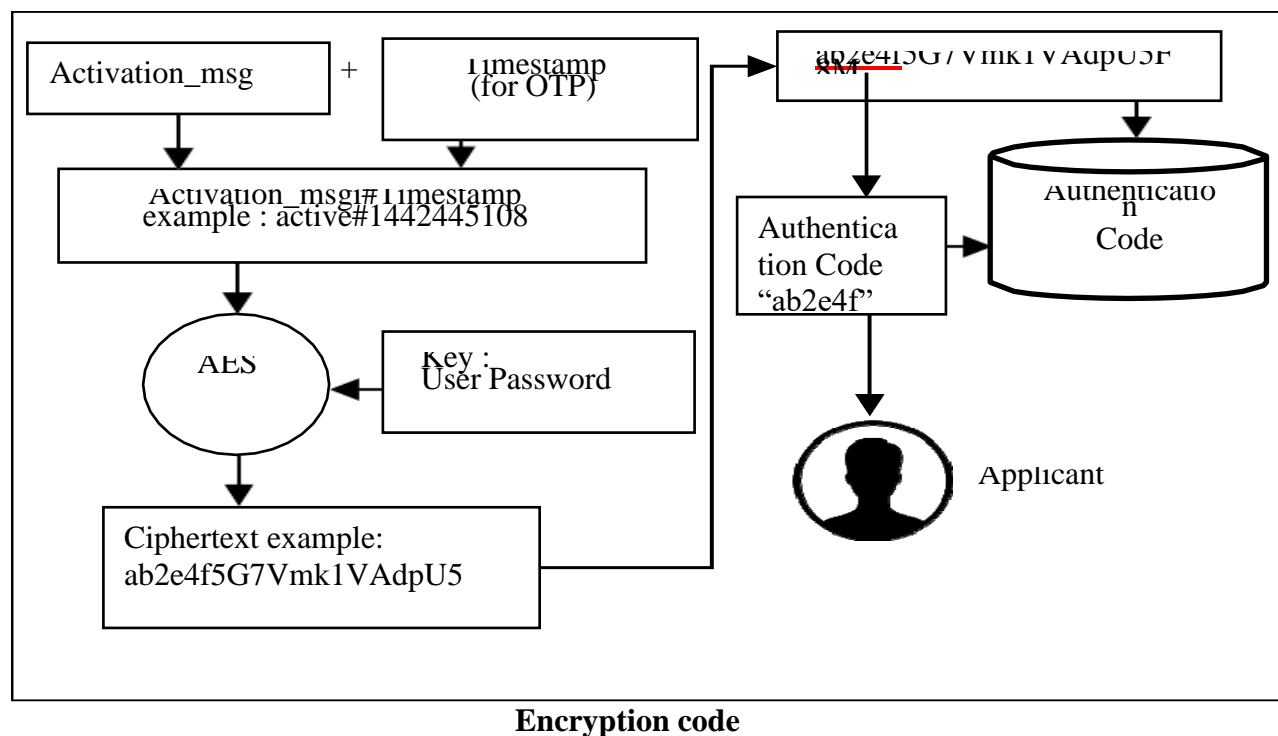
Take some initial digits from encryption to be used as a One Time Password.

#### . EXAMPLE OF ONE TIME PASSWORD

Messa ge	Timestamp OTP	Combine	MD5(Combine)	OTP
Hallo	12485046	hallo12485046	b6bd54116198900ce703379	b6bd5
Hallo	12485047	hallo12485047	aa94894ff7c66e9a8b24f95d	aa948
Hallo	12485048	hallo12485048	b1cdb9995454a2bd7316a645	b1cdb
Hallo	12485049	hallo12485049	db2aef1c7e4c2de3c1be7783f4	db2ae

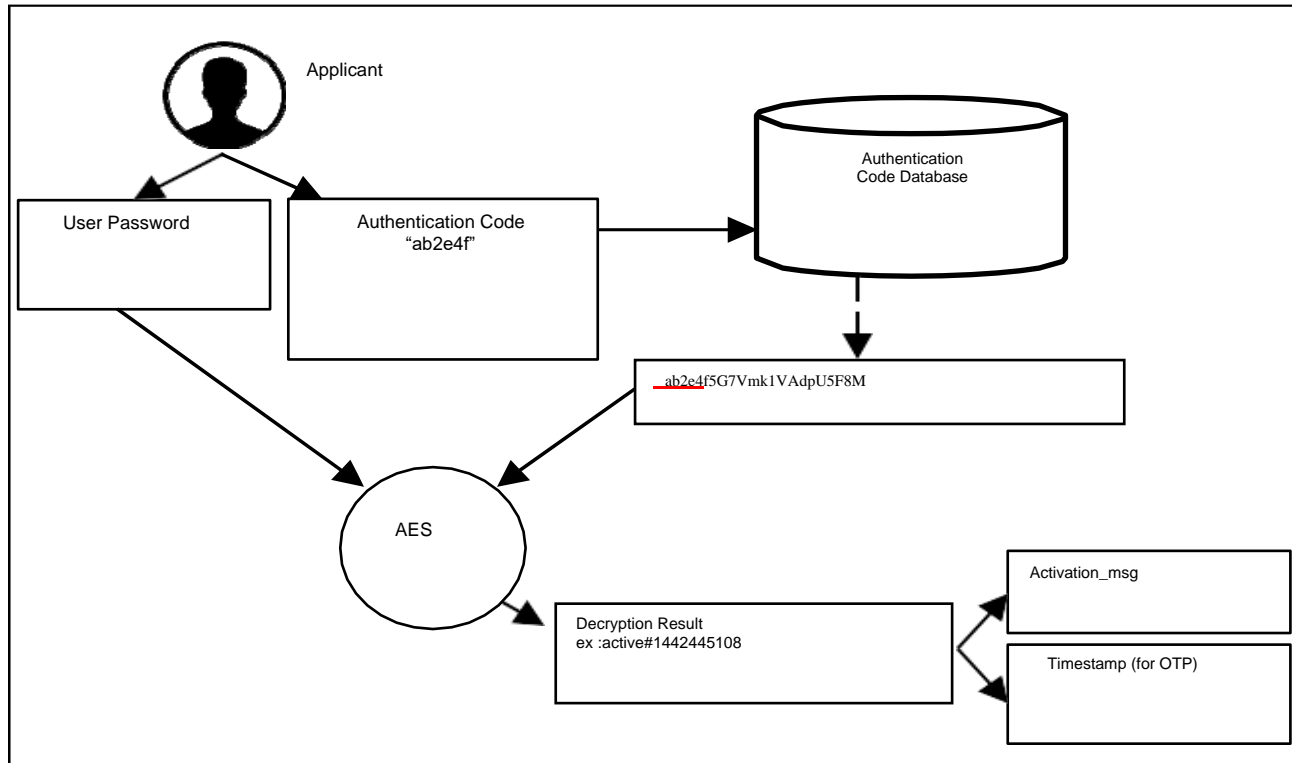
- **Steps to Encrypt and Decrypt Authentication Code**

In making the authentication codes it is needed to know how to generate or dismantle it, therefore, there is needs to design how to call upon and unload authentication code, so that the system will be running as planned, describes in the below flow to generate and decode authentication.



The flow to dismantle the authentication code that has been given to the applicant is as follows.

1. Applicants enter the authentication code received along with the registered password
2. The authentication code entered by applicant would be looked out cipher text pair on the database of authentication code.
3. After the cipher text is found then the next step is the process of decryption keys belonging to the applicant in the form of a password.
4. Then the results obtained from the decrypted cipher text is plaintext.
5. Plaintext is then separated between an activation message to the value of the unit of time to be calculated whether the authentication code is valid.



**Decryption code**

This is the flow to generate the authentication codes.

1. Set up activation message and the value of the unit time (timestamp) then combine both into plaintext
2. .
3. Having obtained plaintext, the next step is to put it into AES 256 encryption key that has been given in the form of applicant's passwords.
4. After going through the encryption process, the cipher text will be obtained..
5. Then the first 6 digits of cipher text produced will be taken to be used as the authentication code, then the cipher text and authentication code will be stored in the database.
6. The next step is to send the authentication code to the registrar to be entered on the stage of activation of a new account

After Applicant receive authentication code, Applicant can use that code to activate their account. Fig. 9 is the scheme how to decrypt Authentication Code.

### **3.4 Disadvantages**

1. It uses too simple algebraic structure.
2. Every block is always encrypted in the same way.
3. Hard to implement with software.
4. The message encryption is only for sending receiving otp.

## 4. Proposed system

### 4.1. Abstract

Encryption is of great importance once the confidentiality of information is to be maintained over the network. SMS being one of the major means of data exchange among the mobile users. Security of SMS is one of the major issue that must be handled during data transmission. So, by using Android technology an application have been developed by us which permits the sender to encode the messages before they are sent over the network. For the encryption and decryption process we have used Advanced Encryption Standard (AES) as the cryptographic algorithm. The application allows the user to input the key and the message which has to be encrypted and hence generate encrypted message which can be decrypted by the receiver. by using symmetric key which has been shared with receiver the OTP is send on end user mobile or on email Which he has specified. And we has two factor authentication by using Aes algorithm to make our system more secure.

### 4.2 Introduction

- With the technological development and era of digitization, nowadays exchange of messages, thoughts or information are done by using various message sending applications.
- Security of data plays an important part in wireless communication system. This communication involves exchange of data between a sender and a receiver, where both the end users seeks the security of their shared information.

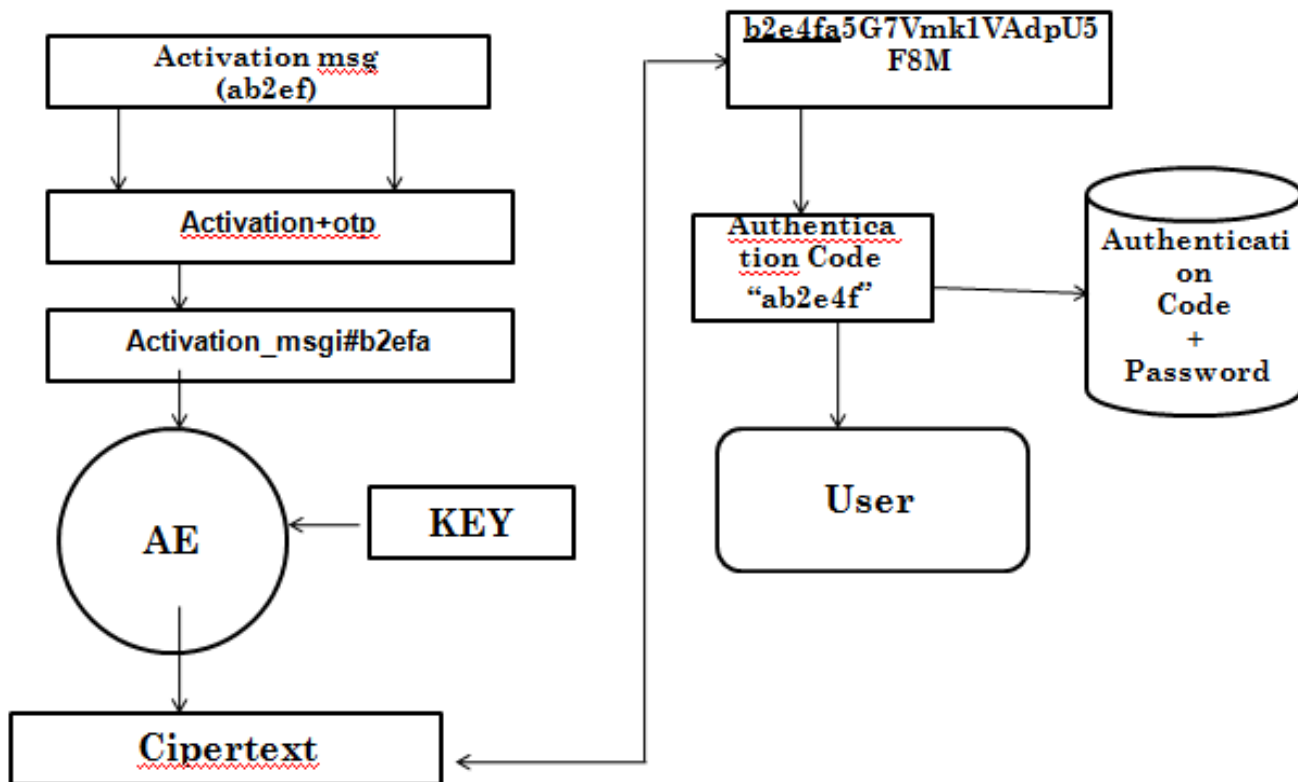
#### Need for secure data transmission

- Protecting the exchanged information from unauthorized access, disclosure, use, destruction, modification or inspection is known as information security. Maintaining privacy in our personal communication are a few things that everybody wishes.
- This secrecy of data can be maintained by means of cryptography. In cryptography, security is ensured by encoding the data before sending it and decoding the data after receiving it. Various cryptographic algorithms are used to ensure that privacy of data is maintained. Nowadays, SMS that stands for short message service is widely accepted as a means of information exchange, its security has become a significant concern for numerous business concern and customers. So, there is a great requirement for an end to end SMS encryption in order to provide secure medium for communication
- AES and DES are most commonly accepted and used cryptographic algorithms.
- While DES uses 56 bit key and hence is unprotected against brute force attack AES is not susceptible to brute force attacks as it uses large sized keys

### 4.3. Implementation

#### Server side Encryption process

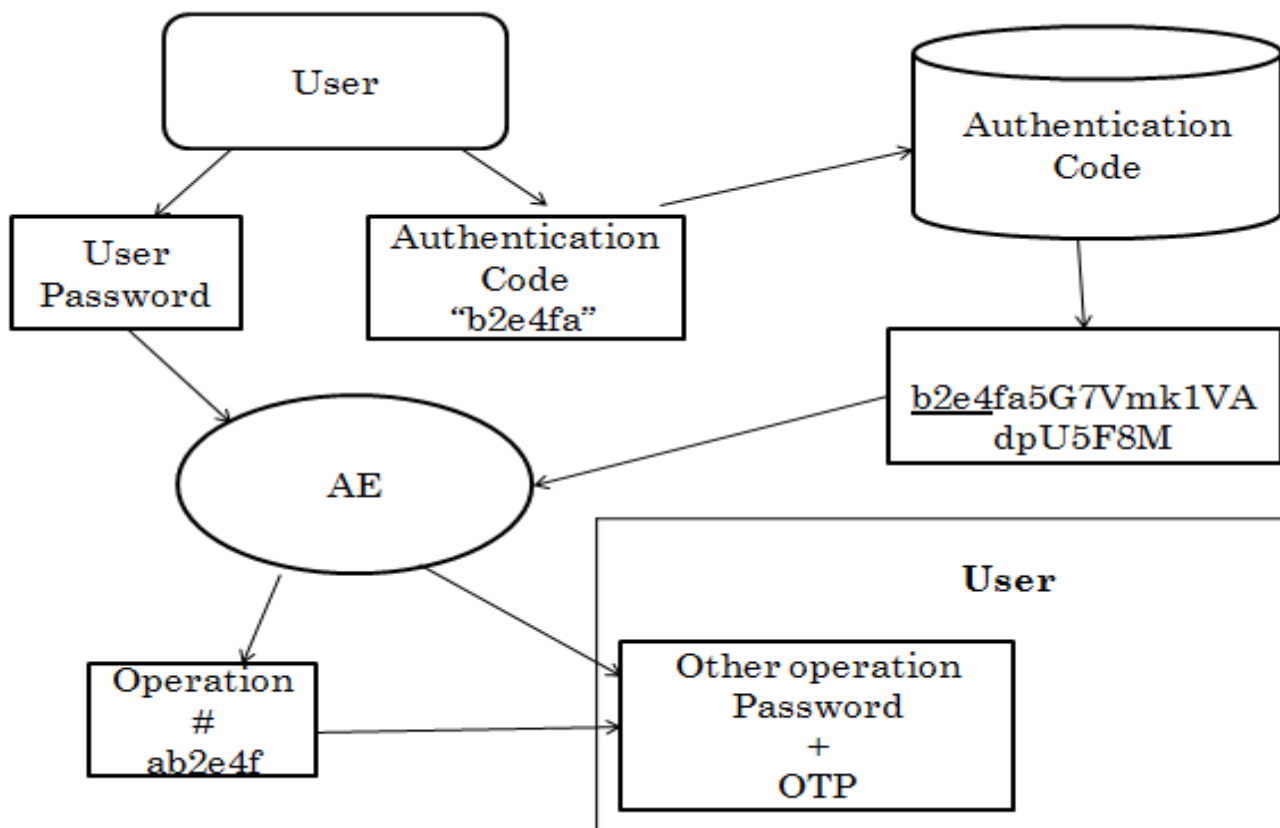
- Taking the Activation message which is needed to send over network.
- Combining value of OTP and activation message .
- Combined message is send to aes algorithm for encryption purpose along with the key Which is known to end user that is password.
- Encrypted cipertext which saved with system database and send to user through network .
- The database store the authentication code in combination with password.



## User side decryption process

This is the flow to generate the authentication codes.

1. User will get authentication code which has been send to him by server side.
2. User will have password which is hare to him at a time of registration so now user will have password and otp.
3. User need to enter password which key for algorithm to retrieve encrypted text.
4. After going through the encryption process, the cipher text will be obtained..
5. Then the first 6 digits of cipher text produced will be taken to be used as the authentication code, then the cipher text and authentication code will be stored in the database.
6. Then user used password along in combination with otp to get access of the system.





#### 4.4. Advantages

- As it is implemented in both hardware and software, it is most robust security protocol.
- It uses higher length key sizes such as 128, 192 and 256 bits for encryption. Hence it makes AES algorithm more robust against hacking.
- It is most common security protocol used for wide various of applications such as wireless communication, financial transactions, e-business, encrypted data storage etc.
- It is one of the most spread commercial and open source solutions used all over the world. No one can hack your personal information.
- For 128 bit, about  $2^{128}$  attempts are needed to break. This makes it very difficult to hack it as a result it is very safe protocol.
- Each time the user demand for the password the ciphertext get changed along with otp.

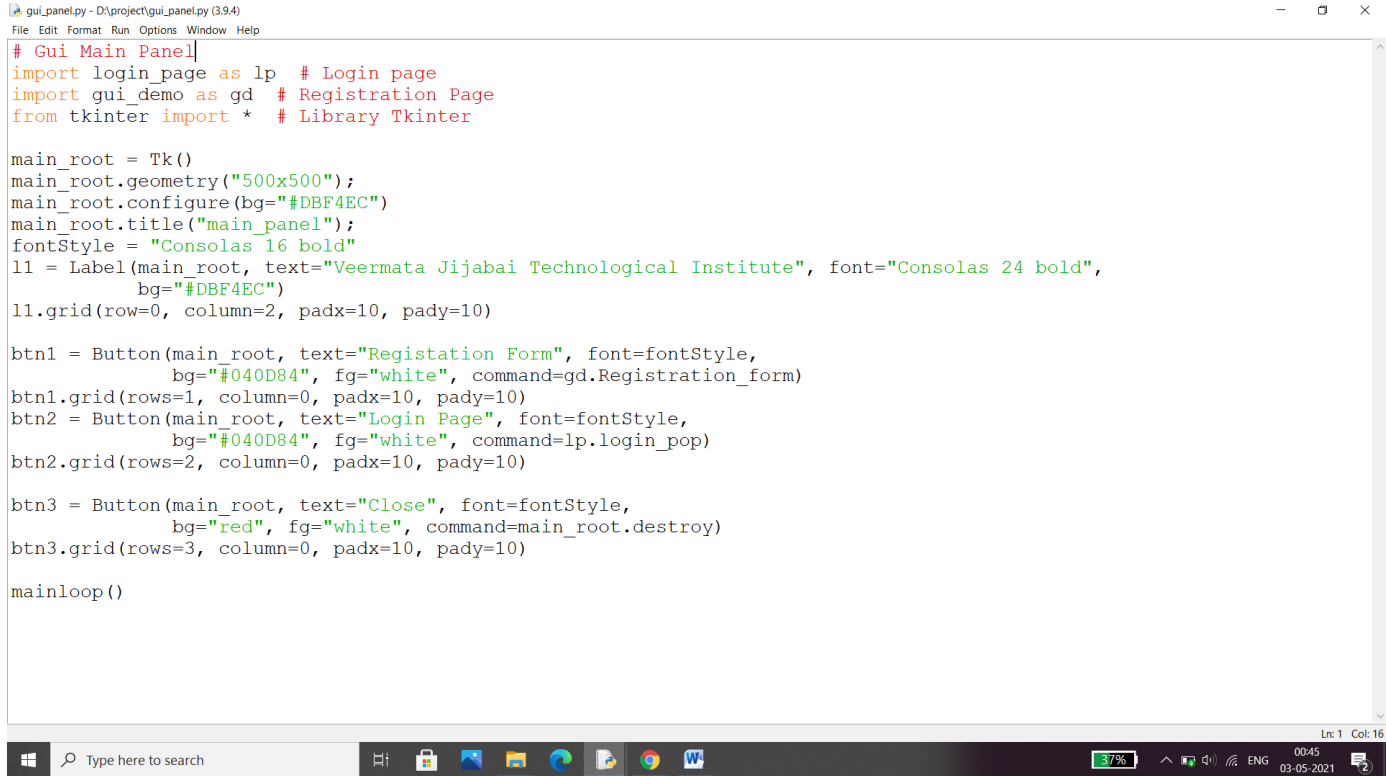
#### 4.5 Future Scope

Key distribution is also one of the major aspect that should be taken care of. As we are working with AES algorithm which is a symmetric encryption technique that is single to be used by both the sender and receiver so we also have to find ways to securely share the key.

Key can be distributed in one of the following ways:

1. Physical transfer of the key from sender to receiver.
2. Key can also be delivered to sender and receiver with the help of trusted third party.
3. Key used by sender and receiver previously can be converted to new Key using Encryption..
4. Key can be provided to both users with help of KDC.
5. Diffie-Hellman method can be used for secure exchange of keys.

## 4.6 Screenshots



```

gui_panel.py - D:\project\gui_panel.py (3.9.4)
File Edit Format Run Options Window Help

# Gui Main Panel
import login_page as lp # Login page
import gui_demo as gd # Registration Page
from tkinter import * # Library Tkinter

main_root = Tk()
main_root.geometry("500x500");
main_root.configure(bg="#DBF4EC")
main_root.title("main_panel");
fontStyle = "Consolas 16 bold"
l1 = Label(main_root, text="Veermata Jijabai Technological Institute", font="Consolas 24 bold",
            bg="#DBF4EC")
l1.grid(row=0, column=2, padx=10, pady=10)

btn1 = Button(main_root, text="Registration Form", font=fontStyle,
              bg="#040D84", fg="white", command=gd.Registration_form)
btn1.grid(rows=1, column=0, padx=10, pady=10)
btn2 = Button(main_root, text="Login Page", font=fontStyle,
              bg="#040D84", fg="white", command=lp.login_pop)
btn2.grid(rows=2, column=0, padx=10, pady=10)

btn3 = Button(main_root, text="Close", font=fontStyle,
              bg="red", fg="white", command=main_root.destroy)
btn3.grid(rows=3, column=0, padx=10, pady=10)

mainloop()

```

Gui code Screenshot



```

cryp_tejas.py - U:\project\cryp_tejas.py (3.9.4)
File Edit Format Run Options Window Help

#crypting_tejas
from Crypto.Cipher import AES
def tejas_cryp(password, message):
    User_keys = bytes(password, 'utf-8')
    print(User_keys)
    message = bytes(message, 'utf-8')
    print(message)
    #creation of cipher model
    cipher = AES.new(User_keys, AES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(message)
    return ciphertext, tag, nonce

```

Cryptographhy File

login\_page.py - D:\project\login\_page.py (3.9.4)

File Edit Format Run Options Window Help

```

from tkinter import *
import random
import check_db
fontStyle = "Consolas 16 bold"
email_check = ""
import decrypt_panel as dp
def login_pop():
    #otp generations
    def generate_otp():
        global email_check
        var1 = "";var2= ""
        var1 = te2.get(); var2 = te3.get()
        email_check =te2.get()
        te2.delete(0, 'end');te3.delete(0, 'end')
        check_db.check_db_data(var1, var2)

    #1st verifications
    def first_verif():
        #email; pass+otp; message
        var1 = "";var2 = ""
        var1 = te5.get();var2 = te6.get()
        te5.delete(0, 'end');te6.delete(0, 'end')
        check_db.first_verif_data(var1, var2, email_check)
        #update_aes_check(email_check,var1,var2)

    w = Tk()
    w.title("Login Page");w.geometry("500x500");w.configure(bg = "#DBF4EC")
    t11 = Label(w, text = "Login Page", font = "Consolas 24 bold",
    bg = "#DBF4EC");t11.grid(row= 0, column=1, padx=10, pady=10)
    #----login email
    #---Email id-----
    t12 = Label(w, text = "User Id: ", font = fontStyle,bg = "#DBF4EC" )
    t12.grid(row = 1, column =0, padx =10, pady =10)
    te2 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    te2.grid(row = 1, column =1, padx = 10, pady = 10)
    t13 = Label(w, text = "Mobile Number: +91- ", font = fontStyle,bg = "#DBF4EC" )
    t13.grid(row = 2, column =0, padx =10, pady =10)
    te3 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    te3.grid(row = 2, column =1, padx = 10, pady = 10)
    tbtn1 = Button(w, text = "Generate OTP", font = fontStyle,
    bg = "#008080", fg = "white", command = generate_otp)

```

## Login Page Code

login\_page.py - D:\project\login\_page.py (3.9.4)

File Edit Format Run Options Window Help

```

from tkinter import *
import random
import check_db
fontStyle = "Consolas 16 bold"
email_check = ""
import decrypt_panel as dp
def login_pop():
    #otp generations
    def generate_otp():
        global email_check
        var1 = "";var2= ""
        var1 = te2.get(); var2 = te3.get()
        email_check =te2.get()
        te2.delete(0, 'end');te3.delete(0, 'end')
        check_db.check_db_data(var1, var2)

    #1st verifications
    def first_verif():
        #email; pass+otp; message
        var1 = "";var2 = ""
        var1 = te5.get();var2 = te6.get()
        te5.delete(0, 'end');te6.delete(0, 'end')
        check_db.first_verif_data(var1, var2, email_check)
        #update_aes_check(email_check,var1,var2)

    w = Tk()
    w.title("Login Page");w.geometry("500x500");w.configure(bg = "#DBF4EC")
    t11 = Label(w, text = "Login Page", font = "Consolas 24 bold",
    bg = "#DBF4EC");t11.grid(row= 0, column=1, padx=10, pady=10)
    #----login email
    #---Email id-----
    t12 = Label(w, text = "User Id: ", font = fontStyle,bg = "#DBF4EC" )
    t12.grid(row = 1, column =0, padx =10, pady =10)
    te2 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    te2.grid(row = 1, column =1, padx = 10, pady = 10)
    t13 = Label(w, text = "Mobile Number: +91- ", font = fontStyle,bg = "#DBF4EC" )
    t13.grid(row = 2, column =0, padx =10, pady =10)
    te3 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    te3.grid(row = 2, column =1, padx = 10, pady = 10)
    tbtn1 = Button(w, text = "Generate OTP", font = fontStyle,
    bg = "#008080", fg = "white", command = generate_otp)

```

## Database Connection

check\_db.py - D:\project\check\_db.py (3.9.4)

File Edit Format Run Options Window Help

```

sdt.send_otp(otp, email, mobile_number)
print(new_otp) #needs to send to mobile& email id
command = f"""update login_demo_01 set otp = ?
where email_id = ?"""
conn.execute(command, (new_otp, email))
conn.commit()
conn.close()
#1st verification
def first_verif_data(checkotp, new_message, email):
    conn = sqlite3.connect("test_login_demo.db")
    command = "select * from login_demo_01 where email_id = ?"
    result = conn.execute(command, (email,))
    for i in result:
        email_id,mobile_number,password,otp,message,ciphertext,nonce,tag = i
        print(i)
        otp = str(otp)
    if checkotp==otp:
        #otp key
        #cmdnew_message #message to be crypted
        crypto_data = cryp_tejas.tejas_cryp(otp, new_message)
        ciphertext, tag, nonce = crypto_data
        new_4_otp = str(ciphertext)
        new_4_otp = new_4_otp[4:8]
        sdt.send_otp(new_4_otp, email, mobile_number) #needs to send to the mobile and email
        command = f"""update login_demo_01 set ciphertext = ?, tag = ?, nonce = ?,
        message = ? where email_id = ?""" #new_4_otp must included
        conn.execute(command, (ciphertext, tag, nonce, new_4_otp, email))
        conn.commit()
        command = "select * from login_demo_01 where email_id = ?"
        result = conn.execute(command, (email,))
        for i in result:
            email_id,mobile_number,password,otp,message,ciphertext,nonce,tag = i
            print(i)
        conn.close()
def dc(password, crypto_otp):
    print(password, crypto_otp)
    conn = sqlite3.connect("test_login_demo.db")
    command = "select * from login_demo_01 where message = ?"
    result = conn.execute(command, (crypto_otp,))
    for i in result:
        email_id,mobile_number,password_not,otp,message,ciphertext,nonce,tag = i
    #decrypting part
    keys = bytes(password, 'utf-8')
    cipher = AES.new(keys, AES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)
    try:
        cipher.verify(tag)
        plaintext = plaintext.decode("utf-8")
        data = f"The message is authentic and access granted: {plaintext}"
        messagebox.showinfo("showinfo", data)
    except ValueError:
        print("Key incorrect or message corrupted",plaintext)

conn.close()

```

## Database Connection

decrypt\_panel.py - D:\project\decrypt\_panel.py (3.9.4)

File Edit Format Run Options Window Help

```

#decrypt_panel
from tkinter import *
import check_db as db
from Crypto.Cipher import AES
fontStyle = "Consolas 16 bold"
def decrypt_data():
    def send_crypto_data():
        var1 = "";var2="";
        var1 = e2.get(); var2 = e3.get()
        db.dc(var1,var2)

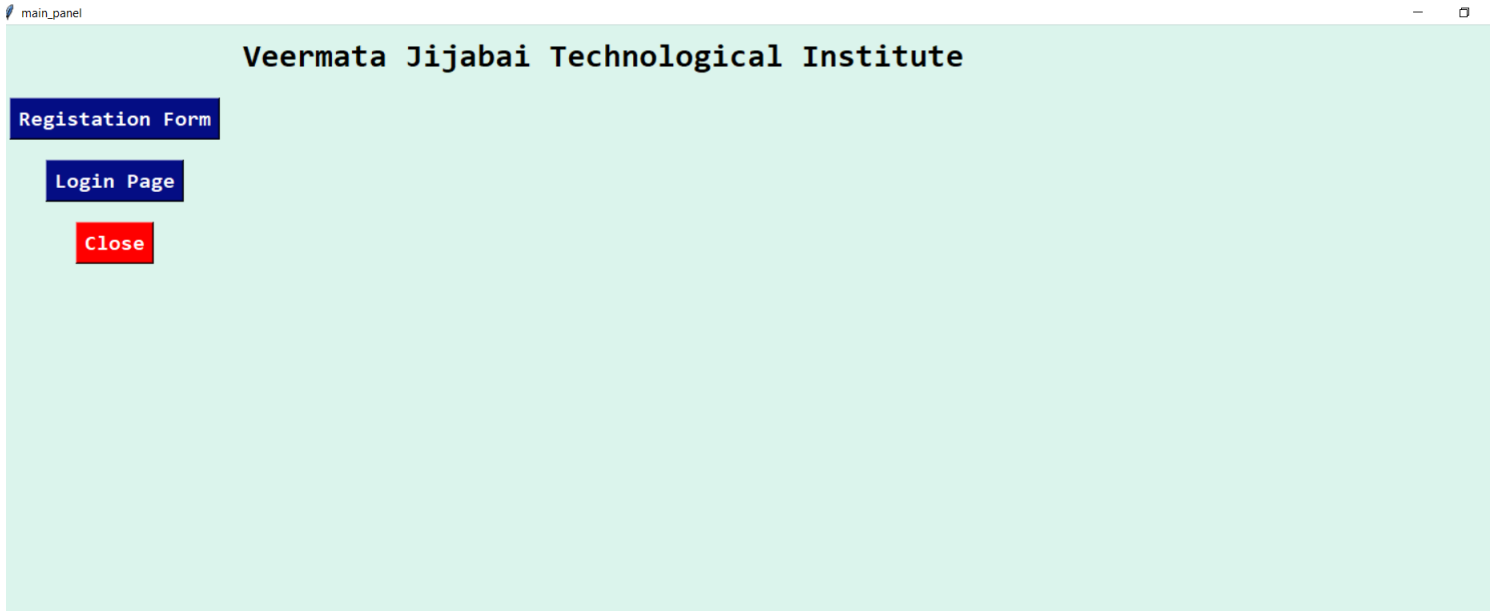
    w = Tk()
    w.title("Decrypter Message");w.geometry("500x500");w.configure(bg = "#DBF4EC")
    t11 = Label(w, text = "Decrypter", font = "Consolas 24 bold",
    bg = "#DBF4EC");t11.grid(row= 0, column=1, padx=10, pady=10)
    l2 = Label(w, text = "Password + OTP: ", font = fontStyle,bg = "#DBF4EC" )
    l2.grid(row = 1, column =0, padx =10, pady =10)
    e2 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    e2.grid(row = 1, column =1, padx = 10, pady = 10)

    l3 = Label(w, text = "Crypto OTP: ", font = fontStyle,bg = "#DBF4EC" )
    l3.grid(row = 2, column =0, padx =10, pady =10)
    e3 = Entry(w, bg = "white", fg = "black", font = fontStyle)
    e3.grid(row = 2, column =1, padx = 10, pady = 10)
    btn3 = Button(w, text = "Check Message", font = fontStyle,
    bg = "#040D84", fg = "white",command = send_crypto_data)
    btn3.grid(rows = 5, column = 1,padx =10, pady =10)

    btn3 = Button(w, text = "Close", font = fontStyle,
    bg = "red", fg = "white",command = w.destroy)
    btn3.grid(rows = 5, column = 1,padx =10, pady =10)
    w.mainloop()

```

## Decryption Message



Homepage Output

The screenshot shows a web browser window with the title 'basic project'. The page has a light green background. At the top center, the text 'Registration form' is displayed in a bold, black, monospace font. Below this, there are three input fields with labels to their left: 'User Id:' followed by a text input containing 'palwetejas08@gmail.c', 'Mobile Number: +91-' followed by a text input containing '9987580640', and 'Password:' followed by a text input containing '9987580640'. Below the input fields, there are two buttons: 'Submit Data' (blue with white text) and 'Close' (red with white text).

Registration page

Login Page

User Id:

Mobile Number: +91-

Password + OTP:

Message:

Login Page

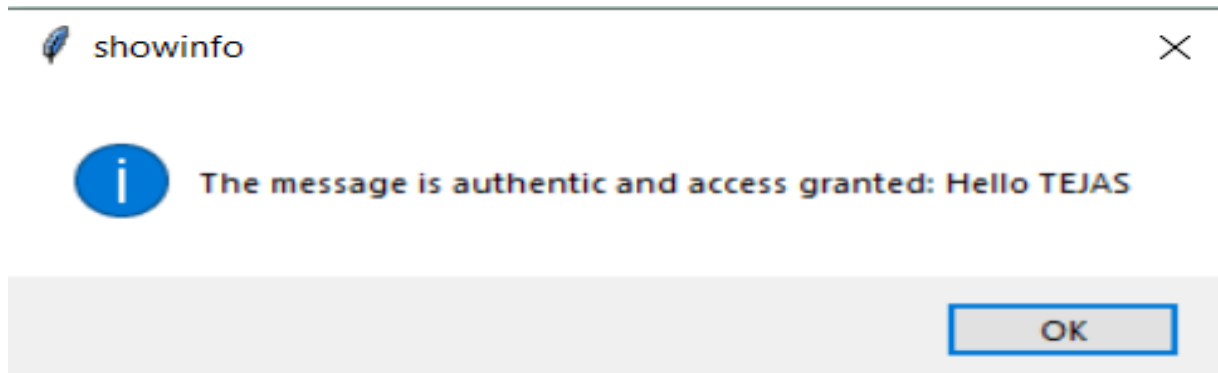
Decrypter Message

Decrypter

Password + OTP:

Crypto OTP:

Decryption Message



### Decrypted Message

```
===== RESTART: D:\project\gui_panel.py =====
9987580640372188
('palwetejas08@gmail.com', '9987580640', '9987580640', 9987580640372188, 'f8j\\', b'\\xf8j\\xc92\\xe9\\xeeD\\x1d0\\x91b\\xdc\\xf8', b'\\x8c\\xd1\\xb7n\\x94b<B5\\x1d\\xd6\\x9a\\xcf0\\x891', b'?To4z$\\xe0\\x90]\\xe5\\xea\\xe7\\x86\\x07\\x060')
b'9987580640372188'
b'Hello TEJAS'
('palwetejas08@gmail.com', '9987580640', '9987580640', 9987580640372188, 'x1d\\', b'V\\x1d\\x07\\xe8\\xab#zrp8S', b'_\\xd9\\x93\\x12\\xdd\\xd0\\xc1\\x1a\\x1e\\r\\xda0\\xfa\\xea\\xf8n', b'P\\xca;i\\xf5\\x92u\\xe3B\\xc6n\\x8f\\xf7G\\xbf\\xef')
9987580640 x1d\\
```

### Encrypted Message

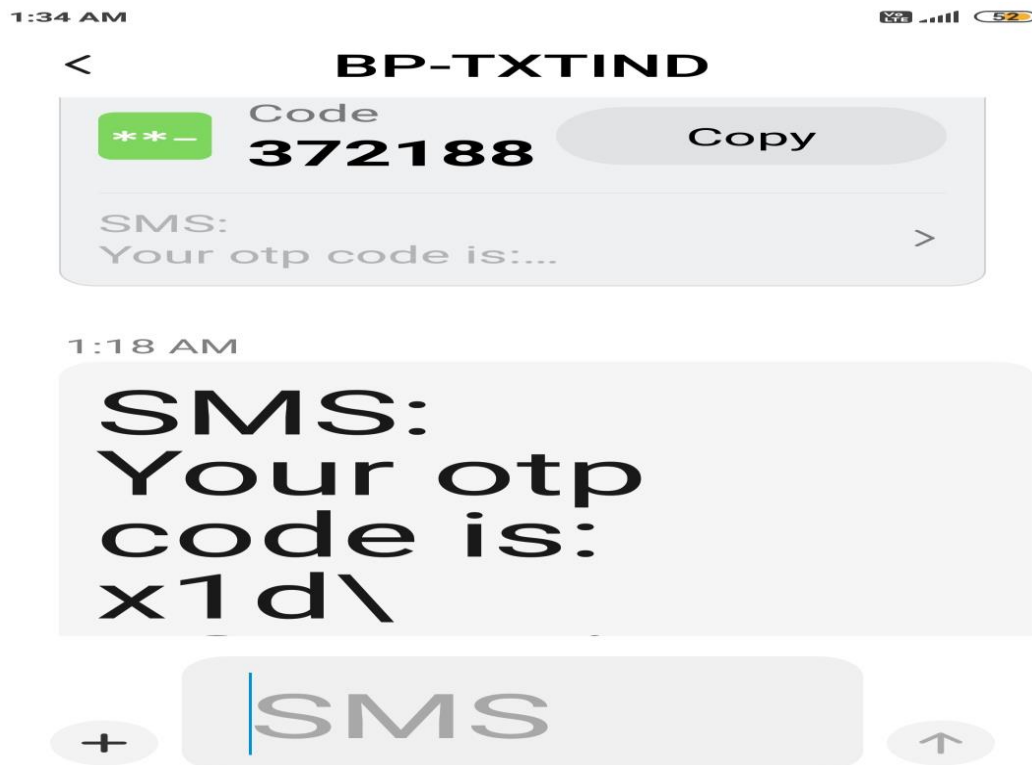
===== RESTART: D:\project\db\_testing.py =====

```

'sqlite3.Cursor object at 0x03351f20>
'shoebahmed370@gmail.com', '7303764284', '1234567890', 1234567890968891, None, b'\xc0\x86jD\xbf@xe1.', b'\xeb.j\x8b,\x87t\x8b9\x1e\x8e<w\\xc9\x0f't', b'\x8fD\x9\xdc\x2\xab\x97\ng\xfb\t\xe6\x16\x11\x87\x8e
)
'shoebahmed370@gmail.com', '7303764284', '1234567890', 1234567890968891, None, b'\xc0\x86jD\xbf@xe1.', b'\xeb.j\x8b,\x87t\x8b9\x1e\x8e<w\\xc9\x0f't', b'\x8fD\x9\xdc\x2\xab\x97\ng\xfb\t\xe6\x16\x11\x87\x8e
)
'pathan', '1234567890', '1234567890', 829683, None, None, None, None)
'musheer', '1234567890', '1234567890', None, None, None, None, None)
'mohammad', '234567890', '23456789', None, None, None, None, None)
'palwetejas@gmail.com', '8308698145', '8308698145', 8308698145256370, None, b'\x88\xd4#m\xa8\xe8j', b'\xcd\x03\x9d\\xbe\xee>\x99/\x9969\xa9\x02\xe1', b'\x03\xa8<\x15c\x9b\xfb\xec\x1b\xe3\x1d\x0e\x16\x98\x0
y')
'rafaq@gmail.com', '8007936509', '8007936509', 8007936509639683, '0b\\x', b'\x0b\xc3\x1c\xe1I(TW\xfbH', b'\x8d?\xe1x,\xe5j\x8d\x93[\x19\xdc6\\'T', b'\xb8+\xea\x1a\x10t\xed+\x04\x8b\x112\x90+js')
'shoeb@gmail.com', '8208071563', '8208071563', 8208071563201202, '83\\x', b'\x83\xe4P\x1e\xc3\xcb\x94)s\x0', b'\xe7\xc3\x84\x8b0H_-x0e/\t\x83\xa5\xb3y\xbd\xa2', b'\x80\xb8\x09\xcf\x95+(=x\xac\x8b5uOq')
'aayati@gmail.com', '9321477777', '9321477777', 9321477777122962, 'ee\\x', b"\xee\cx5\xe2\x8e\x8b'BA\xe4\xab\x0c\x05", b'\xf9t9\xbb(\x93T\x8d7?c2k', b'\xcc\x95\xaf\x87\xfd\xac\x84\x1a;\xcd\xbc\x90)$\x
0j')
'palwetejas08@gmail.com', '9987580640', '9987580640', 9987580640372188, '1d\\', b'v\x1d\x07\xe8\xab#zrpK8', b'\_xd9\x93\x12\xdd\x0c\x1a\x1e\r\xda0\xfa\xea\xf8K', b'P\xca;i\x5\x92u\xe3B\x8f\x8f\x7G\xbf
xef')
>>

```

## Backend Database



## OTP Message



## 5. References

1. SMS Authentication Code Generated by Advance Encryption Standard (AES) 256 bits Modification Algorithm and One Time Password (OTP) to Activate New Applicant Account .
2. A Comparative Analysis of AES Common Modes of Operation.
3. Advanced Encryption Standard (AES) Security Enhancement using Hybrid Approach .
4. Image Steganography Based on AES Algorithm with Huffman Coding for Compression on Grey Images
5. AES page available via <http://www.nist.gov/CryptoToolkit>

## 6. Conclusion

The conclusion is the use of authentication code to activate new accounts to minimize the account creation of rogue or fake accounts, in which the authentication code is derived from the activation message and the encrypted value timestamp for OTP by using a cryptographic algorithm