

PROJECT TITLE:

DATA ENCRYPTION AND TRANSFER USING DMS AND DSP

[21AIE204 & INTRODUCTION TO COMMUNICATION SYSTEM]

DONE BY -

[TEAM-11]:

SABARINATH J [21141]

SHYAM GANESH [21149]

SIDESH SUNDAR [21150]

SASANK SAMI [21160]

BHARADWAJ V [21165]

**ABSTRACT**

1. The internet, a necessity for modern existence, is among the everyday needs of 60% of the world's population. Because hackers can utilize this data, difficulties with data security and privacy arise when utilizing the internet. In light of this, we in this essay suggested the advanced custom adjustable algorithm for AES. To prevent hackers from encrypting a communication, an additional layer of security is added to each letter of the message. In order to decrypt and eliminate the vulnerability to frequent attacks, we are introducing a new layer of encryption that is already established by the method and creating a key via clock synchronization. This new layer safeguards the AES algorithm, which is already more secure. In today's digital environment, digital encryption is essential for safeguarding electronic data flows that are later processed by DSP. These documents include those pertaining to health, finance, the law, automatic and online banking, and more. To satisfy these requirements, electrical data encryption using the Advanced Encryption Standard (AES) can be used. Despite the fact that no substantial AES attacks have been discovered yet.

**OBJECTIVE**

To encrypt the data using the custom AES which is an open source adaptable, strong and secure encryption technique which can stop numerous threats and is extremely reliable to ensure safe data transfer, this can further be extended into message signal transferring via applying Digital Modulation Techniques which ensures unreadable and secured data transfer via signal.

**LITERATURE STUDY**

AES was developed in 1997 by the National Institute of Standards and Technology (NIST). It was developed to replace DES, which was inefficient and vulnerable to numerous assaults. The shortcomings of DES were addressed by the creation of a new encryption algorithm. Following that, AES was made available on November 26, 2001.

According to B.Nageswara Rao et al. (2017) in their article "Design of Modified AES Algorithm for Data Security," increasing the number of rounds (cycles) from 10 to 16 makes the algorithm (AES) more secure. More computing power will be required as the number of cycles increases, making it harder for hackers to compromise the system. The polybius square method is used to create the key.

Ako Muhammad Abdullah used keys that were 128 bits, 192 bits, and 256 bits in block cipher to implement 10 rounds of AES encryption in their 2017 paper titled "AES Algorithm to Encrypt and Decrypt Data." His investigation reveals that AES is more secure than rival algorithms like DES and 3DES.

In their paper "Implementation of Area Efficient 128-bit Based AES Algorithm in FPGA," N Sivasankari et al. (2017) report that both encryption and decryption have been actualized into a single chip (FPGA-XC5VLX50T), and that they perform with minimal resource utilization and a high throughput of 38.65Gbps.

In their article "Encryption and Decryption - Data Security for Cloud Computing - Using AES Algorithm," Talari Bhanu Teja et al. (2017) adopted that both RSA and AES technique are blended for encryption handle using USB device to upload and download data. Downloads and uploads of files are securely preserved. The system's benefit is that it elevates cloud storage frameworks' security by providing them with a spine structure. One drawback of the suggested method is that it only works with text files, not with other kinds of data like photos. Downloads and uploads of files are securely preserved. The system's benefit is that it provides a spine structure for cloud storage frameworks to increase security. The system's weakness is that it only supports text files, not other sorts of data like images, audio, or videos.

Today's digital technologies regularly transport and store sensitive data, thus encryption is commonly employed. AES, the industry standard encryption technique, is widely employed to protect data. When constructing VLSI systems, it can be challenging to strike a balance between space, power, and speed, and hardware encryption is no exception. System needs bring up specific performance factors, but it is not always obvious how to adjust design implementations to meet performance requirements. Despite the fact that several sources in this field of study noticed and examined intriguing AES algorithm features and their consequences on some of the design trade spaces, there was a scarcity of a single comparison analysis.

Key size, mode specificity, round key storage, round unwinding, SBOX implementation, and pipelining are the six AES components that are covered in this research. Readers may quickly analyze how each of the six characteristics affects speed, power, area, latency, and throughput by glancing at a compressed image of the resulting designs.

**METHODOLOGY**

# 

Encryption: The program is started by the sender who wants to transmit a text message to the recipient, a different user. The plaintext is then encrypted after being entered into a textbox and after clicking the "encrypt" button to create an intermediate cipher text. In this encryption, the immediate time serves as the key. Depending on whether the string is odd or even in length, the instantaneous time is placed at a particular index in the cipher text.

Only 50% of the plaintext has been converted to cipher text as of this time. The final stage of encryption is AES encryption, which uses the standard AES algorithm to convert the intermediate cipher text into a complete cipher text.

At the receiver's end, the cipher text must first be converted to plain text before the decryption process can begin. In this stage, the decryption process' unlocking is reversed. Caesar cipher decryption was used to unlock the encrypted data after the cipher text was first unlocked using AES decryption, which revealed the key solely in the cipher text. The sender's original plaintext will be given to the receiver after the decryption procedure is complete.

Caesar cipher is never mentioned, and thus prevents outsiders from figuring out the steps this method takes, which is a key element in its effectiveness. They will have two choices presented to them through a user-facing program:

ENCRYPTION – This will encrypt the plaintext using a key to produce the cipher text.

DECRYPTION - This process reveals the plaintext by decrypting the cipher text using the key.

To insert plaintext for encryption, use this textbox. The plaintext is initially encrypted using the unique Caesar cipher encryption algorithm when the user clicks the "encrypt" button, and then it is encrypted using the AES technique. The user can also alter it to increase the security of the ciphertext. Select the desired plaintext from the drop-down menu, then click "SHOW" to display the encrypted cipher text in its unencrypted version. The user completes this task at the Sender end.

Decryption: In this situation, the user will be able to convert the cipher text into plaintext using a customized Caesar Cipher decryption configuration and the AES method in reverse. This is carried out in the application on the receiver's end.

The AES algorithm's code is written using the JAVA programming language to achieve this. After being encrypted, the plaintext is saved in the database. The decryption procedure follows the same steps, but in the opposite direction. The cipher text is removed from the database and converted back to plaintext. The primary end users of this application will be the military and other top-secret government intelligence agencies. Basically, any business that wants to exchange messages with high security can use this approach.

**CODE:**

# AES Algorithm

## **Configuration**

Choose Key Standard:

keyProperties = 128

config = Config();

if keyProperties == 128

config.Nk = 4;

config.Nb = 4;

config.Nr = 10;

elseif keyProperties == 192

config.Nk = 6;

config.Nb = 4;

config.Nr = 12;

elseif keyProperties == 256

config.Nk = 8;

config.Nb = 4;

config.Nr = 14;

end

SBox configuration:

sBoxConfig = "Test"

if sBoxConfig == "Test"

config.sBox = TestSBox();

config.invSBox = TestInvSBox();

elseif sBoxConfig == "Random"

% To change the random configuration, change the variables

% in SBox.m and InvSBox.m files

config.sBox = SBox();

config.invSBox = InvSBox();

end

Enter the plaintext and cipherkey:

plainText = "sasank"

cipherKey = "SpaceX"

textInts = uint8(char(plainText));

% Test for AES-128

% textInts = uint8([0x32 0x43 0xf6 0xa8 0x88 0x5a 0x30 0x8d 0x31 0x31 0x98 0xa2 0xe0 0x37 0x07 0x34]);

% Test for AES-192

% textInts = uint8([0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0xff]);

% Test for AES-256

% textInts = uint8([0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0xff]);

nblocks = int32(length(textInts)) / (config.Nb \* 4);

npad = uint8(config.Nb \* 4 ...

- (length(textInts) - config.Nb \* 4 \* nblocks));

textInts = [textInts repmat(npad, 1, npad)]

keyInts = uint8(char(cipherKey));

% Test for AES-128

% keyInts = uint8([0x2b 0x7e 0x15 0x16 0x28 0xae 0xd2 0xa6 0xab 0xf7 0x15 0x88 0x09 0xcf 0x4f 0x3c]);

% Test for AES-192

% keyInts = uint8([0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f ...

% 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17]);

% Test for AES-256

% keyInts = uint8([0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f ...

% 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f]);

npad = uint8(config.Nk\*4 - length(keyInts));

keyInts = [keyInts repmat(npad, 1, npad)]

config.K = keyInts;

## **Encryption**

cipherText = [];

for i = 1:(config.Nb \* 4):length(textInts)

fprintf("\n\n-----------------------------------------------------------------------\n");

fprintf("Process for blob %i:", int32(i/(config.Nb \* 4) + 1));

fprintf("\n-----------------------------------------------------------------------\n");

blob = Cipher( ...

textInts(i:i+config.Nb \* 4-1), ...

config.K, ...

config.Nk, ...

config.Nb, ...

config.Nr ...

);

cipherText = [cipherText blob];

end

display(cipherText);

## **Decryption**

decryptedInts = [];

for i = 1:(config.Nb \* 4):length(textInts)

fprintf("\n\n-----------------------------------------------------------------------\n");

fprintf("Process for blob %i:", int32(i/(config.Nb \* 4) + 1));

fprintf("\n-----------------------------------------------------------------------\n");

decryptedBlob = InvCipher( ...

cipherText(i:i+config.Nb \* 4-1), ...

config.K, ...

config.Nk, ...

config.Nb, ...

config.Nr ...

);

decryptedInts = [decryptedInts decryptedBlob];

end

decryptedInts = decryptedInts(:);

if range(decryptedInts(end-decryptedInts(end)+1:end)) == 0

fprintf("Decryption successful!\nPlain text: %s", ...

char(decryptedInts(1:end-decryptedInts(end))));

else

fprintf("Decryption failed :(");

end

**CODE EXPLANTION:**

First we have select the key standard.

Each key properties has its own matrix sizes i.e. for 128 4x4,192 6x4,256 8x4.

Now set the input type for the data.

If the type is test a testbox will appear in which we have to give the input , if it option is selected as random the computer will generate the text.

We also have to initialize the chiperkey which will help us to encrypt the data in more safer way.

Now the data and cipher key will be converted to integer.

In the next step we will create the process blob in that total of 13 rounds will take place to encrypt the given data.

After completion of the process blobs the encrypted data will be provided in matrix form.

The encrypted data will be given as input to the decrypter which will debug the process blobs and give us the input data.

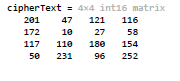
If the decryption succeeds the decryptor will give the data which we gave if the decryption fails decryptor will show us the ouyput as decryption failed.

**OUTPUT:**

**INPUT:**

**“sasank”**

**ENCRYPTED DATA:**



**DECRYPTED DATA:**



**INFERENCE:**

The core of the work is the extended AES algorithm with custom configuration, a novel concept. When text is encrypted using a programmable algorithm, the user has the ability to change the algorithm every time without even realizing it. The method employs AES and includes certain system-customizable phases. It is common knowledge that the world is shifting more and more toward digital systems, and everywhere has generally good internet accessibility. The AES configuration is widely used to convey classified messages. For such governments as well as other organizations, this technology offers an extra layer of safety that is completely hidden from everyone, including the user. It cannot theoretically be broken without assistance from within.