Crypto-steganographic Hybrid Blockchain Model for Network Security

B.Tech. Project Report

By

Name of the B.Tech. Students
Sandeep Shaw
Akash Kumar Sen
Surajit Bera
Arijit Mukherjee

Under Supervision of
Dr. Bijoy Kumar Mondal

Department of Computer Sc. and Engineering

Government College of Engineering and Ceramic Technology
Kolkata

November 2022

Crypto-Steganographic Hybrid Blockchain Model for Network Security

A Project Report

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Sc. and Engineering

By

NAME OF THE STUDENTS (with University Roll Number)
Sandeep Shaw(GCECTB-R19-3022)
Akash Kumar Sen(GCECTB-R19-3002)
Surajit Bera(GCECTB-R19-3036)
Arijit Mukherjee(GCECTB-R19-3007)

Department of Computer Sc. and Engineering

Government College of Engineering and Ceramic Technology
Kolkata

November 2022

Name and Roll No. of the Students	Signature of the Students
1. Sandeep Shaw(GCECTB-R19-3022)	……………………………
2.Akash Kumar Sen(GCECTB-R19-3002)	…………………………….
3.Surajit Bera(GCECTB-R19-3036)	…………………………….
4.Arijit Mukherjee(GCECTB-R19-3007)	…………………………….

Place:

Date:

Government College of Engineering and Ceramic Technology

73, A. C. Banerjee Lane, Kolkata, West Bengal 700010

BONAFIDE CERTIFICATE

Certified that this literature survey report titled Crypto-steganographic Hybrid Blockchain

Model for Network Security is the realistic work carried out by

1. **Sandeep Shaw(GCECTB-R19-3022)**
2. **Akash Kumar Sen(GCECTB-R19-3002)**
3. **Surajit Bera(GCECTB-R19-3036)**
4. **Arijit Mukherjee(GCECTB-R19-3007)**

who will carried out the project work under my / our supervision.

……………………………………… ……………………………………………

Dr. Bijoy Kumar Mondal
SUPERVISOR **JOINT SUPERVISOR**
Assistant Professor
Department of Computer Science and Engineering
Government College of Engineering and
Ceramic Technology
Kolkata-700010 **(if any)**

…………………………………… ……………………………………………

Dr. K. Saha Roy **External**
HEAD OF THE DEPARTMENT
Assistant Professor & Head
Department of Computer Science and Engineering
Government College of Engineering and Ceramic Technology,
Kolkata

Abstract

The purpose of this project work is mainly focused on developing a secure and efficient model for data transfer between client and server that is end-to-end encrypted and promotes data integrity. This model is demonstrated using a simple text messaging application which provides assurance to the user by securing their data. The idea here is to address two major data security threats that usually make any regular chat application vulnerable, the former is the Man in the Middle Attack (MITM Attack) and the latter is an incident of Data Breaching.

The Man in the Middle Attack is one of the most common forms of cyber-attack the goal of an attack is to steal personal information, such as login credentials, account details, and credit card numbers. In order to tackle this form of cyber vulnerability, we propose to design an algorithm that could easily be used to communicate between the client and server, and even if the data is tapped by any perpetrator it will take billions of years to crack down the information and is safe against any brute force attacks.

The other principal motive is to store the encrypted messages in the database server so that the admin who is authorized for such a centralized database can never view the text chat between the sender and the receiver. Even in the case of a Data Breach which is a security violation, in which sensitive, protected, or confidential data is copied, transmitted, viewed, stolen, or used by an individual unauthorized to do so or in other terms are unintentional information disclosure, data leak, information leakage, and data spill, the data is still encrypted and the information can't be decoded very easily.

Moreover, on the server-side if any unauthorized user made an attempt to modify the data then this activity can easily be identified as the text chat between the sender and receiver is stored in a block chain manner. Overall, this purpose idea promotes significant improvement over the existing concepts in terms of both security features and data integrity solutions related to the messaging chat application from the perspective of the client and server side.

Introduction

We are living in the modern era of computers and internet where most of our activity is digitalized and is performed online. The information on every aspect of our lives are being uploaded and transferred through internet. In a more generalized way internet is a form of computer network where multiple digital transaction occur within splits of seconds as our data is uploaded and transferred through the computer networks. Moreover, it is also important to secure the privacy as confidential online transaction or information might be accessed or manipulated by the attackers or any other third party attacks. Therefore, to protect the data, the role of network security comes into action and to prioritize safety of confidential information across the network.

In this project, we have proposed to build a secure and efficient model for data transmission between client and server. This model will be illustrated using a text messaging application using Hashing, Cryptography, Steganography with some concept of Block chain which will be used to store and retrieve the message data in a more secure manner.

In this model we have tried to address some major data security threats that usually make any regular chat application vulnerable are:

- Man in the Middle Attack (MITM)
- Data Breaching
- Brute-Force Attack

Following are the fields that we are going to make use of in order to implement our proposed idea:

Computer Network:
Computer network is a system that helps us to connect and communicate a large number of self-dependent computers in the intention of sharing information and resources or in simpler words when two or more computers are connected together, the entire system of two or more connected computers is called a computer networks. Today's user-centric applications are built on the most popular and widely used form of a computer network called internet.

Network Security:
In order to make a computer network trustworthy and reliable we have to go through three security measures confidentiality, integrity and availability which forms the basis of Network Security. The measures taken by the application developers and the computer scientists to make confirm these three actions are known as network security.

Attacks:
The three goals of security, confidentiality, integrity and availability can be threatened by some third party unethical measures. Such phenomenon are known as attacks. Attacks can take place in any from like snooping, traffic analysis, unauthorized modification, unauthorized access, data breaching etc. But throughout the project we have decided to create an algorithm that deals with two specific type of attacks, MITM or Man in the Middle Attack and Data Breaching. Also we have tried to modify an existing algorithm to prevent any instance of Brute Force Attack.

Man in the Middle Attack(MITM):
The MITM or Man in the Middle Attack is a general term of when a third person have view or modify access in a conversation between the server and the client. This breaches one of the most important aspects of network security that is confidentiality.

Data Breaching:
A data breach is a security violation, in which sensitive, protected or confidential data is copied, transmitted, viewed, stolen or used by an individual unauthorized to do so. Other terms for Data Breaching are unintentional information disclosure, data leak, information leakage and data spill.

Brute Force Attack:
In cryptography, a brute-force attack consists of an attacker systematically checks all possible passwords and passphrases until the correct one is found. This method of hacking uses trial and error to crack passwords, login credentials, and encryption keys. It is a simple yet reliable tactic for gaining unauthorized access to individual accounts and organizations' systems and networks.

Cryptography:
Cryptography is technique of securing information and communications through use of codes so that only those people for whom the information is intended can understand it

and process it. Thus preventing unauthorized access to information. The prefix “crypt” means “hidden” and suffix “graphy” means “writing”.

In Cryptography the techniques which are used to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions. In our project we have tried to cipher a text message using cryptographic hash algorithms.

Hashing:
Unlike Cryptography where the cipher text can be encoded and decoded, hashing algorithms are one-way programs which utilize a mathematical function to unscramble the text which can't be further decoded. This enciphered text can then be stored instead of the password itself, and later used to verify the user.

Features of Hashing Algorithm

Takes an arbitrary amount of data input and produces a fixed-size output of enciphered text called a hash value.
Minute change in input data should produce vast change in hash value.
In this project we used SHA-512 to obtain a hash value from arbitary length message which is used as a key for cryptography.

Steganography:
Steganography is the technique of hiding secret data within an ordinary, non-secret, file or message in order to avoid detection; the secret data is then extracted at its destination. The use of steganography can be combined with encryption as an extra step for hiding or protecting data. In this project we have tried to use a PNG file as the pseudo non-secret file for hiding the data.

Blockchain:
A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The

innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

We have tried to implement a hybrid model of blockchain technology by implementing some key concept of Blockchain in existing databases.

Literature Survey

In this section, we provide a brief literature review on recent crypto-steganography and blockchain techniques with a focus on secure data encryption, effective steganography methods, and block-chained data integrity.

Data encryption and Steganography are one of the most widely used techniques which are used to protect data from cyber threats and cybercriminals. Moreover, these techniques of data protection have been proposed by many studies to effectively tackle the problems. The main challenge that was encountered was to generate secure and dynamic encryption and embed this data inside the image effectively. Since LSB steganography is one of the most effective ways to easily embed data to create a stego-object. However, this technique comes with a drawback as it is very easy to crack by just obtaining all the least significant bits [1].

Moreover, the length of the message may vary, and therefore if the message is small enough then it would be unnecessary to embed it in a large file on the other hand if the encrypted message is very large then it might not be able to fit completely inside the file. Therefore, the main challenge that was faced while designing the solution to this problem was to implement a dynamic image object that will be generated automatically based on the encrypted message unlike providing a fixed or static image. However, to implement such a dynamic stego-object, we need to design a specific design format that could fetch the data appropriately. Also, we need to provide a methodology to detect single-bit errors and to provide more variation to the existing LSB steganography algorithm to prevent easy retrieval of data.

Apart from Dynamic Image Steganography, the main aim is to provide an encryption algorithm that is secure and could generate a key to decipher the text message. After going through various cryptography concepts [2] the best alternative was to apply Symmetric Key Encryption. The reason to follow this mode of encryption is that in our text messaging applications any amount of data may be passed. Secondly, since encryption and decryption are end-to-end encrypted, therefore, the resource utilization of the algorithm should be less. Hence symmetric encryption is the preferred choice as the encryption process is fast, used when a large amount of data is required to transfer, and resource utilization is low as compared to asymmetric key encryption.

Examples of various Symmetric key encryption are:

- Blowfish
- AES (Advanced Encryption Standard)
- RC4 (Rivest Cipher 4)
- DES (Data Encryption Standard)
- RC5 (Rivest Cipher 5)
- RC6 (Rivest Cipher 6)

The most commonly used symmetric key algorithms are Blowfish, AES-128, AES192, and AES-256. Symmetric key encryption is the encryption method where we use only one key for encrypting and decrypting the data. Out of various symmetric key encryption, the most reliable is the Advanced Encryption Standard (AES) algorithm. AES algorithm is far superior and more secure as compared to Data Encryption Standard (DES). Moreover, AES is Byte-Oriented and the key length and the number of rounds to decode or encode the cipher text varies depending on the AES variants like 128, 192, or 256.

AES consists of 2 important processes: Key Scheduling and Encryption/Decryption Process. In Key Scheduling, we use operations like Rot Word and Byte Substitution for key expansion and generate several round keys from the given symmetric key. The Encryption process includes Byte Substitution from S-box, Shifts Rows, Mix Columns, and adding a Round Key generated from Key Scheduling. Moreover, the Decryption Process also follows the same process but in reverse order consisting of Inverse Byte Substitution, Inverse Shift Rows, Inverse Mix Column, and adding a Round Key which is the core implementation of AES. Moreover, the current AES algorithm uses a square matrix of order 4 which generates a 128-bit cipher text and cipher key.

As of now, the AES algorithm is unbreakable, however, potential problems exist in AES-128 and AES-256 in recent times, the new process of attacks is combined for boomerang and rectangle attacks. This uses the weaknesses of a few nonlinear transformations in the key schedule algorithm of ciphers and it can break some reduced-round versions of AES which is its disadvantage [3][4]. In order to overcome this problem, 512 bits symmetric key of the algorithm can be used to increase the robustness by keeping the processing time low [5] where we use a square matrix of order 8 to generate the round keys and cipher text which is 512 bits.

After providing data security which is to be delivered from the client to the server, the next responsibility of the text message application is to design an effective data structure that can not only provide easy CRUD operations but also check the integrity of the data whether or not it has been manipulated by adding, removing or updating the database information without user's consent. In order to implement such a feature, the chat data is stored in a hybrid blockchain manner where the hash of the previous message is stored in the current message. It helps in the verification and traceability of multistep transactions needing verification and traceability. It can provide secure transactions, reduce compliance costs, and speed up data transfer processing. After the client API ask for the encrypted chat data, the server will process the blockchain technology by calculating the data integrity of each chat one by one.

The main disadvantage of using blockchain technology is that as the size of the data increases the time complexity to verify every transaction also increases. Therefore, in our chat application as the number of user increase, the amount of data also increases linearly so in order to accommodate this blockchain technology, a relational database SQL or NO-SQL database can be utilized where data is stored in a specific format but with the scope to implement a certain functionality of blockchain. Overall the method of storing data is hybrid in nature with the implementation of core blockchain concepts with underlying SQL, Graph-based, or No-SQL database frameworks [6]. Since speed is the factor that limits the use of pure blockchain concepts in these applications it is also an essential feature apart from security. Therefore, the emergence of creating a hybrid model to store data is the feasible option that balances both data integrity by blockchain and speed by using the traditional database approach.

Proposed Method

The principle idea is to construct a model where at various phases of data transmission from client to server and vice-versa, a specific algorithm is utilized to secure the data. This algorithm includes hashing, dynamic crypto-steganography, and the use of hybrid blockchain database storage. Thus this model of various algorithms is used to implement a text messaging app system that provides a secure data transfer from the client to the server and stores that particular data inside the database such that the integrity of data is conserved.

The methodology in which the design and analysis of the solution proposed are:

Constant Key Storing

First of all, the users of the chatroom will share a common and consistent secret code generated randomly when the chatroom is established. Depending on the nature of security these secret code may be stored either in user's device in decentralized manner or inside more secure layered database in centralized manner. Moreover, the unique-id of the chatroom can also be used as the secret code or any randomly generated characters that will remained be constant.

Hashing and Dynamic Encryption

Once the connection is established and secret key is generated for the users, we will move on to the next step. The secret key combined with other fields of the message will be used to generate a hash value using Secure Hash Algorithm (SHA-512).

Then this hash value generated above is used as a symmetric key for the Advance Encryption Standard algorithm (AES) to encrypt the message. Moreover, we will analyze various AES algorithm like AES-128, AES-192 and AES-256 which belong to the family of same 128 bits cipher key. Further we will compare it with AES-512 algorithm which we try to implement using a matrix of order 8, which will generate 512 bits cipher key.

Image Steganography

Once the data is encrypted, it is passed as input to generate an image using Steganography algorithms. We will try to analyze various steganography algorithm which suites best of this application and compare it with traditional Least Significant

Bit Steganography (LSB). According to the length of the encrypted message, a particular image of given dimensions will be calculated and used for the effective embedding of data inside the image. Then the Stego-object is sent to the server via API. Moreover, this type of image steganography will generate dynamic images based on the encrypted message that can be accommodated very easily and effectively. Moreover it should also provide room for additional bits to check the occurrence of errors such as single bit error.

Storing Data to Server

At the server, the message is extracted from Stego-object which outputs the encrypted message. This data is fetched and stored inside the database. While storing the encrypted message in a database, the traditional database will also perform and additional function by linking the hash of the previous message in the same chat room with the current message id using any Secure Hashing Algorithm. In this way the message in the chat is linked in a manner similar to blockchain. The functionality is provided to the user to check or authenticate the integrity in their data. This feature will mine the existing data up to desired period and compare the linked hash value and if the data is removed or manipulated then it will be easily recognized by the user. The database admin can neither view the message nor it can be manipulated. Moreover in case of data breach, it is still encrypted and also the message in the chat room is secured.

Fetching Data from Server

The chat data can be retrieved in bulk from the server to the client. As the data was stored in encrypted format therefore it is still secure and at the client side the chat will be decrypted. The symmetric key can be obtained using the constant secret key and other attributes of the message which is passed to the AES and the plain text is obtained from the cipher text and is rendered on the screen.

In this manner, the data is encrypted and decrypted on the client side only therefore it is end-to-end encrypted as the encrypted message is stored on the remote database with a hybrid blockchain functionality to authenticate and check tampering in data if it exists.

Algorithms

1. Secure Hash Algorithms

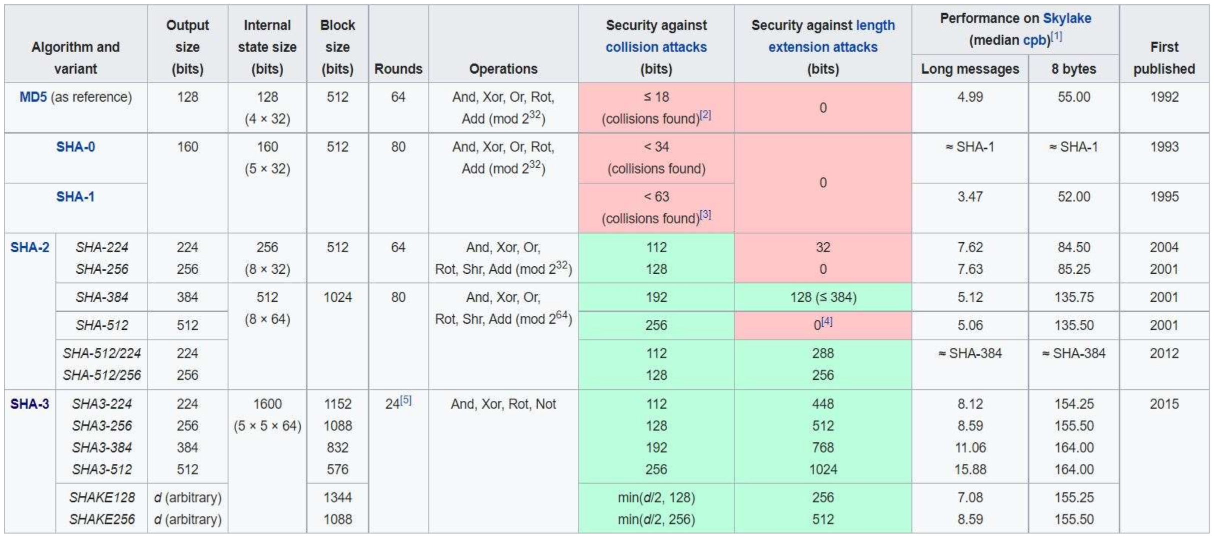
The Secure Hash Algorithms are a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS), including:

1. **SHA-0:** A 160-bit hash function
2. **SHA-1:** A 160-bit hash function that shares characteristics similar to the MD5 hash function
3. **SHA-2:** It consists of two hash functions with different block sizes SHA-256 and SHA-512 and additionally, there are four truncated versions.
4. **SHA-3:** It is formerly known as Keccak as it supports the same hash lengths as SHA-2, and its internal structure differs significantly from the rest of the SHA family.

1.1. SHA3-512

In this model, we have used SHA-3 as a principal hashing algorithm. It is because SHA-0 and SHA-1 is no longer used in real-world applications due to security vulnerabilities. SHA-3 is considered a more secure option than SHA-2 as its internal structure differs significantly from the rest of the SHA family.

The comparison among all families of SHA in terms of security:



Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security against collision attacks (bits)	Security against length extension attacks (bits)	Performance on Skylake (median cpb)[1]		First published
									Long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Or, Rot, Add (mod 2^{32})	≤ 18 (collisions found)[2]	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Or, Rot, Add (mod 2^{32})	< 34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1							< 63 (collisions found)[3]		3.47	52.00	1995
SHA-2	*SHA-224*	224	256 (8 × 32)	512	64	And, Xor, Or, Rot, Shr, Add (mod 2^{32})	112	32	7.62	84.50	2004
	SHA-256	256					128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Or, Rot, Shr, Add (mod 2^{64})	192	128 (≤ 384)	5.12	135.75	2001
	SHA-512	512					256	0[4]	5.06	135.50	2001
	SHA-512/224	224					112	288	≈ SHA-384	≈ SHA-384	2012
	SHA-512/256	256					128	256			
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	24[5]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088			128	512	8.59	155.50	
	SHA3-384	384		832			192	768	11.06	164.00	
	SHA3-512	512		576			256	1024	15.88	164.00	
	SHAKE128	*d* (arbitrary)		1344			min(*d*/2, 128)	256	7.08	155.25	
	SHAKE256	*d* (arbitrary)		1088			min(*d*/2, 256)	512	8.59	155.50	

https://en.wikipedia.org/wiki/Secure_Hash_Algorithms

The aim is to implement dynamic encryption using AES where the cipher key will vary according to the information passed. As AES-512 is designed and used to encrypt data which requires 512 bits of cipher key, therefore it is more suitable to use SHA3-512 which generates 512 bits of hash that can be easily passed to the AES algorithm.

1.2. SHA3-512 Implementation

As SHA3-512 is capable to withstand collision attacks and length extension attacks [x] therefore it is far more secure and efficient. To implement this algorithm, we will be using an external module name **hashlib** in Python programming language. This module provides us with most of the algorithms belonging to the SHA family.

1. Checking the presence of SHA3-512

```
import hashlib



# Available algorithm

print(hashlib.algorithms_guaranteed)
```

```
# Output

{

‘blake2b’, ‘shake_256’, ‘sha512’, ‘sha3_224’,

‘sha384’, ‘sha3_512’, ‘sha3_256’, ‘sha3_384’,

‘md5’, ‘sha256’, ‘sha224’, ‘sha1’, ‘blake2s’,

‘shake_128’

}
```

We can observe that the algorithm **‘sha3_512’** exist.

2. Using SHA3-512

```
import sys

import hashlib



if sys.version_info < (3, 6):

   import sha3



str = "CSHB Model"



# create a sha3 hash object

hash_sha3_512 = hashlib.new("sha3_512", str.encode())



print("\nSHA3-512 Hash: \n", hash_sha3_512.hexdigest())
```

```
SHA3-512 Hash:
fe3a5c5d82571204dce93169a1d8bedf1857a986d71437ff73090b8f3930c1825275
ff565b935e2edc76212d9fb218710438770f1a939b43931b5d280e1e22ee
```

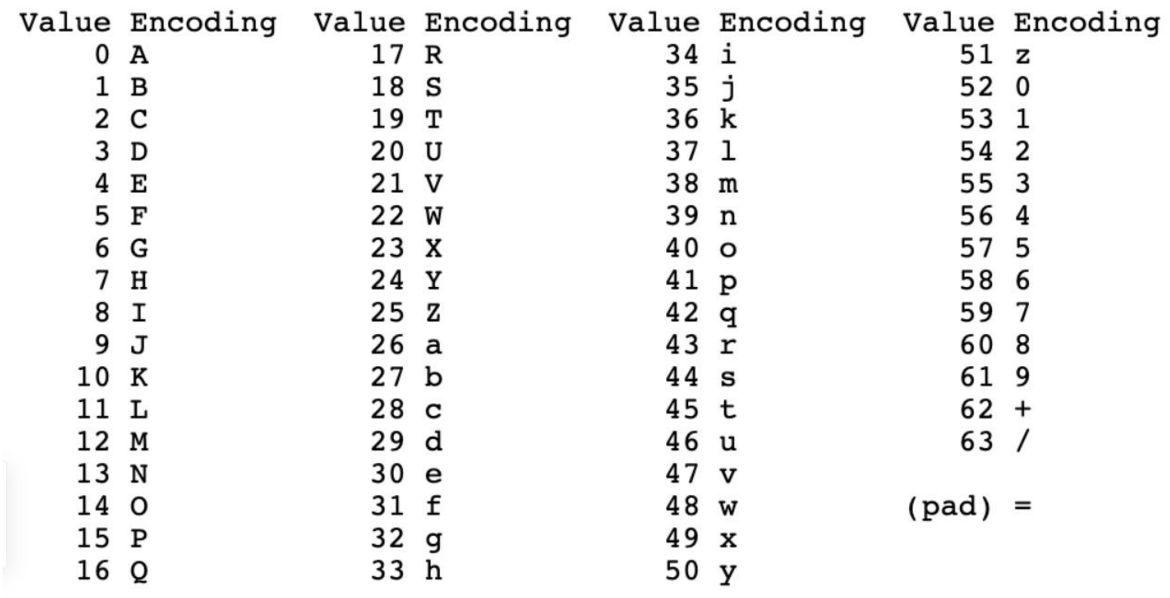
2. Base64

An encoding scheme that converts binary data into text format so that encoded textual data can be easily transported over the network without being corrupted or any data loss. The problem that arises while sending normal binary data is that bits can be misinterpreted by underlying protocols, and produce incorrect data at receiving node. The term Base64 is taken from the Multipurpose Internet Mail Extension (MIME) standard, which is widely used for HTTP and XML, and was originally developed for encoding email attachments for transmission.

Base64 encoding and decoding are mainly used in Advance Encryption Standard to represent the encrypted data in this format or to decrypt the message represented in this format. As the encryption and decryption algorithm will require changing base64 data to binary and vice-versa, therefore the use of base64 is considered.

2.1 Base64 Implementation

The implementation of base64 is simple where the process of converting binary data into a limited character set of 64 characters. The characters are A-Z, a-z, 0-9, +, and /. This character set is considered the most common character set, and is referred to as MIME's Base64. The 6 consecutive bits are converted to specific ASCII character and in case bits are not sufficient then it is padded.



Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

To implement this algorithm, we use a predefined module name **base64** in Python programming language.

```
from base64 import b64encode, b64decode

DATA = 'data to be encoded'

encoded = b64encode(DATA)
print('Encoded: ', encoded)

decode = base64.b64decode(encoded)
print('Decoded: ', decoded)
```

OUTPUT

```
Encoded: ZGF0YSB0byBiZSBlbmNvZGVk
Decoded: data to be encoded
```

3. Advance Encryption Standard

The Advanced Encryption Standard (AES) is a symmetric key encryption, also known as Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is a variant of the Rijndael block cipher which is a family of ciphers with different key and block sizes. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.

The three types of AES mainly used are:

1. AES-128
2. AES-192
3. AES-256

AES operates on a 4 × 4 column-major order array of 16 bytes b0, b1, b2, ... b15 termed the state:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

3.1 High-level Algorithm Description

This algorithm can be divided into 4 parts: Key Scheduling, Encryption Process, Decryption Process, and Padding. The encryption and decryption process follows a similar process but with different operations. The Key Scheduling algorithm is performed on the cipher key which is used both in the Encryption Process to encrypt the data and in the Decryption Process to decrypt the cipher data. However, the data which undergoes AES may or may not be in the multiple of 128 bits. Therefore in order to fit this data to a 4 × 4 column-major order array of 16 bytes, padding is used before encryption and it is removed after decryption.

3.1.1 Key Scheduling

It is a process of key expansion where a short key is expanded into a number of separate round keys that are used at different rounds in the encryption-decryption process.

The number of round keys generated for different variants of AES is

AES	Key Length	Round Key Generated	Total(Including initial key)
128	128b / 16B	10	11
192	192b / 24B	12	13
256	256b / 32B	14	15

The operations involved in Key Scheduling are:

1. **Rot Word**: It is defined as one byte left-circular shift

RotWord([b0, b1, b2, b3]) = [b1, b2, b3, b0]

2. **Sub Word**: The substitution of the four bytes of the word using AES S-Box.

SubWord([b0, b1, b2, b3]) = [S(b0), S(b1), S(b2), S(b3)]

3. **Round Constants**: The round constant ($rcon_i$) for round 'i' of the key expansion is the 32-bit word is given by:

$rcon_i = [\ rc_i, (00)_{16}, (00)_{16}, (00)_{16}\ ]$

The value of rc_i is an eight-bit value defined by

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11\text{B}_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

:

where $\oplus$ are the bitwise XOR operator and constants such as (00)16 and (11B)16 are given in hexadecimal.

The value of rc_i in hexadecimal up to i=10

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

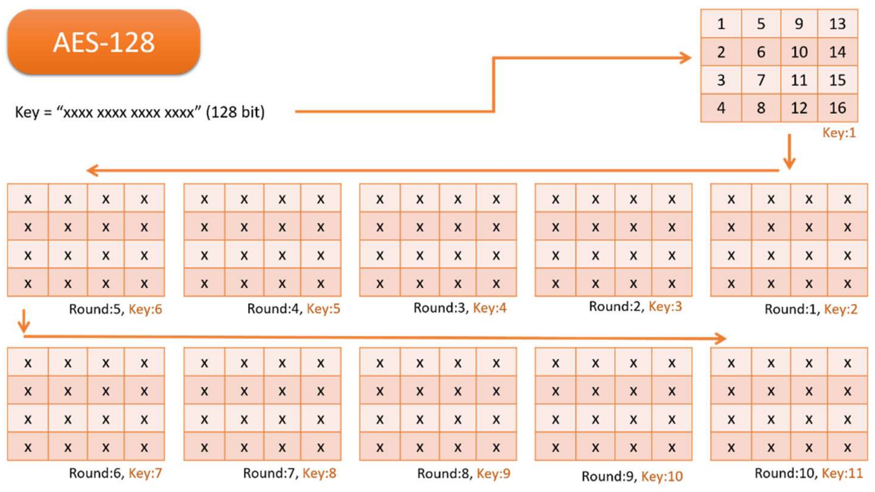
The round constant used by variants of AES are:

AES	Round Constant Used (i)
128	$rcon_{10}$
192	$rcon_8$
256	$rcon_7$

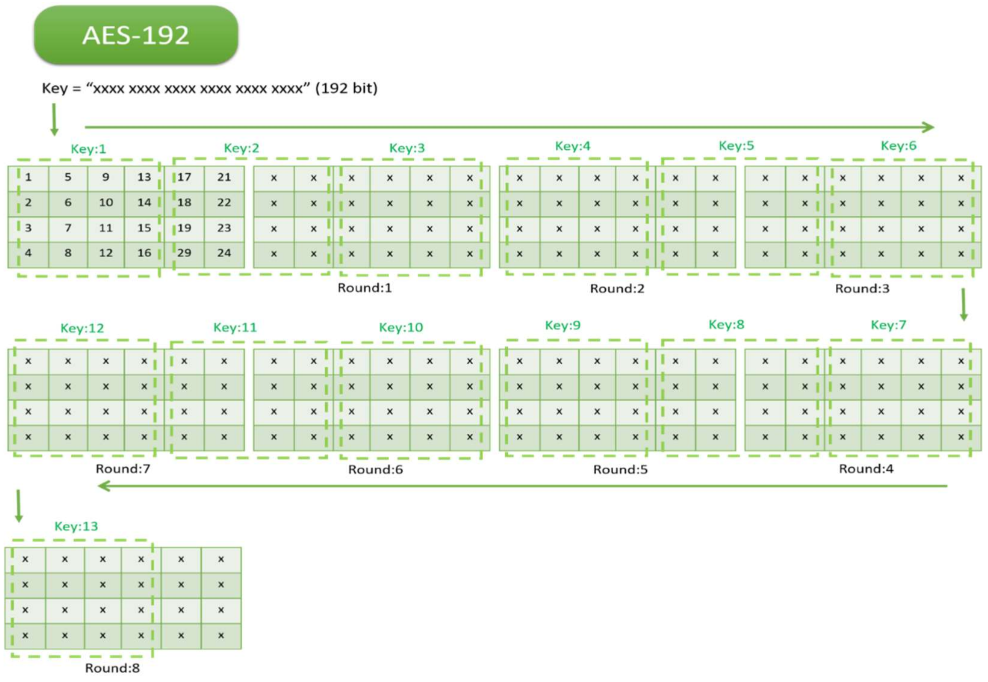
Key Expansion in Key Scheduling

- If we consider 32 bit / 4 bytes as 1 word then according to key length, there will be 4 words for AES-128, 6 words for AES-192, and 8 words for AES-256
- Let the words be represented by N
- Let K_0, K_1, ... K_{N-1} as the 32-bit words of the original key
- Let R as the number of total round keys needed.
- Let W_0, W_1, ... W_{4R-1} as the 32-bit words of the expanded key
- Then for "i" in range from 0 to 4*R - 1 the expanded value of W_i is calculated as

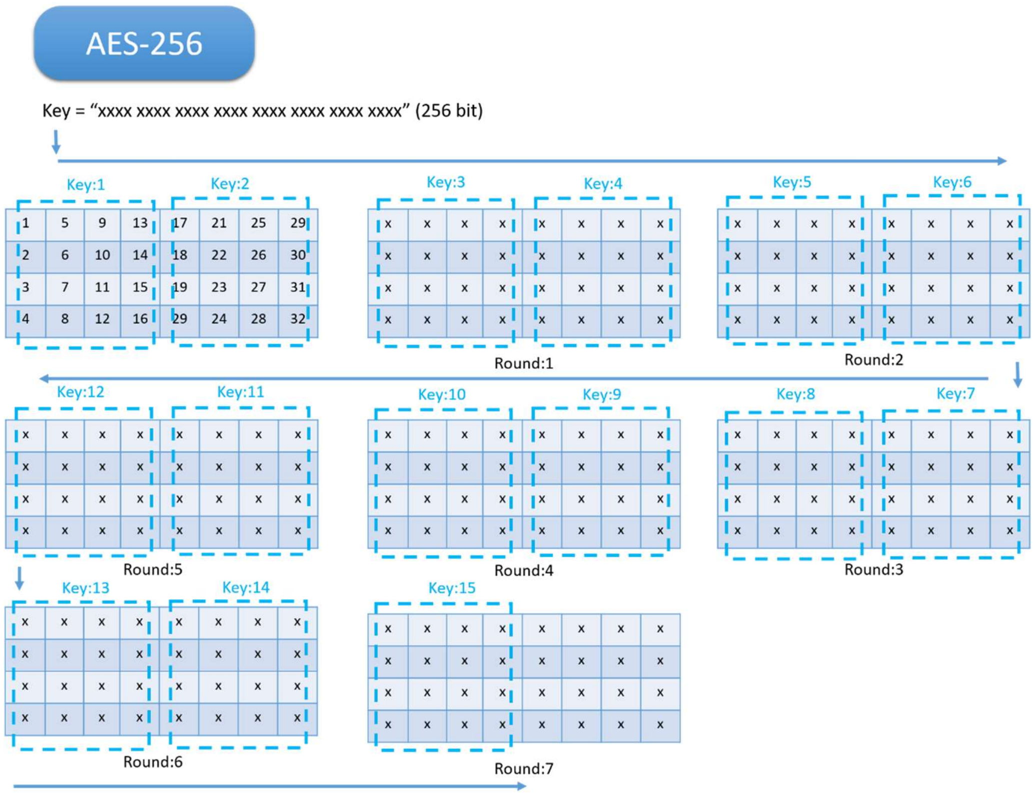
$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \mathrm{SubWord}(\mathrm{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \mathrm{SubWord}(W_{i-1}) & \text{if } i \geq N,\ N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$



Key expansion of AES-128



Key expansion of AES-192



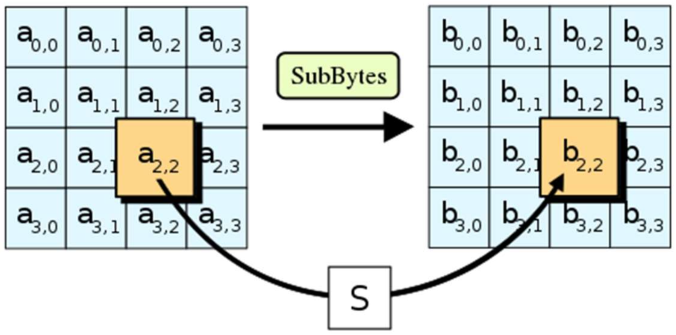
Key expansion of AES-256

3.1.2 Encryption

In this process, the plain text is divided into 128-bit text and for each division, encryption is performed. It uses the round keys generated in the Key Scheduling to obtain a cipher text at each round. The rounds consist of an initial round, an intermediate round, and the final round and it varies depending on the variants of AES.

The different operations involved are:

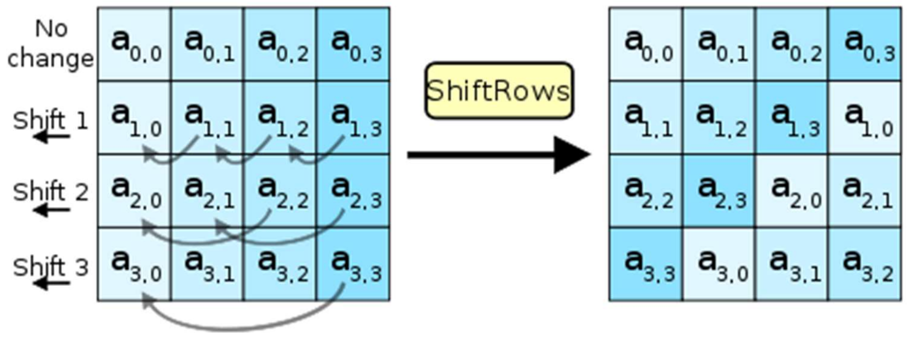
1. **SubBytes** It is the byte substitution process where the state array is substituted by the values in S-Box. This S-Box is used because it is derived from the multiplicative inverse over GF(2^8) which is known to have good non-linearity properties.



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

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

2. **ShiftRows:** The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively.



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

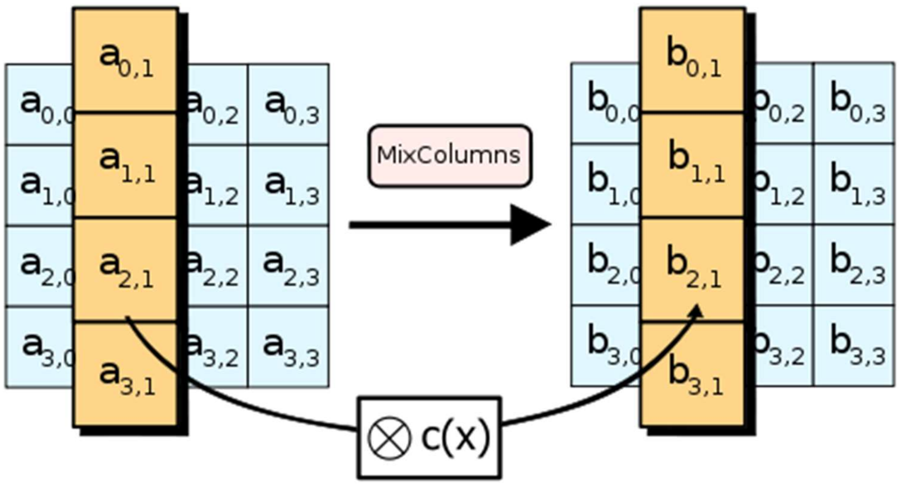
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

3. **MixColumn:** In this process, the four bytes of each column of the state are combined using an invertible linear transformation. The operation consists of the modular multiplication of two four-term polynomials whose coefficients are elements of GF(2^8). The modulus used for this operation is x^4+1.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad 0 \leq j \leq 3$$

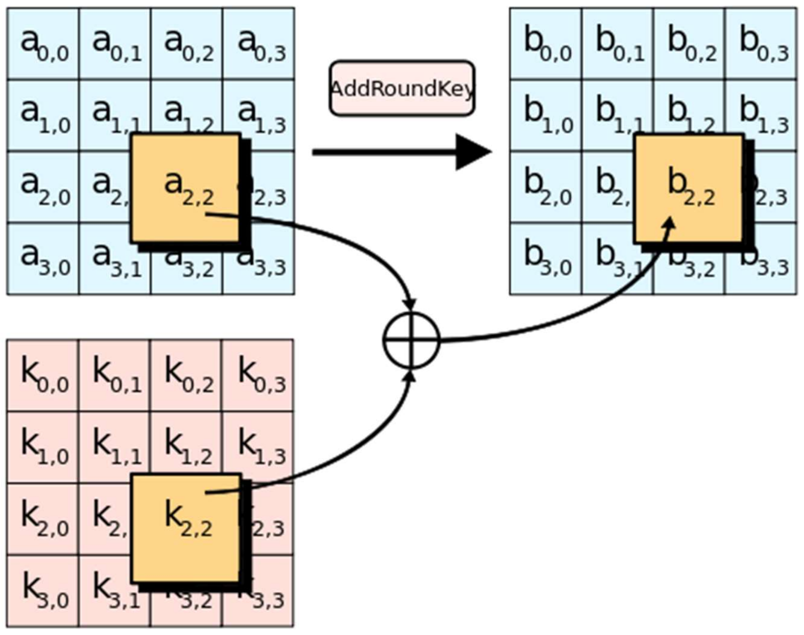
In this case, the circulant matrix is generated using [2 3 1 1]

It is similar to Matrix Multiplication but with different operation types.XOR operation is performed instead of Addition. Multiplication is modulo irreducible polynomial {x^{8}+x^{4}+x^{3}+x+1}. If processed bit by bit, then, after shifting, a conditional XOR with 1B16 should be performed if the shifted value is larger than FF16 (overflow must be corrected by subtraction of generating polynomial). These are special cases of the usual multiplication in GF(2^8).



https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

4. **AddRoundKey:** In this process, the round keys generated from the Key scheduling is combined with the state using XOR operations

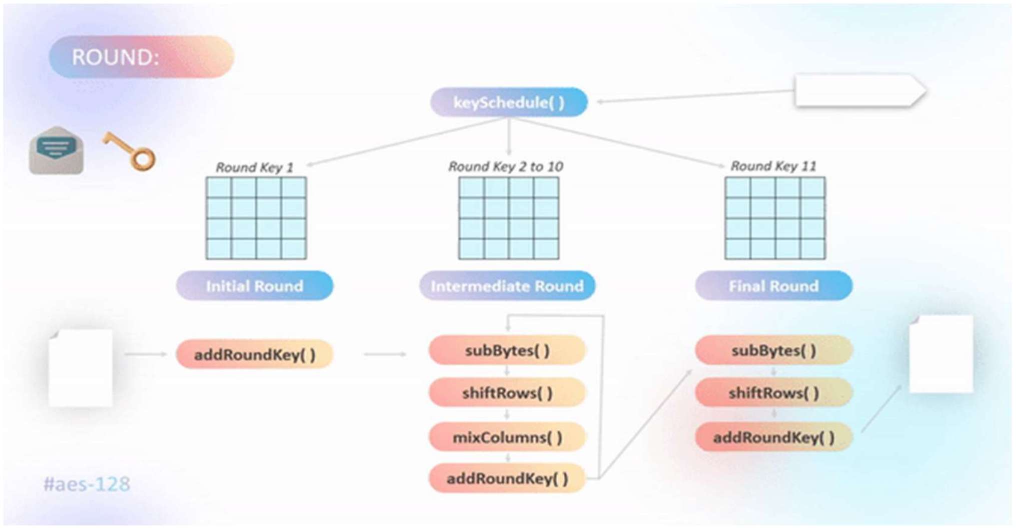


AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$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

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

For performing AES encryption the steps are:

- Initial Round
  - Perform AddRoundKey() with the cipher key
- Intermediate Round
  - SubBytes()
  - ShiftRows()
  - MixColumn()
  - AddRoundKey() with 'n'th round key
- Final Round
  - SubBytes()
  - ShiftRows()
  - AddRoundKey() with the last round key



3.1.3 Decryption

The decryption process in AES is similar to its encryption operation but with some differences in the order and the nature of the operations.

The different operations involved are

1. **Inverse SubBytes**: The process is similar to SubBytes but instead of substituting the value from the S-Box, it uses Inverse S-Box.
2. **Inverse ShiftRows**: The rows are shifted in reverse order of the offset as applied in the ShiftRows.
3. **MixColumn:** This process is also the same in nature but it uses a different circulant matrix which is the inverse of [2 3 1 1].

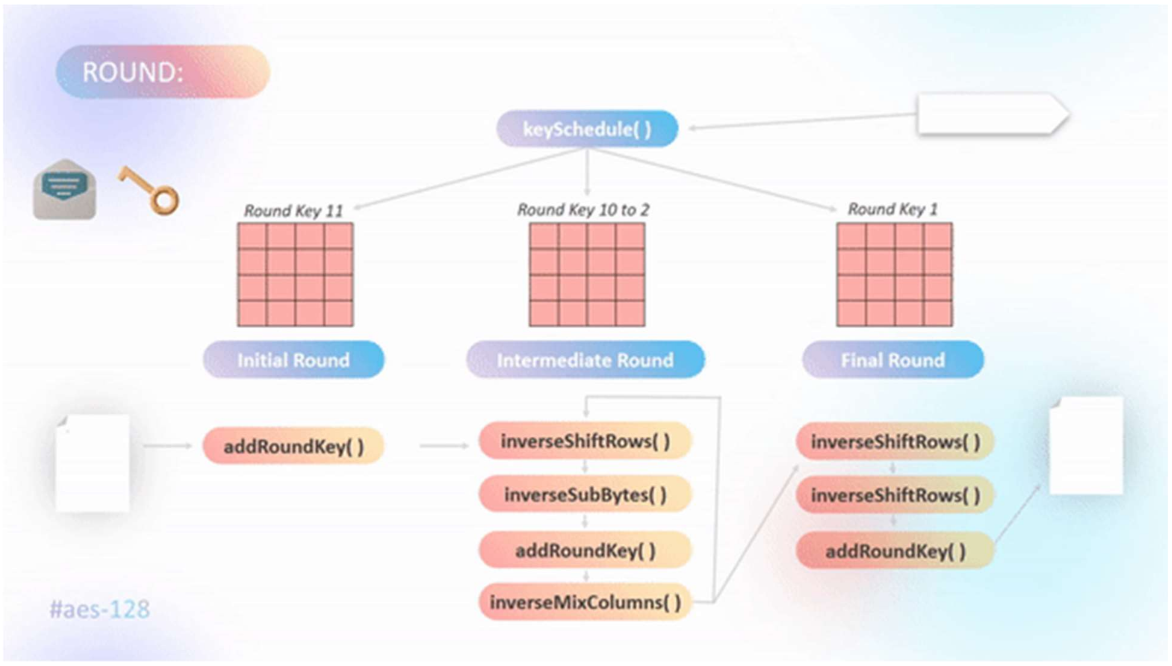
$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad 0 \le j \le 3$$

In this case, the circulant matrix which is used is [14 11 13 9]

4. **AddRoundKey:** In this process, the round keys generated from the Key scheduling are combined with the state using XOR operations in reverse order.

For performing AES decryption, the steps are:

- Initial Round
  - Perform AddRoundKey() with last round key
- Intermediate Round
  - Inverse SubBytes()
  - Inverse ShiftRows()
  - AddRoundKey() with 'n'th round key from the end
  - MixColumn()
- Final Round
  - SubBytes()
  - ShiftRows()
  - AddRoundKey() with cipher key



3.1.4 Padding

As the AES family use 128 bits of data to perform operation therefore it is also important to ensure that the data should fit into that array. However not all data can be a perfect fit, hence padding of extra bits is required. Apart from padding, it is also important to identify and fetch the correct data by removing those padding.

Algorithm to Add Padding

1. x := No. of vacant spaces to be filled
2. y := Hexadecimal form of 'x'
3. Add all the vacant space with 'y' (A hexadecimal form of 'x')

For instance, if the total character required is 16.

```
TEXT := 012345
x := Vacant Space = (16 - 6) = 10
y := Hexadecimal Form = 0a

Append '0a' 10 times to Text

The TEXT after padding:
** ** ** ** ** ** 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a
```

Implementation in Python Programming Language

```
#self.ORDER = 4 (For Square Matrix with order 4)
def __addPadding(self, data):
    bytes = self.ORDER**2           # 16 bytes
    bits_arr = []
    while(True):                    # Loop through the text
        if(len(data) > bytes):      # No Padding
            bits_arr.append(data[:bytes])
            data = data[bytes:]
        else:                       # Padding
            space = bytes-len(data) # Calculate vacant space
            bits_arr.append(data + chr(space)*space)    # Add Padding
            break
    return bits_arr
```

Algorithm to Delete Padding

1. x := Get the last character
2. y := Get the Decimal value of 'x'
3. z := Construct a substring by appending 'x' by 'y'
4. Compare 'z' with the text from the end

For instance, if the total character required is 16.

```
TEXT := 31 54 87 11 12 11 87 07 07 07 07 07 07 07 07 07
x := Last Character = 07
y := Decimal Form = 7
z := 07 07 07 07 07 07 07

Compare 'z' to the TEXT from the end

If exist then REMOVE 'z' from TEXT,
TEXT := 31 54 87 11 12 11 87 07 07

else the same TEXT
```

Implementation in Python Programming Language

```
#self.ORDER = 4 (For Square Matrix with order 4)
def __delPadding(self, data):
    verify = data[-1]                   # Get last Character
    bytes = self.ORDER**2               # 16 bytes
    if(verify >= 1 and verify <= bytes-1):
        pad = data[bytes-verify:]
        sameCount = pad.count(verify)       # Identify Padding
        if(sameCount == verify):            # Padding exist
            return data[:bytes-verify]      # Remove Padding
        return data
```

3.2 Motivation for using AES-512

The new process of attacks is a combination of boomerang and rectangle attack. This uses the weaknesses of a few nonlinear transformations in the key schedule algorithm of ciphers and it can break some reduced-round versions of AES. Rijndael inherits many properties from the Square algorithm. So, the Square attack is also valid for Rijndael which can break round-reduced variants of Rijndael up to 6 or 7 rounds (i.e., AES-128 and AES-192) faster than an exhaustive key search proposed some optimizations that reduce the work factor of the attack [5].

3.2.1 Known Attack

At present, there is no known practical attack that would allow someone without knowledge of the key to read data encrypted by AES when correctly implemented. However, for cryptographers, a cryptographic "break" is anything faster than a brute-force attack – i.e., performing one trial decryption for each possible key in sequence. A break can thus include results that are infeasible with current technology. Despite being impractical, theoretical breaks can sometimes provide insight into vulnerability patterns. By 2006, the best-known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys.

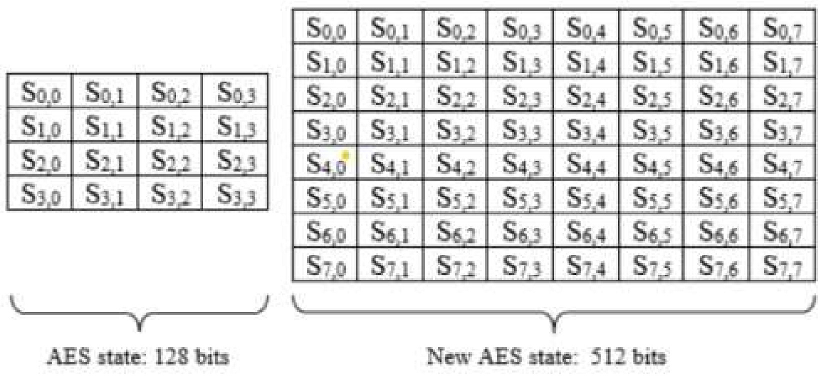
3.2.2 Side-channel attacks

A side-channel attack is a security exploit that aims to gather information from or influence the program execution of a system by measuring or exploiting indirect effects of the system or its hardware rather than targeting the program or its code directly. They attack implementations of the cipher on hardware or software systems that inadvertently leak data. [5] [9] [10]

AES-128 is faster and more efficient and less likely to have a full attack developed against it due to a stronger key schedule. On the other hand, AES-256 is more resistant to brute force attacks and is only weak against related key attacks. As the version of the key schedule for AES-128 seems quite stronger than the key schedule for AES-256 when considering resistance to related-key attacks.

3.3 AES-512 Implementation

In order to increase the robustness of the encryption algorithm, we use longer key expansion which can be achieved by implementing a larger data matrix of order 8. To keep the processing time at low values, we have to maintain the complexity of the AES algorithm upon which the proposed algorithm is partially based. [5] [8]



3.3.1 Key Scheduling

All operations for key expansion will be similar but in a different order.

1. RotWord([b0, b1, b2, b3, b4, b5, b6, b7, b8]) = [b1, b2, b3, b4, b5, b6, b7, b0]
2. SubWord([b0, b1, b2, b3, b4, b5, b6, b7, b8]) = [S(b0), S(b1), S(b2), S(b3), S(b4), S(b5), S(b6), S(b7)]
3. $rcon_i$ = [rc_i, $(00)_{16}$, $(00)_{16}$, $(00)_{16}$, $(00)_{16}$, $(00)_{16}$, $(00)_{16}$, $(00)_{16}$], where the value of rc_i is discussed earlier

Implementation in Python Programming Language

```
def keySchedule(self, KEY):

    #rows and column
    ROW, COL = 8, 8

    # Convert character to hexadecimal matrix state
    hexKey = keyToHexArray(KEY, ROW, COL)

    # Initial Round
    self.ROUNDKEY.append(hexKey)

    # Intermediate Rounds
    for i in range(0, self.ROUND):
        prev_arr = self.ROUNDKEY[-1]
        last_col = prev_arr[ROW-1]
        shift_col = arrayShift(last_col)        # RotWord
        sbox_col = arraySbox(shift_col)         # SubBytes

        col_1 = xorArray(prev_arr[0], sbox_col, self.ORDER, i)  # Round Constant
        col_2 = xorArray(col_1, prev_arr[1], self.ORDER)
        col_3 = xorArray(col_2, prev_arr[2], self.ORDER)
        col_4 = xorArray(col_3, prev_arr[3], self.ORDER)
        col_5 = xorArray(col_4, prev_arr[4], self.ORDER)
        col_6 = xorArray(col_5, prev_arr[5], self.ORDER)
        col_7 = xorArray(col_6, prev_arr[6], self.ORDER)
        col_8 = xorArray(col_7, prev_arr[7], self.ORDER)

        # New Round Keys
        new_arr = np.array([col_1, col_2, col_3, col_4, col_5, col_6, col_7, col_8])

        # Save Round Keys in the List
        self.ROUNDKEY.append(new_arr)

    # Convert 1 8*80 Matrix to 10 8*8 Matrix
    self.convertRoundKey()
```

3.3.2 Encryption and Decryption

The nature of operation of all the operations will be similar except Mix Column and Inverse Mix Column. For the state matrix of order 8, it will require a circulant matrix of 8 columns instead of 4 which was used in the existing AES model. Column mixing is based on the concept of a polynomial over a finite field or Galois field of GF(2n). The columns in the data matrix will be multiplied by a fixed polynomial of a(x) as given below:

Mix Column
The multiplication result is modulo by p(x)= x^8 + 1 to maintain the resulting polynomial with a degree of less than 8. The inverting of column mixing will be multiplied with the inverse of the polynomial a'(x) as given below

$$a(x) = [02]x^7 + [01]x^6 + [03]x^5 + [01]x^4 + [01]x^3 + [01]x^2 + [01]x^1 + [01]x^0$$

$$\begin{bmatrix} s'_{0.c} \\ s'_{1.c} \\ s'_{2.c} \\ s'_{3.c} \\ s'_{4.c} \\ s'_{5.c} \\ s'_{6.c} \\ s'_{7.c} \end{bmatrix} = \begin{bmatrix} 02 & 01 & 03 & 01 & 01 & 01 & 01 & 01 \\ 01 & 03 & 01 & 01 & 01 & 01 & 01 & 02 \\ 03 & 01 & 01 & 01 & 01 & 01 & 02 & 01 \\ 01 & 01 & 01 & 01 & 01 & 02 & 01 & 03 \\ 01 & 01 & 01 & 01 & 02 & 01 & 03 & 01 \\ 01 & 01 & 01 & 02 & 01 & 03 & 01 & 01 \\ 01 & 01 & 02 & 01 & 03 & 01 & 01 & 01 \\ 01 & 02 & 01 & 03 & 01 & 01 & 01 & 01 \end{bmatrix} \begin{bmatrix} s_{0.c} \\ s_{1.c} \\ s_{2.c} \\ s_{3.c} \\ s_{4.c} \\ s_{5.c} \\ s_{6.c} \\ s_{7.c} \end{bmatrix}$$

Inverse Mix Column

$$a'(x) = [0E]x^7 + [01]x^6 + [09]x^5 + [01]x^4 + [0D]x^3 + [01]x^2 + [0B]x^1 + [01]x^0$$

$$\begin{bmatrix} s'_{0.c} \\ s'_{1.c} \\ s'_{2.c} \\ s'_{3.c} \\ s'_{4.c} \\ s'_{5.c} \\ s'_{6.c} \\ s'_{7.c} \end{bmatrix} = \begin{bmatrix} 0e & 01 & 09 & 01 & 0d & 01 & 0b & 01 \\ 01 & 09 & 01 & 0d & 01 & 0b & 01 & 0e \\ 09 & 01 & 0d & 01 & 0b & 01 & 0e & 01 \\ 01 & 0d & 01 & 0b & 01 & 0e & 01 & 09 \\ 0d & 01 & 0b & 01 & 0e & 01 & 09 & 01 \\ 01 & 0b & 01 & 0e & 01 & 09 & 01 & 0d \\ 0b & 01 & 0e & 01 & 09 & 01 & 0d & 01 \\ 01 & 0e & 01 & 09 & 01 & 0d & 01 & 0b \end{bmatrix} \begin{bmatrix} s_{0.c} \\ s_{1.c} \\ s_{2.c} \\ s_{3.c} \\ s_{4.c} \\ s_{5.c} \\ s_{6.c} \\ s_{7.c} \end{bmatrix}$$

3.3.3 Rounds
This AES-512 algorithm can also be implemented using 128 bits, but using that approach would cost us more time as around 22 rounds will be required and the round key produced will be weak in nature with less variation. The use of AES-512 with a 512-bit cipher of order 8 shares the same behavior as AES-128 with a 128-bit cipher of order 4. Therefore this AES-512 algorithm can operate similarly to AES-128 but with longer key expansion and hence 10 rounds is sufficient.

Moreover higher key expansion can also be achieved following this algorithm:

Key Size	**No. of rounds for AES-512 bits (excluding initial round)**
512 bits	10
768 bits	12
1024 bits	14

However, we will follow the 512-bit key size as it produces more strong round keys using AES-512. Using 768 bits and 1024 bits in AES-512 will cause similar problems as faced by AES-192 and AES-256 in 128-bit cipher.

Implementation of AES Encryption in Python Programming Language

```
def encryptProcess(self, TEXT):

hexData = keyToHexArray(TEXT, self.ORDER, self.ORDER)   # Text to Hexadecimal Matrix state

cipher_arr = addRoundKey(hexData, self.ROUNDKEY[0])     # Initial Round: addRoundKey( )


  for i in range(1, self.ROUND+1):                      # Intermidiate & Final Round
    arr = cipher_arr
    arr = subBytes(arr)                                 # -> SubRows( )
    arr = shiftRow(arr, left=True, order=self.ORDER)    # -> ShiftRows( )
    if(i != self.ROUND):                                # -> If not Final Round
      arr = mixColumn(arr, order=self.ORDER)            # -> MixColumn( )
    arr = addRoundKey(arr, self.ROUNDKEY[i])            # -> addRoundKey( )
    cipher_arr = arr
  return cipher_arr
```

Implementation of AES Decryption in Python Programming Language

```
def decryptProcess(self, CIPHER_HEX):
    hexData = hexToMatrix(CIPHER_HEX, self.ORDER)          # Construct Hexadecimal Matrix state
    plain_arr = addRoundKey(hexData, self.ROUNDKEY[-1])     # Initial Round: addRoundKey( )

    for i in range(self.ROUND-1, -1, -1):                  # Intermidiate & Final Round
        arr = plain_arr
        arr = shiftRow(arr, left=False, order=self.ORDER)   # -> ShiftRows( )
        arr = subBytes(arr, inverse=True)                   # -> SubRows( )
        arr = addRoundKey(arr, self.ROUNDKEY[i])            # -> addRoundKey( )
        if(i != 0):                                         # -> If not Final Round
            arr = inverseMixColumn(arr, order=self.ORDER)       # -> MixColumn( )
        plain_arr = arr
    return plain_arr
```

4. LSB Image Steganography

For image steganography, we are using Spatial methods. In the spatial method, the most common method used is the LSB substitution method. The least significant bit (LSB) method is a common, simple approach to embedding information in a cover file. In steganography, LSB substitution method is used. I.e. since every image has three components (RGB). This pixel information is stored in an encoded format in one byte. The first bits containing this information for every pixel can be modified to store the hidden text. For this, the preliminary condition is that the text to be stored has to be smaller or of equal size to the image used to hide the text. The LSB-based method is a spatial domain method. But this is vulnerable to cropping and noise. In this method, the MSB (most significant bits) of the message image to be hidden are stored in the LSB (least significant bits) of the image used as the cover image.

As of now, the implementation of this algorithm is static in nature. As the data which is converted to stego-object may vary which can either cause redundancy of padded bit or may not fully fit inside the stego-object. Moreover, it should also support the mechanism to detect errors using error bits. Overall the image steganography algorithm in our case should be dynamic in nature to accommodate the data perfectly without much redundancy and also to support error detection.

5 Hybrid Blockchain

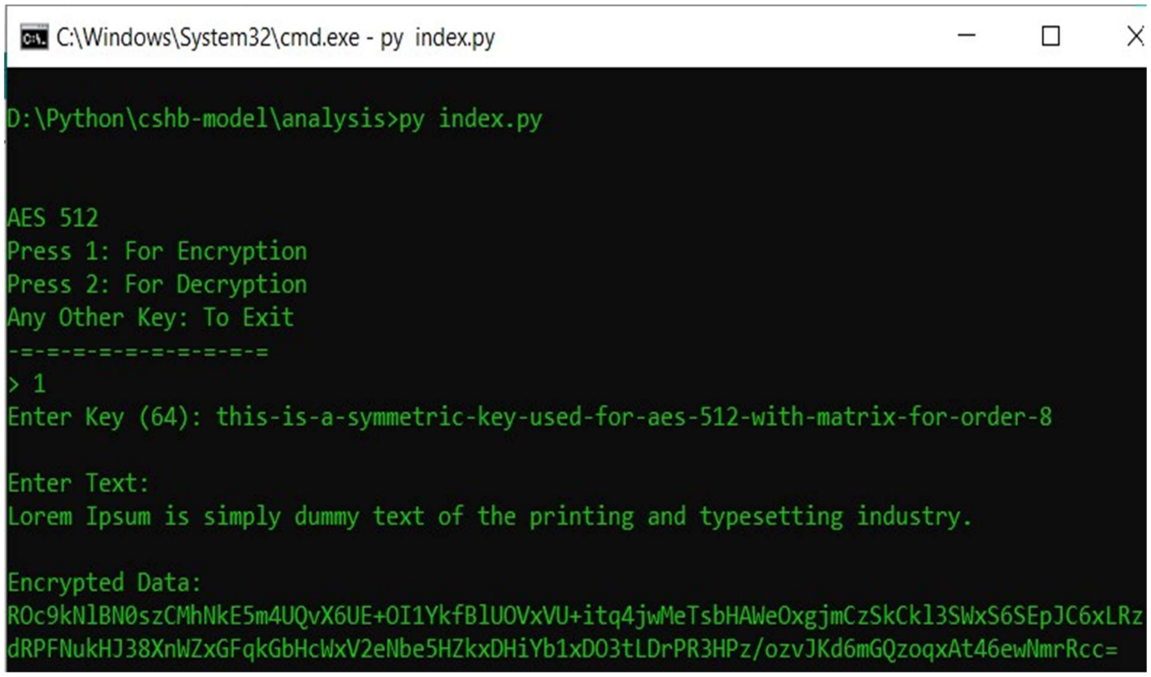
Despite the strength of blockchain, scalability issues like low throughput, high latency, storage issues and poor read performance are its big challenges. Moreover, Blockchain applications have much lower throughput compared to non-blockchain applications [11]. For example, Bitcoin support 3–4 transactions per second as compared to UPI which supports 10,000 transactions per second.

Therefore, it is extremely difficult to implement complete blockchain functionality for a real-time application. In order to check the integrity of the data, the hash of the previous data is linked to the hash of the current data. Moreover, to overcome this challenge the linked hash concept can be implemented in traditional SQL and No-SQL databases with sufficient caching [6]. This functionality to check data integrity can be reduced by restricting the user to verify their data only a few times a day.

EXPERIMENTAL ANALYSIS AND RESULTS

1. Working of AES-512

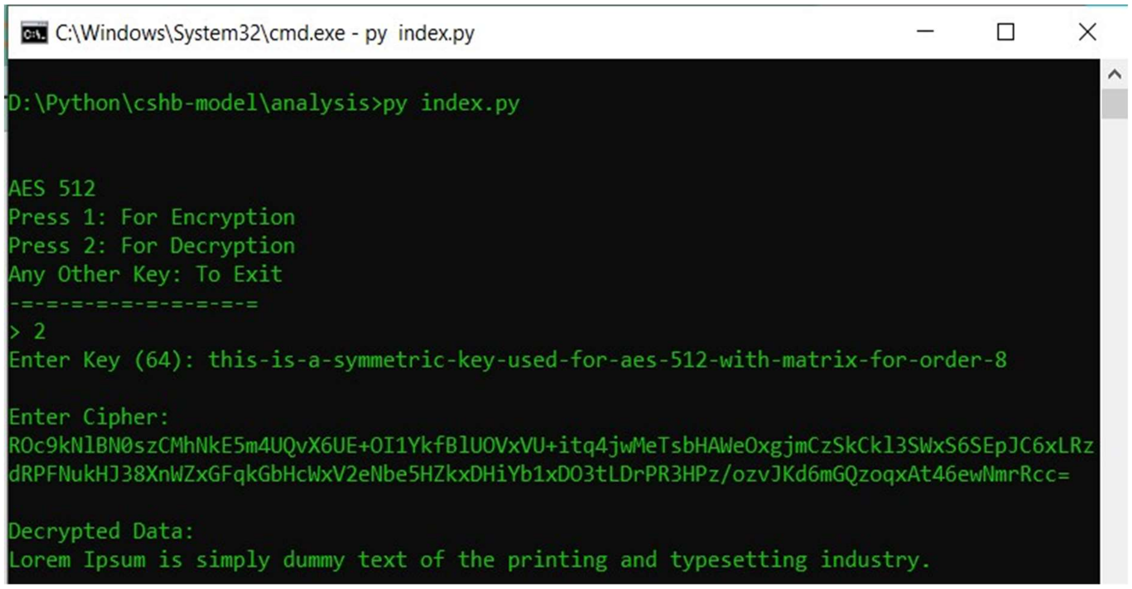
1.1 Encryption



```
D:\Python\cshb-model\analysis>py index.py


AES 512
Press 1: For Encryption
Press 2: For Decryption
Any Other Key: To Exit
-=-=-=-=-=-=-=-=-=
> 1
Enter Key (64): this-is-a-symmetric-key-used-for-aes-512-with-matrix-for-order-8

Enter Text:
Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Encrypted Data:
ROc9kNlBN0szCMhNkE5m4UQvX6UE+OI1YkfBlUOVxVU+itq4jwMeTsbHAWeOxgjmCzSkCkl3SWxS6SEpJC6xLRz
dRPFNukHJ38XnWZxGFqkGbHcWxV2eNbe5HZkxDHiYb1xDO3tLDrPR3HPz/ozvJKd6mGQzoqxAt46ewNmrRcc=
```

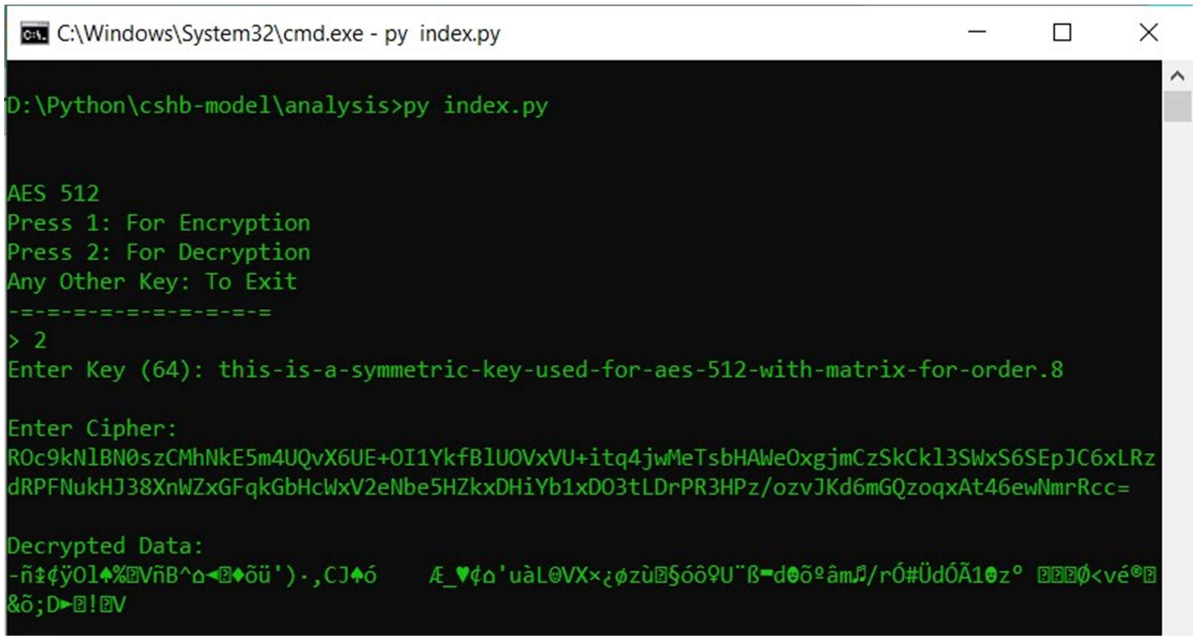
1.2 Decryption

Decrypting using correct symmetric key



```
D:\Python\cshb-model\analysis>py index.py


AES 512
Press 1: For Encryption
Press 2: For Decryption
Any Other Key: To Exit
-=-=-=-=-=-=-=-=-=
> 2
Enter Key (64): this-is-a-symmetric-key-used-for-aes-512-with-matrix-for-order-8

Enter Cipher:
ROc9kNlBN0szCMhNkE5m4UQvX6UE+OI1YkfBlUOVxVU+itq4jwMeTsbHAWeOxgjmCzSkCkl3SWxS6SEpJC6xLRz
dRPFNukHJ38XnWZxGFqkGbHcWxV2eNbe5HZkxDHiYb1xDO3tLDrPR3HPz/ozvJKd6mGQzoqxAt46ewNmrRcc=

Decrypted Data:
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
```

Decrypting using incorrect symmetric key

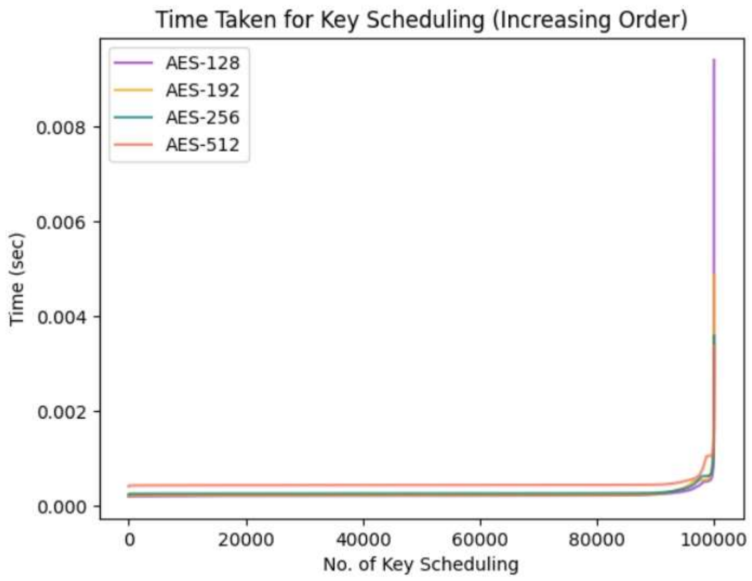

C:\Windows\System32\cmd.exe - py index.py

```
D:\Python\cshb-model\analysis>py index.py


AES 512
Press 1: For Encryption
Press 2: For Decryption
Any Other Key: To Exit
-=-=-=-=-=-=-=-=-=
> 2
Enter Key (64): this-is-a-symmetric-key-used-for-aes-512-with-matrix-for-order.8

Enter Cipher:
ROc9kNlBN0szCMhNkE5m4UQvX6UE+OI1YkfBlUOVxVU+itq4jwMeTsbHAWeOxgjmCzSkCkl3SWxS6SEpJC6xLRz
dRPFNukHJ38XnWZxGFqkGbHcWxV2eNbe5HZkxDHiYb1xDO3tLDrPR3HPz/ozvJKd6mGQzoqxAt46ewNmrRcc=

Decrypted Data:
-ñ‡¢ÿOl♠%VñB^⌂◄♦õü')·,CJ♠ó    Æ_♥¢⌂'uàL☺VX×¿øzù§óô♀U¨ß▬d☻õºâm♫/rÓ#ÜdÓÃ1☻z°  ø<vé®
&õ;D►!V
```

2 Performance Comparisons of AES-512 with AES-128, 192 and 256

2.1 Key Scheduling Time

The observation of the Key Scheduling time for AES algorithm on 1,00,000 keys (1 lakh).

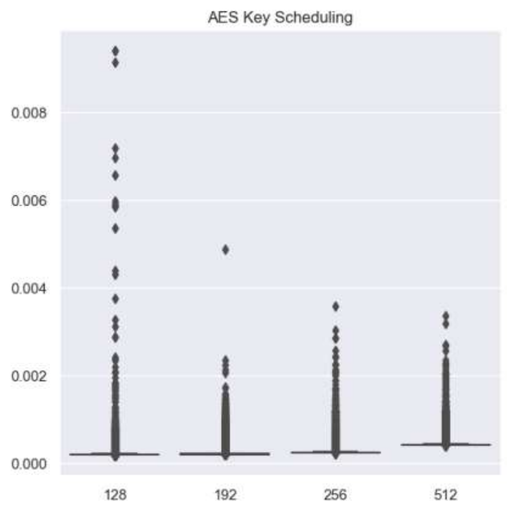
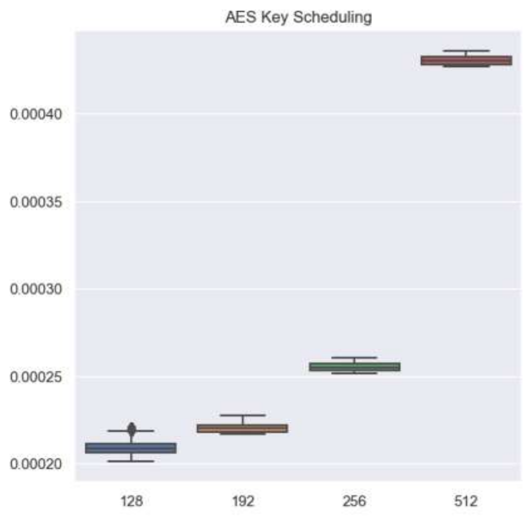


The Observation Table of key scheduling time is given below where the mean, maximum and minimum time is represented in microseconds

	128	**192**	**256**	**512**
mean	223	240	272	448
min	190	201	235	403
max	9403	4878	3578	3369

Box Plot

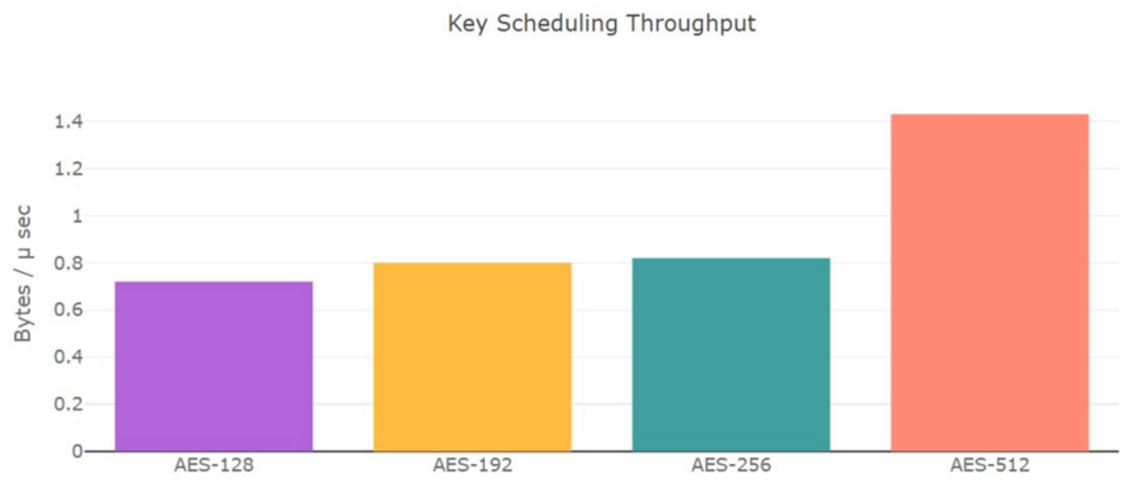
The Box Plot of the observation

With Outlier	**Without Outlier**
	
No Quantile Capping	(12.5 - 87.5)% Quantile Capping

Throughput

Throughput of Bytes generated per seconds in Key Expansions

AES	State Size (Bytes)	Key Size(Bytes)	Expansion	Total Bytes	Average Time	Throughput (Bytes/µ sec)
128	16	16	10	160	223	0.72
192	16	24	8	192	240	0.80
256	16	32	7	224	272	0.82
512	64	64	10	640	448	1.43



Observation

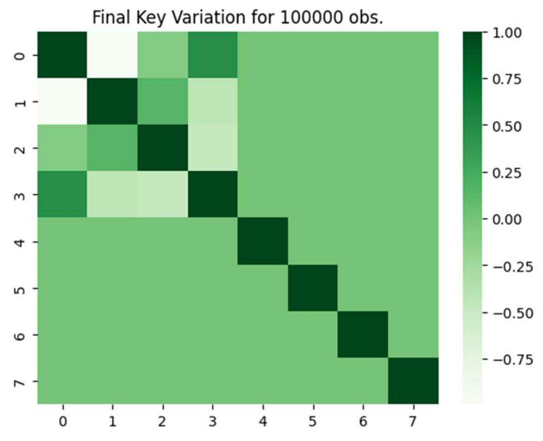
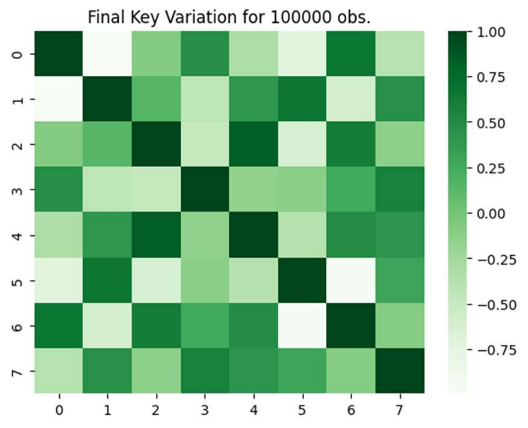
- We can observe that the proposed AES-512 algorithm takes twice the time than AES-128
- There exist certain key combination for which AES-512 is even faster than traditional AES.
- However, the traditional AES key scheduling works on 128 bit state but AES-512 works on 512 bit state. Therefore we can observe that the Throughput for AES-512 higher than other AES variants.
- In the worst case, AES-512 can be delayed up to 3 milliseconds which is around 1/100th of the second considering that Key Expansion occur only once while encrypting and decrypting.

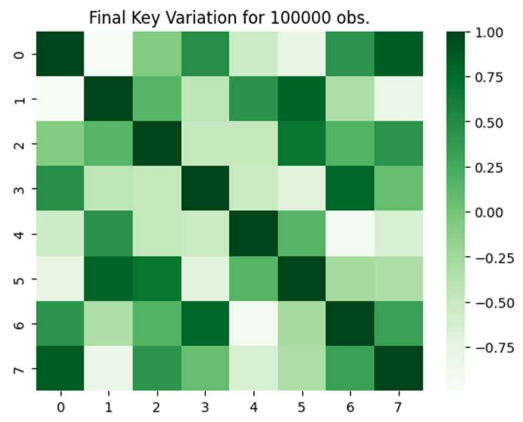
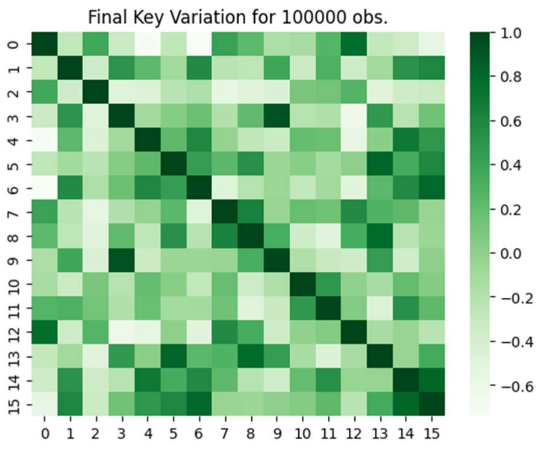
2.2 Variation in Key Scheduling

We will analyse the correlation between the original secret key with the last key obtained in the key scheduling algorithms. As it is observed that AES-256 generate weaker round keys as compared to AES-128, therefore we will try to analysis this statement and will test the key variation of AES-512 algorithm . We will be using a module in Python Programming Language called **numpy** which provides a function called corrcoef which calculate the Pearson product-moment correlation coefficients of single and double dimensional array.

The analysis of secret key and last round keys for 1,00,000 (1 lakh) observation, where the symbol 'ρ' represents the new state generate is strong when the correlation is in range 0.25 to -0.25

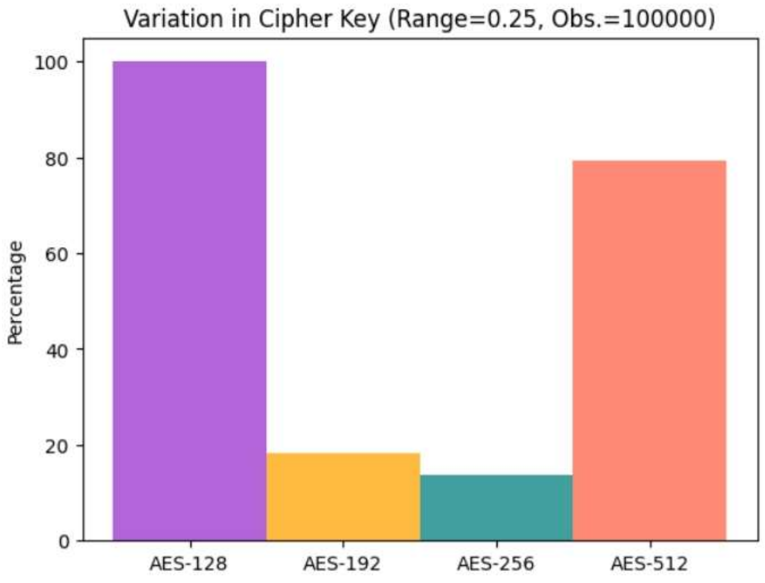
Heat map of Co-relation

AES-128	**AES-192**
	

AES-256	AES-512
	

AES	Required element	Co-relation ≤ 0.25	ρ %
128	22	22	100
192	4	22	18.18
256	3	22	13.64
512	73	92	79.35

Comparison of all AES variants

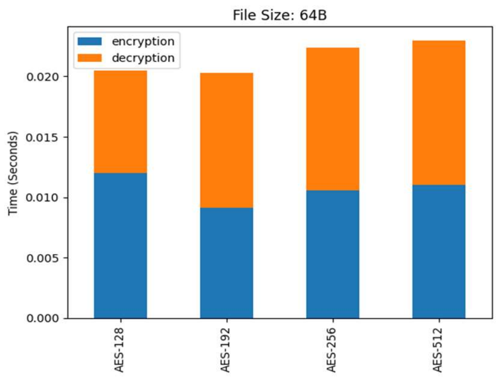
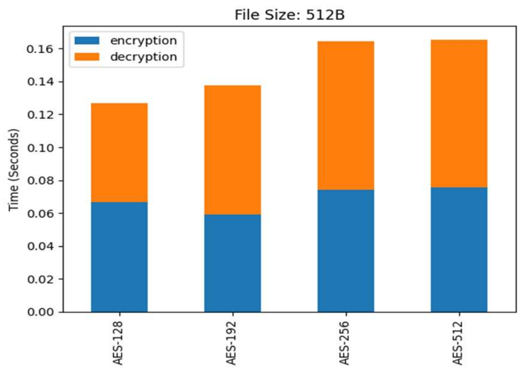
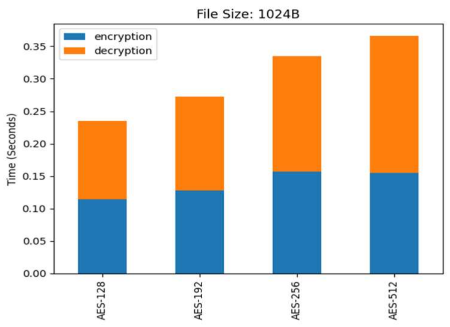
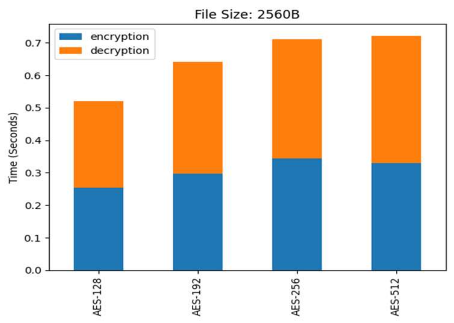


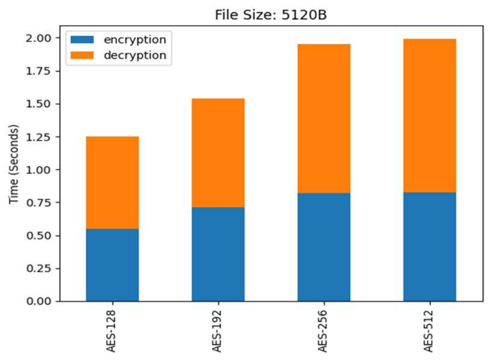
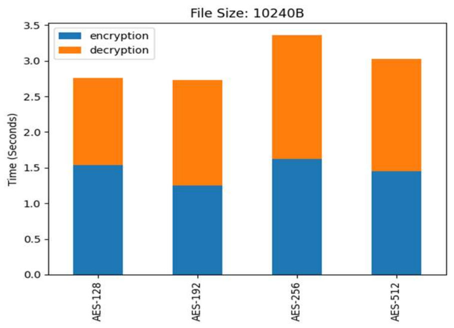
Observation

- AES-128 Key Schedule have less correlation hence it is produces more variation
- AES-256 and AES-192 is mainly have high correlation or high negative correlation
- AES-512 have slightly more correlation as compared to AES-128
- The order of key variations: AES-128 > AES-512 > AES-192 ≈ AES-256

2.3 Encryption and Decryption Time

The analysis of the time taken by the AES algorithm to perform encryption and decryption for different file sizes are

64B	**512B / 0.5KB**
	
1024B / 1KB	**2560B / 2.5KB**
	

5120B / 5KB	**10240B / 10KB**
	

Observation

- AES-512 takes nearly same time as compared to other AES variants mainly AES-256
- However, as the size of data increases the performance of AES-512 increases

Future Scope

i. Optimization of existing AES-512 algorithm by grouping different operations in a round and implementing dynamic programming.

ii. Methods to increase more variation for round keys in key scheduling for AES-512 with an attempt to bring it closer to AES-128.

iii. Dynamic LSB Steganography implementation with an additional space for error bits to successfully detect manipulation of the data with higher accuracy.

iv. Implementation of data schemas at server with additional features of blockchain by storing hash values. Comparing better alternatives for this implementation either using No-SQL (Firebase/MongoDB) or Relational SQL (Supabase/MySQL/Sqlite3).

Reference

1. *Yeuan-Kuen Lee, Graeme Bell, Shih-Yu Huang, Ran-Zan Wang, and Shyong-Jian Shyu: An Advanced Least-Significant-Bit Embedding Scheme for Steganographic Encoding (2009).*
2. *Applied Cryptography Protocols, Algorithms, and Source Code in C, A book by Bruce Schneier.*
3. *H. Gilbert and M. Minier, "A collision attack on seven rounds of Rijndael", Proceedings of the 3rd AES Candidate Conference, (2000) April: 230-241.*
4. *B. Schneier, "The GOST Encryption Algorithm", Dr. Dobb's Journal, v.20, n. 1, (1995) January: 123-124.*
5. *"DATA SECURITY USING 512 BITS SYMMETRIC KEY BASED CRYPTOGRAPHY IN CLOUD COMPUTING SYSTEM", Bijoy Kumar Mandal, Debnath Bhattacharyya and Xiao-Zhi Gao, (2019): 3-4*
6. *Efficient High-Performance FPGA-Redis Hybrid NoSQL Caching System for Blockchain Scalability, Abdurrashid IbrahimSankaMehdi HasanChowdhuryRay C.C.Cheung, 2021.*
7. *The Design of Rijndael AES – The Advanced Encryption Standard, a book by Joan Daemen, Vincent Rijmen, November 26, 2001.*
8. *"AES 512: 512-bit Advanced Encryption Standard Algorithm Design and Evaluation", Abidalrahman Moh'd and Yaser Jararweh Lo'ai Tawalbeh.*
9. *"Ease of Side-Channel Attack on AES-192/256 by Targeting Extreme Keys", Antonie Wurcker.*
10. *"Side Channel Power Analysis of an AES-256 Bootloader", Colin O'Flynn and Zhizing (David) Chen.*
11. *"A Cost analysis of AES-128 and AES-512 on Apple mobile processors", Vatchara Saicheur and Krerk Piromsopa*