Crypto-steganographic Hybrid Blockchain Model for Network Security

B.Tech. Project Report

By

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

Under Supervision of
Dr. Bijoy Kumar Mondal

Department of Computer Sc. and Engineering

Government College of Engineering and Ceramic Technology
Kolkata

April 2023

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)
Akash Kumar Sen (GCECTB-R19-3002)
Arijit Mukherjee (GCECTB-R19-3007)
Sandeep Shaw (GCECTB-R19-3022)
Surajit Bera (GCECTB-R19-3036)

Department of Computer Sc. and Engineering

Government College of Engineering and Ceramic Technology
Kolkata

April 2023

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

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

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

.....

Dr. Bijoy Kumar Mondal
SUPERVISOR
Assistant Professor
Department of Computer Science and Engineering
Government College of Engineering and
Ceramic Technology
Kolkata-700010

.....

JOINT SUPERVISOR

(if any)

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

---

EXTERNAL

Abstract

This project aims to develop a secure and efficient model for end-to-end encrypted data transfer between a client and server, with a focus on promoting data integrity. The model is demonstrated through a text messaging application, which provides users with the assurance that their data is secured against two major data security threats: Man in the Middle (MITM) attacks and incidents of data breaches.

The proposed algorithm addresses the vulnerability of MITM attacks by providing a secure communication channel between the client and server, with encrypted data that cannot be easily decrypted even by brute force attacks. Furthermore, the model promotes data integrity by storing encrypted messages in a centralized database server that only authorized administrators can access. Even in the event of a data breach, sensitive information remains secure as the data is encrypted and cannot be easily decoded.

In addition, the model employs a blockchain mechanism to store text chats between the sender and receiver in a secure and tamper-proof manner. This enables easy identification of any unauthorized attempts to modify the data on the server-side, further improving the overall security and data integrity of the messaging application. Overall, this model presents a significant improvement over existing concepts in terms of security features and data integrity solutions related to messaging chat applications from both client and server-side perspectives.

Acknowledgements

We would like to express our sincere gratitude to our guides, Dr Bijoy Kumar Mandal, for their invaluable guidance and support throughout the course of this project. Their extensive knowledge and expertise in the field of Computer Science and Engineering have been instrumental in shaping the direction and scope of our research.

We would also like to thank our colleagues and peers for their valuable feedback and insights, which have helped us refine and improve our work. In addition, we would like to extend our thanks to the staff and faculty of the Department of Computer Science and Engineering for their support and encouragement throughout our academic journey.

Overall, this project has been an enriching experience for us, and we are grateful for the opportunity to have worked on such an exciting and challenging topic. We hope that our work will contribute to the advancement of knowledge in the field of Computer Science and Engineering and inspire others to explore new frontiers in this exciting field.

Akash Kumar Sen (GCECTB-R19-3002)
Arijit Mukherjee (GCECTB-R19-3007)
Sandeep Shaw (GCECTB-R19-3022)
Surajit Bera (GCECTB-R19-3036)

Content

Chapter 1

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 is 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):

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.

In order to tackle the vulnerability caused by the threats, the key concept and algorithm used in this model in order to achieve our objectives are:

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 arbitrary 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 generated a stego-object in the form of Scalable Vector Graphic (SVG).

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.

Chapter 2

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].

To overcome these challenges, various techniques were implemented in the project such as a dynamic image object generator that uses SVG format to embed the encrypted message. The SVG format provides the flexibility to modify the image size and shape based on the size of the message, and it can be rendered on various devices without compromising the quality of the image. Additionally, to detect errors, we added the hash of the entire message at the end of the data before converting to stego-object.

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)

Symmetric key encryption is a widely used encryption method, where only one key is used for both encryption and decryption of data. Among various symmetric key algorithms, Advanced Encryption Standard (AES) algorithm is the most reliable and secure. AES is byte-oriented and its key length and number of rounds to decode or encode the cipher text can vary depending on the variant used, such as 128, 192, or 256.

AES has two main processes, Key Scheduling and Encryption/Decryption. Key Scheduling involves using operations like Rot Word and Byte Substitution to expand the key and generate several round keys. The Encryption process includes Byte Substitution from S-box, Shifts Rows, Mix Columns, and adding a Round Key generated from Key Scheduling. The Decryption Process is the reverse of Encryption process, consisting of Inverse Byte Substitution, Inverse Shift Rows, Inverse Mix Column, and adding a Round Key, which is the core implementation of AES.

The current AES algorithm uses a square matrix of order 4, which generates a 128-bit cipher text and cipher key. AES-128, AES-192, and AES-256 are the most

commonly used variants of AES. Overall, AES is considered far superior and more secure than its predecessor, Data Encryption Standard (DES).

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.

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.

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.

The text message application's next responsibility is to design a data structure that provides easy CRUD operations and checks the data's integrity against unauthorized changes. Chat data is stored in a hybrid blockchain manner, where the hash of the previous message is stored in the current message, providing verification and traceability of transactions. After requesting encrypted chat data, the server processes the blockchain technology by calculating the data integrity of each chat one by one. To accommodate the linear increase in data size with the number of users, a hybrid model is used in the chat application that combines blockchain concepts with a relational database like SQL or No-SQL, allowing for secure and traceable transactions while maintaining processing speed. This approach balances both data integrity and speed, which are essential features for these types of applications [6].

Chapter 3

Proposed Method

The principal 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 this 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.

Dynamic Image Steganography

Once the data is encrypted, it can be passed as input to generate an image using Steganography algorithms. In this application, we can explore the use of SVG steganography, which has advantages over traditional Least Significant Bit Steganography (LSB). Unlike LSB, SVG steganography can embed data in the vector format of the image, allowing for higher quality and more flexibility. Furthermore, since SVG images are scalable, the size of the image can be adjusted to accommodate the size of the encrypted message, making it more efficient than creating fixed-size images. The resulting Stego-object can then be sent to the server via API.

SVG steganography also has the capability to generate dynamic images based on the encrypted message, allowing for effective embedding of data. Additionally, it can provide additional bits for error correction, which can help detect single bit errors. However, like any steganography method, it is important to implement measures to prevent easy retrieval of data by unauthorized parties. Overall, SVG steganography presents a promising solution for secure and efficient data embedding in our chat application.

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.

Chapter 4

Algorithms

4.1 Secure Hashing Algorithm

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:** Formerly known as Keccak, supports same hash lengths as SHA-2, and its internal structure differs from the rest of the SHA family.

We have used SHA3-512 and SHA-256 to generate a hash to be used in chaining all the encrypted conversion and also to generate a hash to be added at the end of the stego-object which act as an error bit.

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

4.2 Base64 Encoding & Decoding

Base64 is an encoding scheme used to convert binary data into text format for easy transport over the network without any data loss. It solves the problem of misinterpreted bits during data transmission by underlying protocols. Base64 is derived from the MIME standard and was originally used for encoding email attachments for transmission. It is widely used for HTTP and XML protocols. Base64 encoding and decoding are crucial in Advance Encryption Standard for representing encrypted data or decrypting messages. The use of base64 is essential for encryption and decryption algorithms as they require changing data from base64 to binary and vice versa.

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		

https://www.siakabaro.com/base64-decode/

4.3 Advance Encryption Standard (AES-512)

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 commonly used are: AES-128, AES-192 and AES-256.

AES operates on a 4 × 4 column-major order array of 16 bytes b_0, b_1, b_2, ... b_{15} termed the state:

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

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]

The motivation for using AES-512 is that 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].

4.3.1 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 with 128 bits state 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 key length and the number of round keys generated by AES-512 is given in the table the below. The key length will be constant in all cases but depending on the nature of security the rounds may vary. Since we will we using this algorithm for end-to-end encryption for our chat application. Therefore, considering the speed

and efficiency and based on the result and analysis performed by us in the later chapter, we choose 10 rounds of key generations.

AES	Key Length	Round Key Generated	Total (Including initial key)
512	512b / 64B	10	11

The operations involved in Key Scheduling for 128 bits state are:

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

$$\text{RotWord}([b_0, b_1, b_2, b_3]) = [b_1, b_2, b_3, b_0]$$

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

$$SubWord([b_0, b_1, b_2, b_3]) = [S(b_0), S(b_1), S(b_2), S(b_3)]$$

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:

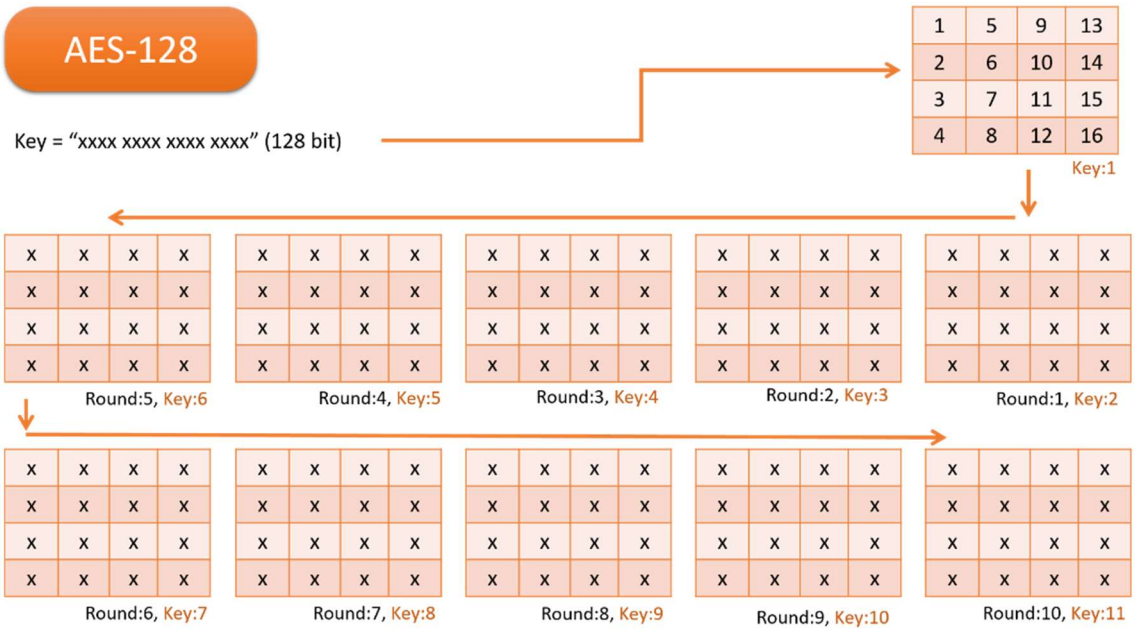
AES	**Round Constant Used (i)**
128	$rcon_{10}$
192	$rcon_8$
256	$rcon_7$

The Key Expansion algorithm for Key Scheduling is given by:

- 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

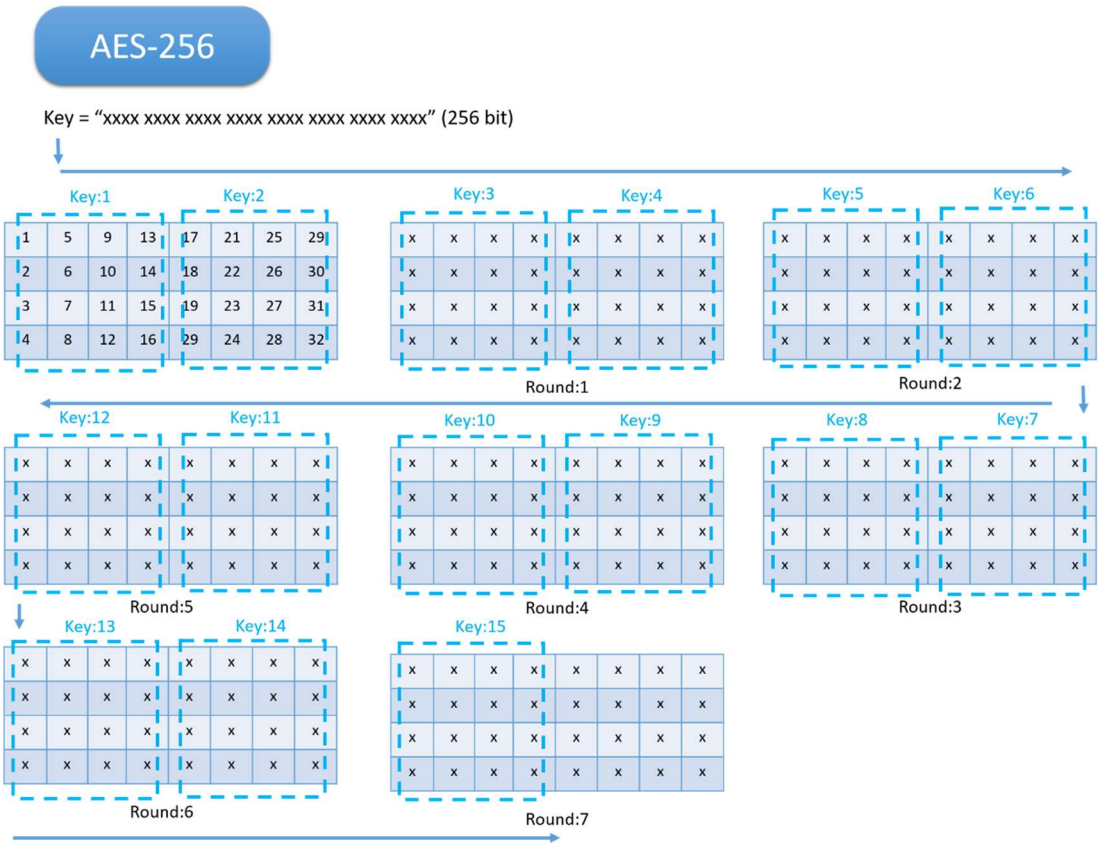
$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1}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

The key expansion visualization for 128 bits AES is:



Key expansion of AES-128

Key expansion of AES-192



Key expansion of AES-256

The AES-512 will perform key scheduling similar to AES-128 and the Key Expansion algorithm is also similar. However, the only difference lies on the size of the state matrix. In order to perform AES-512 the operations are:

$$\boldsymbol{RotWord}([b_0,\ b_1,\ b_2, b_3\ , b_4\ , b_5, b_6, b_7]) = [\ b_1,\ b_2, b_3\ , b_4, b_5, b_6, b_7, b_0]$$

$$\boldsymbol{SubWord}([b_0,\ b_1,\ b_2, b_3\ , b_4\ , b_5, b_6, b_7]) = [S(b_0), S(b_1), S(b_2), S(b_3), S(b_4), S(b_5), S(b_6), S(b_7)]$$

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

Were the value of RotWord, SubWord, $rcon_i$ and rc_i is discussed earlier.

The implementation of this algorithm for AES-512 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()
```

The round constant used by AES-512 are:

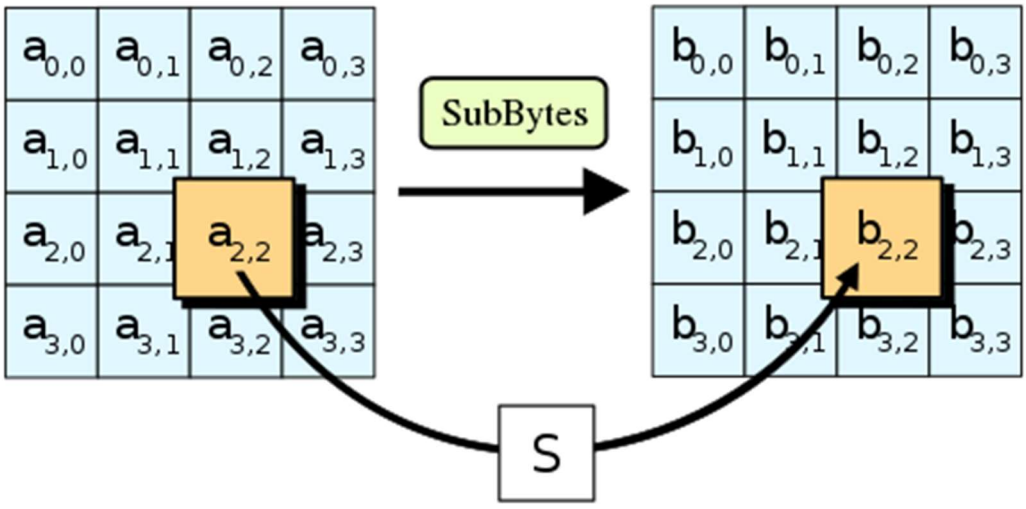
AES	Round Constant Used (i)
512	$rcon_{10}$

4.3.2 Encryption and Decryption

In this process, the plain text is divided into 512-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. Similarly, it the encrypted data can be passed and using same key, we can decrypt the original message which is encoded. Moreover, for Encryption and Decryption the process of Key Scheduling is performed once.

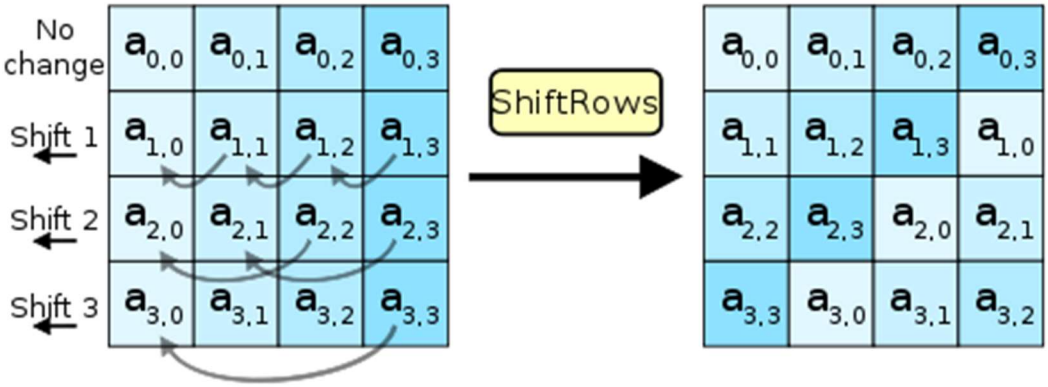
The different operations involved are:

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. In AES-512 the approach will be same but with 8x8 matrix. For decryption we use Inverse SubBytes operation which is exactly the opposite of this method.



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. In AES-512 the approach will be same but with 8x8 matrix. For decryption we use Inverse ShiftRows operation which is exactly the opposite of this method.



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

3. **MixColumn:** 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. For decryption we use Inverse MixColumn operation which is exactly the opposite of this method.

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

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

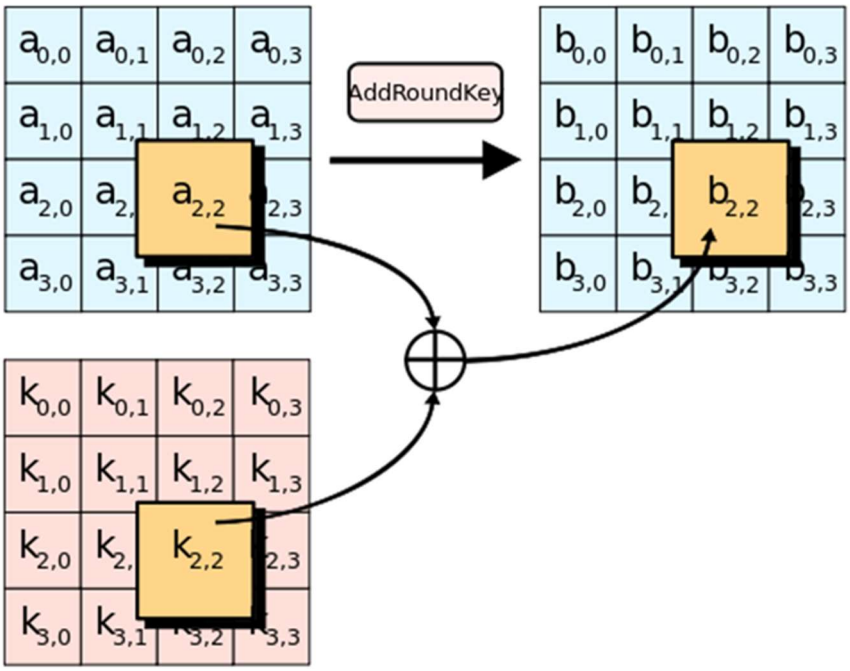
Mix Column

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

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

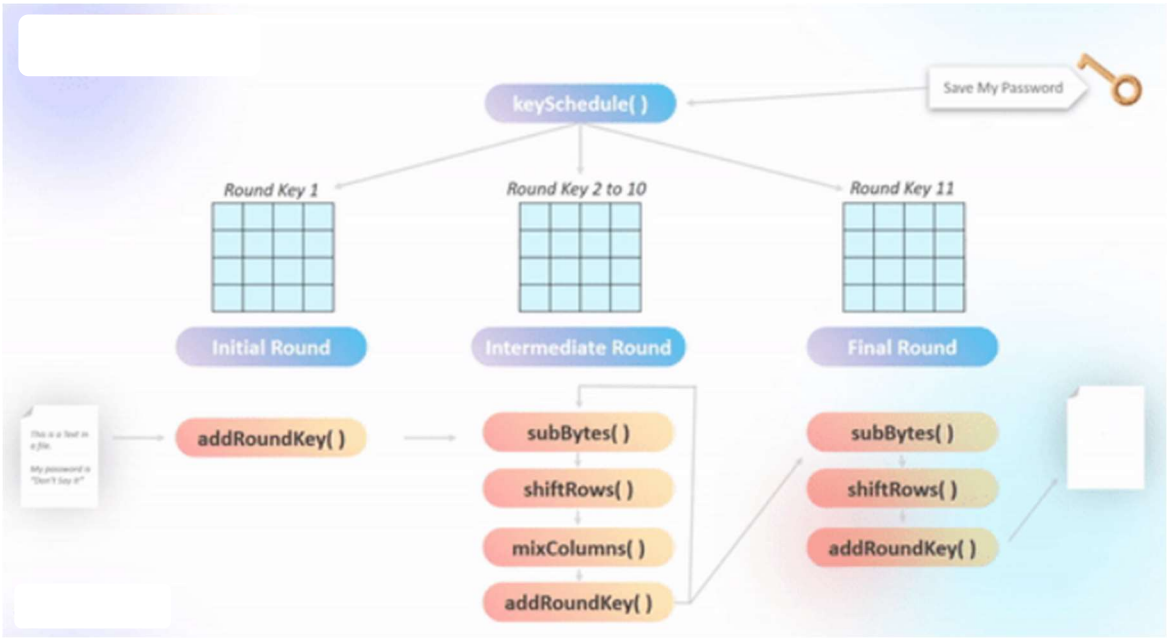
Inverse Mix Column

AddRoundKey: In this process, the round keys generated from the Key scheduling is combined with the state using XOR operations. In AES-512 the approach will be same but with 8x8 matrix. For encryption and decryption, we use Inverse SubBytes operation which is exactly the opposite of this method.

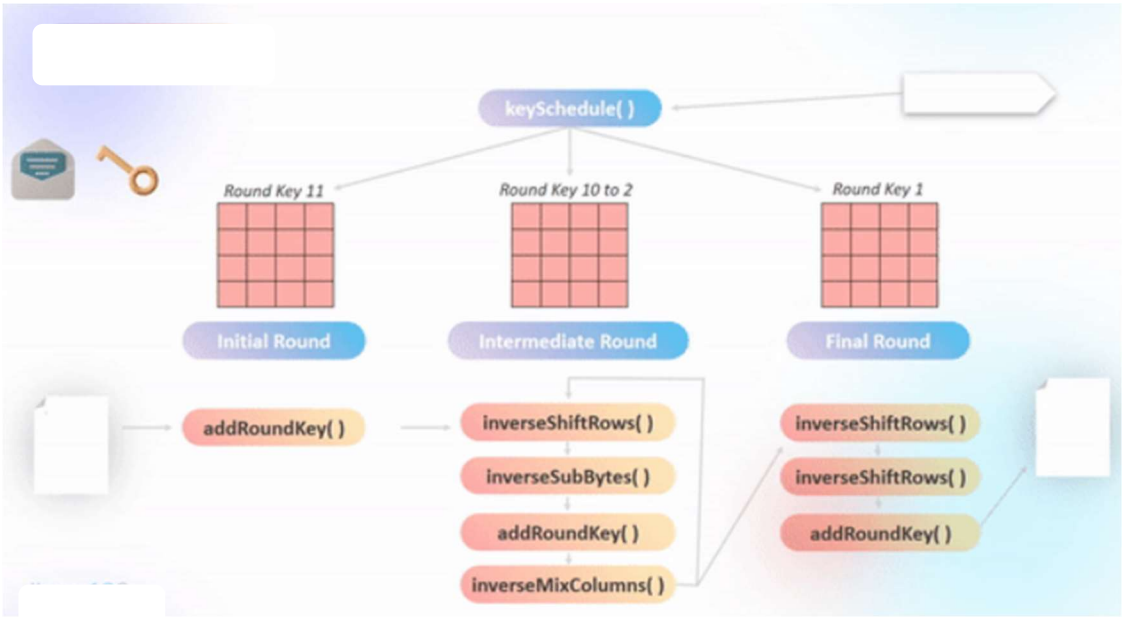


https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

For performing AES encryption, the steps are:



For performing AES decryption, the steps are:



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 behaviour 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
```

4.3.3 Padding

As the AES family use 128 bits of data or AES-512 use 512 bits of data therefore 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
```

4.4 Dynamic Image Steganography

For image steganography, we are currently using the LSB substitution method in the spatial domain. However, this method has limitations such as vulnerability to cropping and noise. Therefore, we are considering the use of a more dynamic and robust method, SVG Dynamic Image Steganography. This approach generates dynamic images based on the encrypted message, allowing for more efficient and effective embedding of data. It also includes error detection capabilities using error bits. Unlike the current implementation, which is static in nature, SVG Dynamic Image Steganography can better accommodate varying data sizes and prevent redundancy of padded bits. Overall, this approach will improve the performance and reliability of our image steganography algorithm.

4.4.1 Binary to Scalable Vector Graphic

For SVG steganography, we convert the binary data into a hexadecimal string using SHA-256 to generate the hash of the message string. This hash is then converted to binary and appended to the message string. The resulting binary string is then converted to a hexadecimal string. This ensures error detection during the retrieval of the message. The hexadecimal string is then passed as input to the function, which generates an SVG image with a series of rectangular shapes representing the hexadecimal colours. This algorithm is dynamic in nature, allowing for the accommodation of data without redundancy, and supports error detection.

Implementation in TypeScript

```
 function hexStringToSVG(hexString: string): void {
  const hexColors = hexString.match(/.{1,6}/g) ?? [];     // validate hexString
  const rectWidth = Math.ceil(100 / hexColors.length);  // set width of each color
  const rects = hexColors.map((hexColor, i) => {
    const x: number = Math.ceil(rectWidth * i);
    if(hexColor.length === 6){
      return `<rect x="${x}%" width="${rectWidth}%" height="100%"
fill="#${hexColor}"/>`
    }
    else{
```

```
      let padColor = `${hexColor}00000`.substring(0, 6)
      return `<rect x="${x}%" width="${rectWidth}%" height="100%"
class="#${hexColor}" fill="#${padColor}"/>`;
    }
  });
  const svgVal = `<svg xmlns="http://www.w3.org/2000/svg" width="100%"
height="100%" fill="none">${rects.join('')}</svg>`;
  setLoadedData(svgVal)                // showing SVG data svgVal
 }
```

4.4.2 Scalable Vector Graphic to Binary

The given function is used to extract hex codes from an SVG image and convert them to a string of binary values. The function takes an SVG image as input in the form of a string and uses a regular expression to match hex codes of varying lengths (ranging from 1 to 6 characters). The matched hex codes are then extracted and concatenated to form a single string of hex values. This string is then validated using another function called validateString and if it is a valid hex string, it is converted to binary using TypeScript functions. Finally, the binary string is returned.

To summarize, the extractHex function extracts hex codes from an SVG image, validates the hex string, converts it to binary and returns the binary string.

Implementation in TypeScript

```
 export function extractHex(data: string): string {
      const regex = /#([a-fA-F0-9]{6}|[a-fA-F0-9]{5}|[a-fA-F0-9]{4}|[a-fA-F0-
9]{3}|[a-fA-F0-9]{2}|[a-fA-F0-9]{1})/g;

    const matches = data.match(regex) ?? [];
    const hexArr =  matches.map((match) => match.replace("#", ""));
    const stringVal =  hexArr.slice(0, -1).join('');
    if(validateString(stringVal)) return stringVal
    return  ' '
}
```

4.4.3 Error Bit Generation and Detection

The protectString() function takes in a message string, generates the SHA-256 hash of the message string, and returns the first 8 characters of the hash. This hash value is then appended to the original message string, which creates a new string with the error bit included. This error bit can be used for detecting errors in the message during the retrieval process.

The validateString() function takes in the stego-object string, removes the last 8 characters (the error bit generated by the protectString() function), and then uses the protectString() function to regenerate the original message string along with the error bit. If the regenerated string matches the original stego-object string, then the message is considered valid, and no errors were introduced during the transmission process. If the strings do not match, then an error was detected, and the message should be considered corrupt.

In summary, these functions use the SHA-256 hash to generate an error bit that is appended to the message string to detect any errors during transmission. The error bit is then used to validate the stego-object during the retrieval process.

Implementation in TypeScript

```
export function protectString(data: string): string {
  let hashVal = hashSHA256(data).substring(0, 8)
  return `${data}${hashVal}`;
}

export function validateString(data: string): boolean {
  let dataVal = data.slice(0, -8)
  let requireVal = protectString(dataVal)
  return requireVal === data
}
```

4.5 Hybrid Blockchain

Despite the strength of blockchain, scalability issues such as low throughput, high latency, storage problems, and poor read performance pose significant challenges. Furthermore, blockchain applications have much lower throughput compared to non-blockchain applications [11]. For instance, while Bitcoin supports 3-4 transactions per second, UPI supports 10,000 transactions per second. Therefore, it is extremely difficult to implement complete blockchain functionality for real-time applications.

To ensure data integrity, the hash of the previous data is linked to the hash of the current data. To overcome scalability issues, the linked hash concept can be implemented in traditional SQL and No-SQL databases with sufficient caching [6]. However, to reduce the functionality required for checking data integrity, users can be restricted to verify their data only when they wish to do so. Otherwise, implementing this functionality every time a chat is sent could significantly slow down the system as the number of chats increases.

Algorithm to implement hybrid blockchain functionality in No-SQL

1. Initialize a database with a chat collection
2. For each message in the chat:
    a. Generate the hash of the previous message
    b. Append the hash to the attribute of the current message
    c. Insert the message into the chat collection

Algorithm to verify the integrity of the chat using hybrid blockchain

1. For each message in the chat:
    a. Generate the hash of the previous message
    b. Compare the generated hash with the hash attribute of the current message
    c. If the hashes are equal, move on to the next message
    d. If the hashes are not equal, the chat has been tampered with
2. If all hashes match, the chat has not been tampered with.

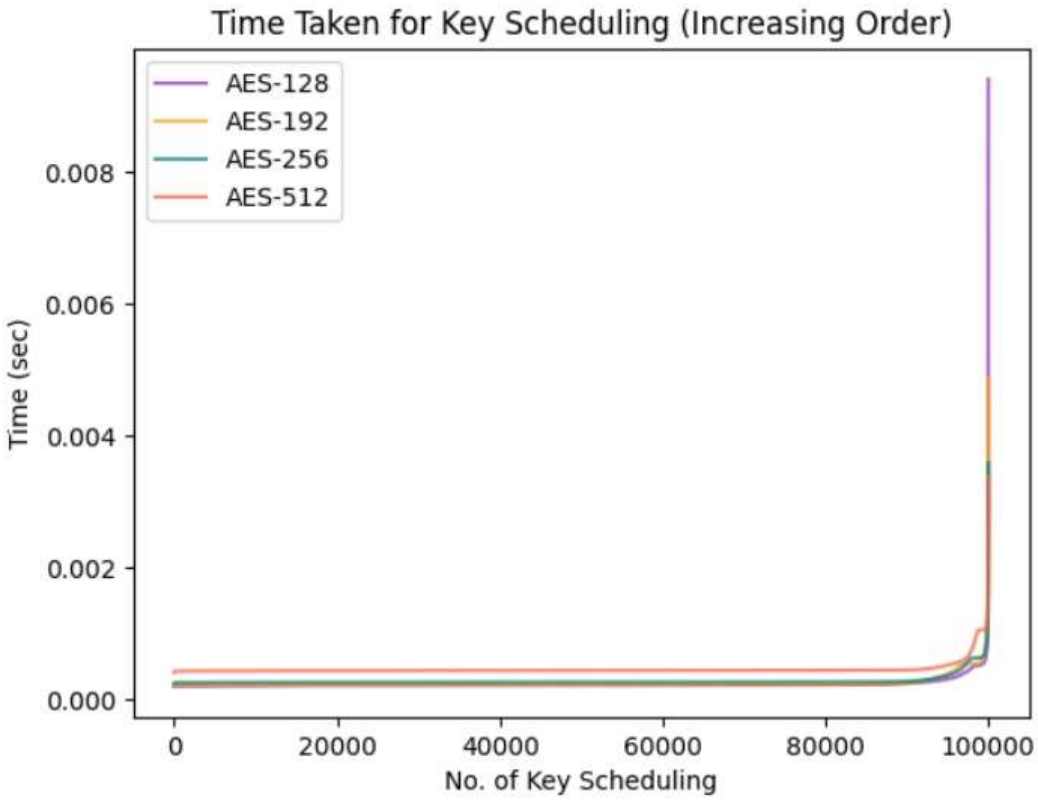
Chapter 5

Experimental Analysis and Results

5.1 Key Scheduling Throughput

The Key Scheduling Time of various 128 bits AES family like AES-128, AES-192 and AES-256 is compared with 512 bits state of AES-512. The observation in made on a randomly generated 1,00,000 keys and their Key Scheduling time is calculated for all the AES algorithm. One of the important things to consider is that the key size of different AES families is different and there might exist a scenario where the larger size AES key algorithm might perform faster than those of smaller key size algorithm.

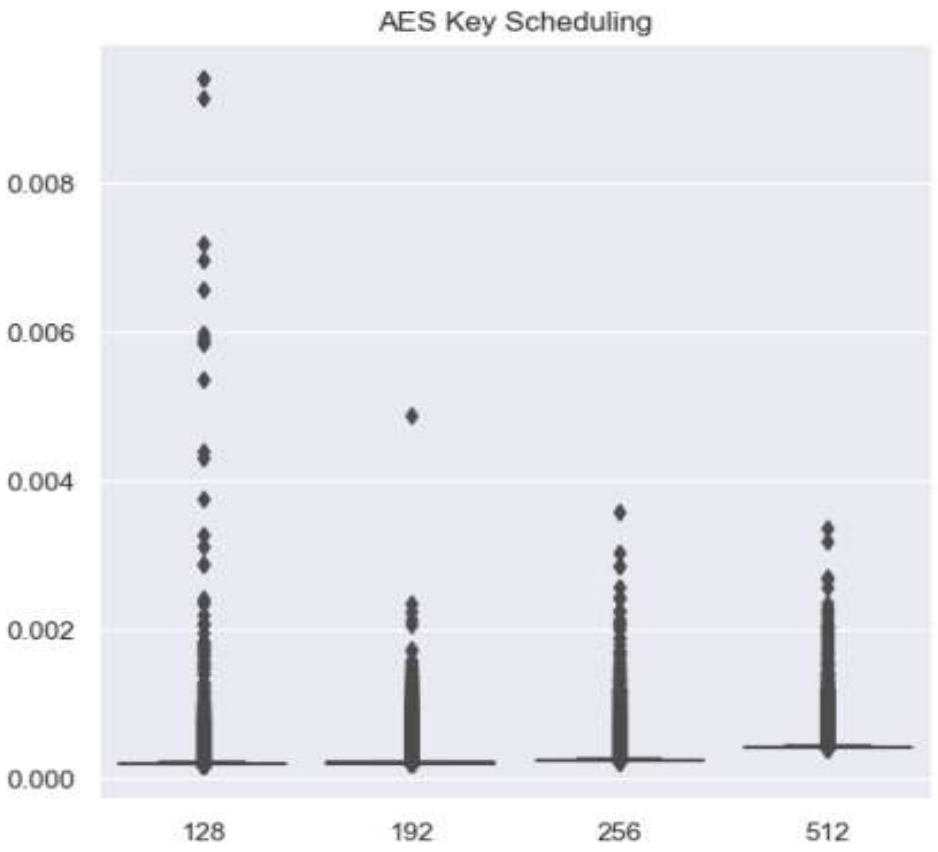
This analysis was conducted using Python Jupyter and the observed charts and data where the time is recorded in terms of seconds.



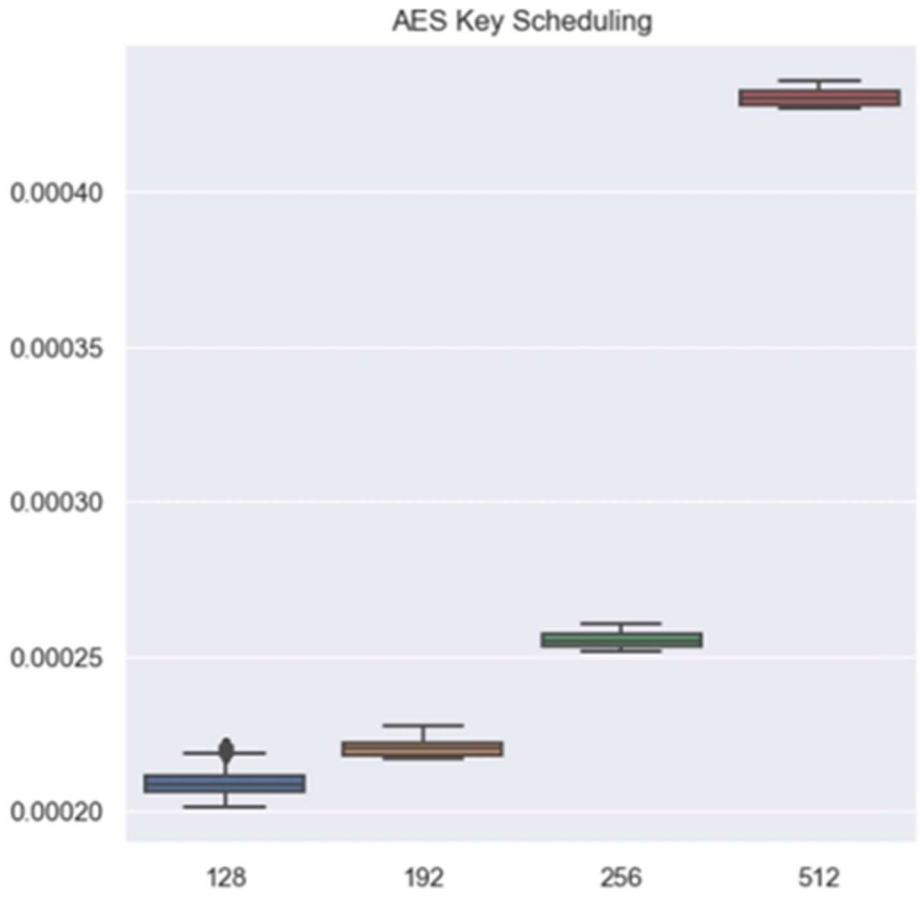
The observed data in terms of average time, maximum and minimum time taken in microseconds.

State Matrix	128 bits			512 bits
Algorithm	**AES-128**	**AES-192**	**AES-256**	**AES-512**
mean	223	240	272	448
min	190	201	235	403
max	9403	4878	3578	3369
range	9213	4677	3343	2966

The box-plot observation of the key scheduling time including Outliers



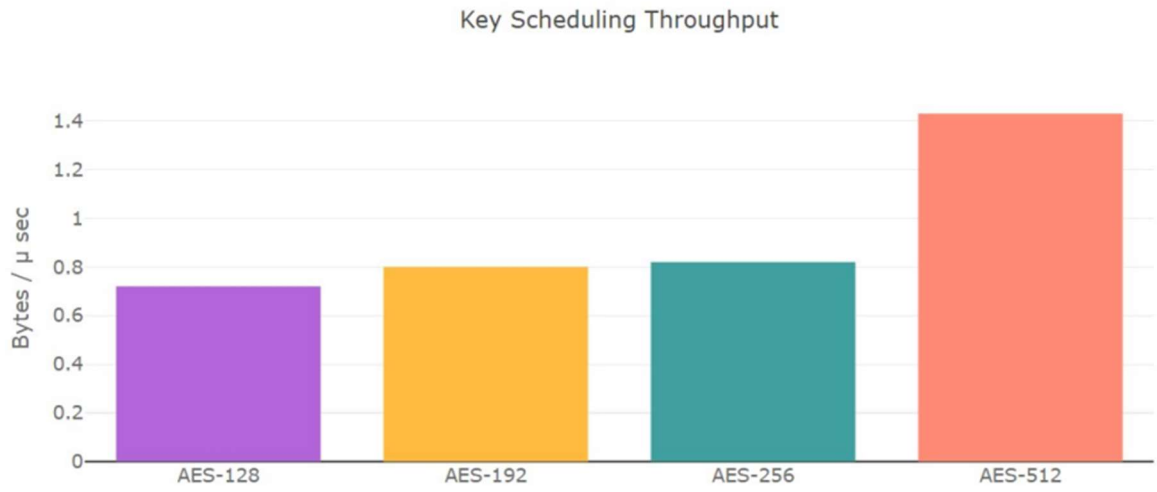
The box-plot observation of the key scheduling time excluding Outliers by applying Quantile Capping in range 12.5% to 87.5%



As AES-512 is working on 512 bits state matrix, therefore it is also important to observe the throughput as compared to other AES algorithm of 128 bits.

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

The graph plot of the Throughput for Key Scheduling



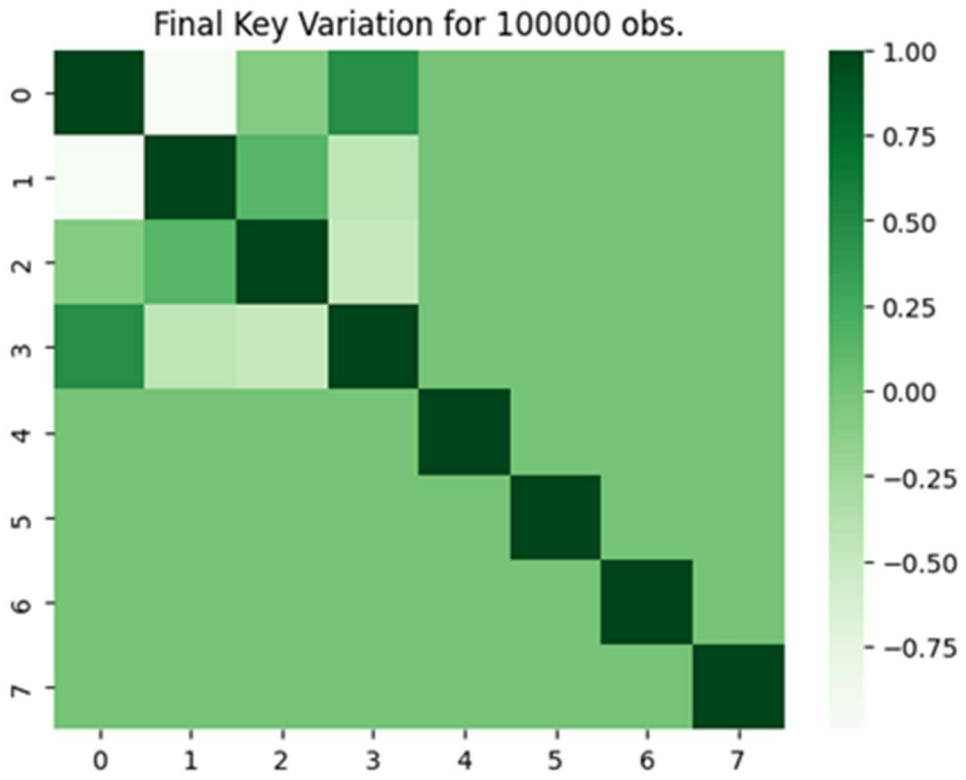
Observation

1. We can observe that the proposed AES-512 algorithm takes twice the time than AES-128.
2. There exists certain key combination for which AES-512 is even faster than traditional AES.
3. 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.
4. 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.
5. When we observe the boxplot with outlier, the range of value of AES-512 is very less as compared to others. This imply that there might exist a scenario with certain key combination which perform better than all AES family of 128 bits.

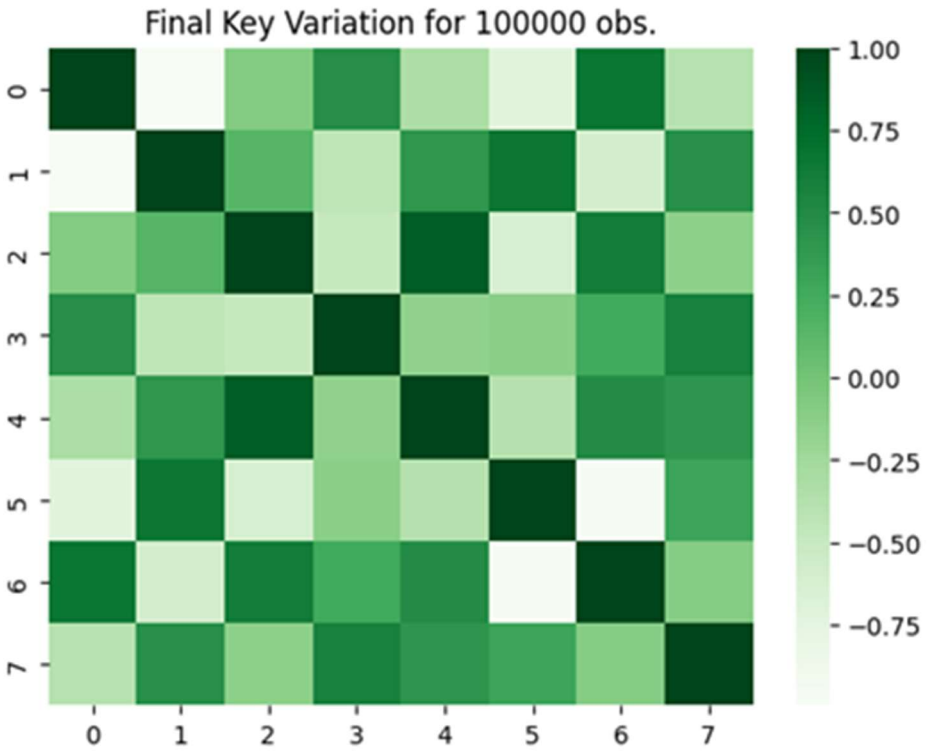
5.2 Key Scheduling Variation

We will analyse the correlation between the original secret key with the final round 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.

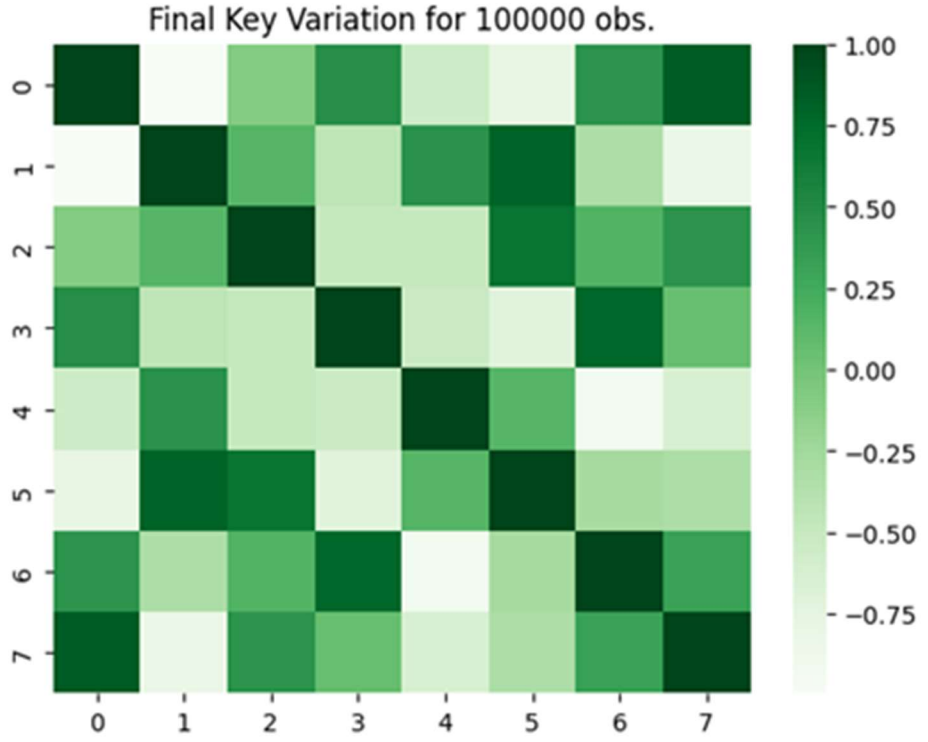
The analysis of secret key and last round keys for 1,00,000 observations, where the symbol 'ρ' represents the new state generate is strong when the correlation is in range 0.25 to -0.25. This is the value range assumed by us for as higher magnitude value represents strong negative or positive correlation.



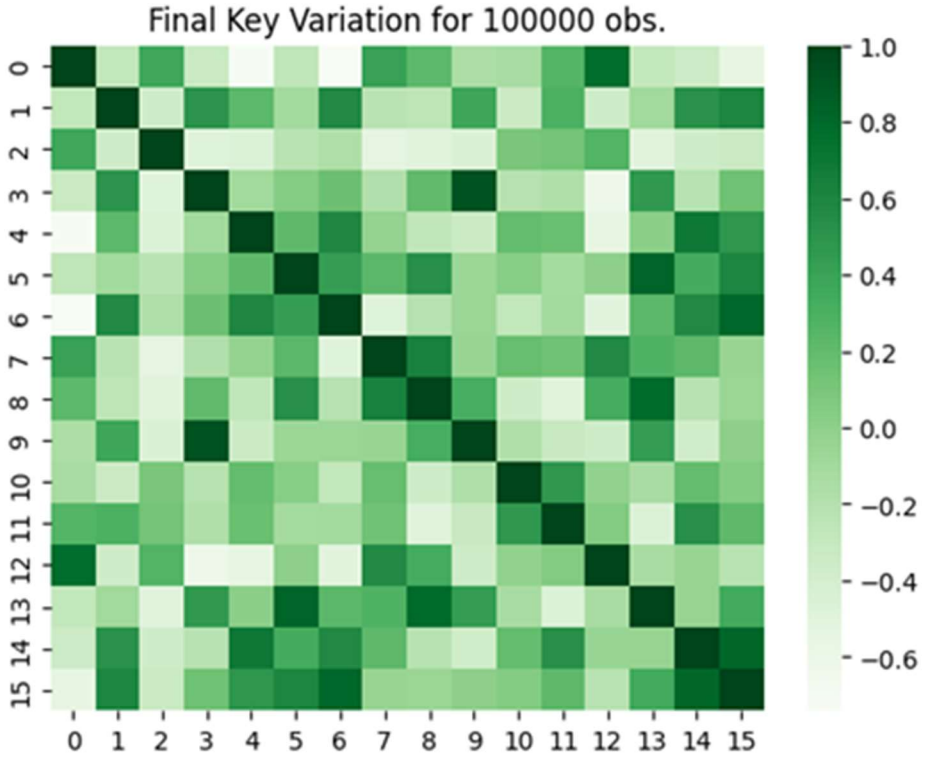
Heatmap of AES-128



Heatmap of AES-192



Heatmap of AES-256



Heatmap of AES-512

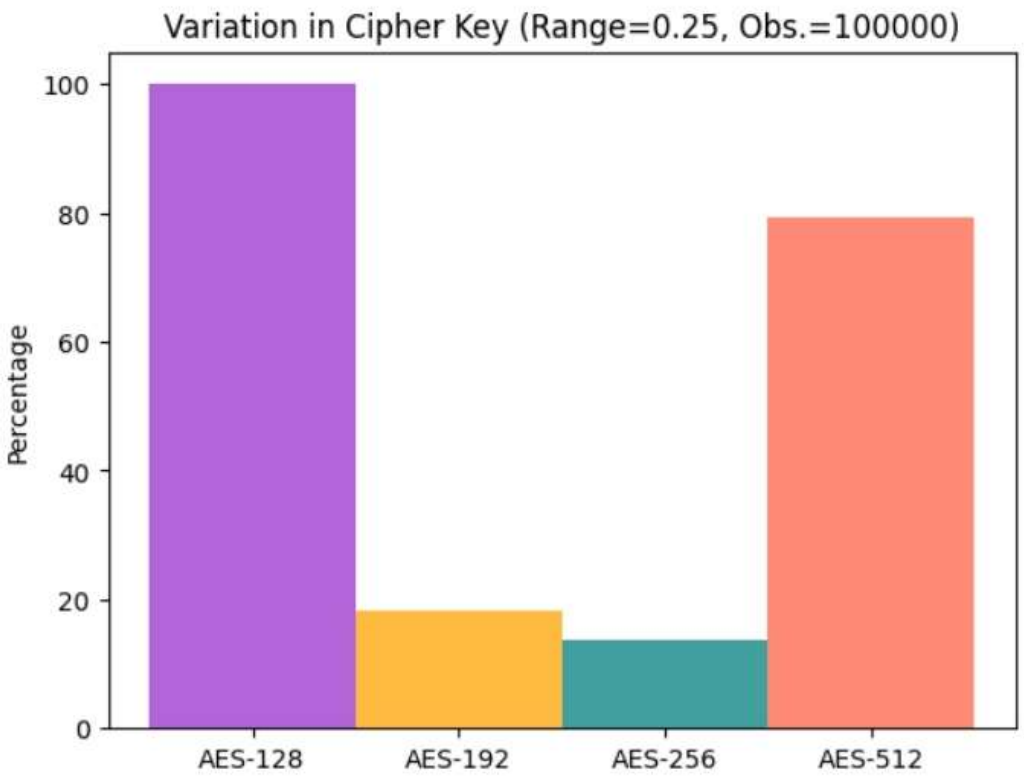
The heatmap plotted above is symmetric in nature, the diagonal is highly related as it is compared with itself. Moreover, for AES-128, AES-192 and AES-256 while comparing the initial and final round key, the square matrix says ‘A’ from A [0][0] to A [3][3] may not show uniform correlation as it is comparing with the same matrix state. As we are interested to find variation with 2 different matrix and not find variation with itself, therefore value from A [0][0] to A [3][3] along with the diagonal element A [n][n] ($0 \geq n \geq 7$) can be omitted.

Similarly for AES-512 the omitted value from the Heatmap plot will be the square matrix say A, from A [0][0] to A [7][7] and the diagonal element A [n][n] ($0 \geq n \geq 15$).

After omitting the redundant and unwanted value, the percentage of variation that lies within 'ρ' [-0.25, 0.25] is observed in the table below:

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

The graph plot of the variation which helps us to visualize the variations in key scheduling when 'ρ' lies within the magnitude 0.25



Observation

1. The AES-128 key schedule has been found to exhibit less correlation compared to other AES key sizes, resulting in a wider range of variation during key scheduling. This increased variation can lead to improved security, as it makes it more difficult for attackers to predict the key schedule and recover the original key.

2. AES-256 and AES-192, on the other hand, tend to exhibit higher correlation or negative correlation in their key scheduling, which can result in less variation and potentially weaker security. However, it should be noted that these correlations may not be exploitable in practice.

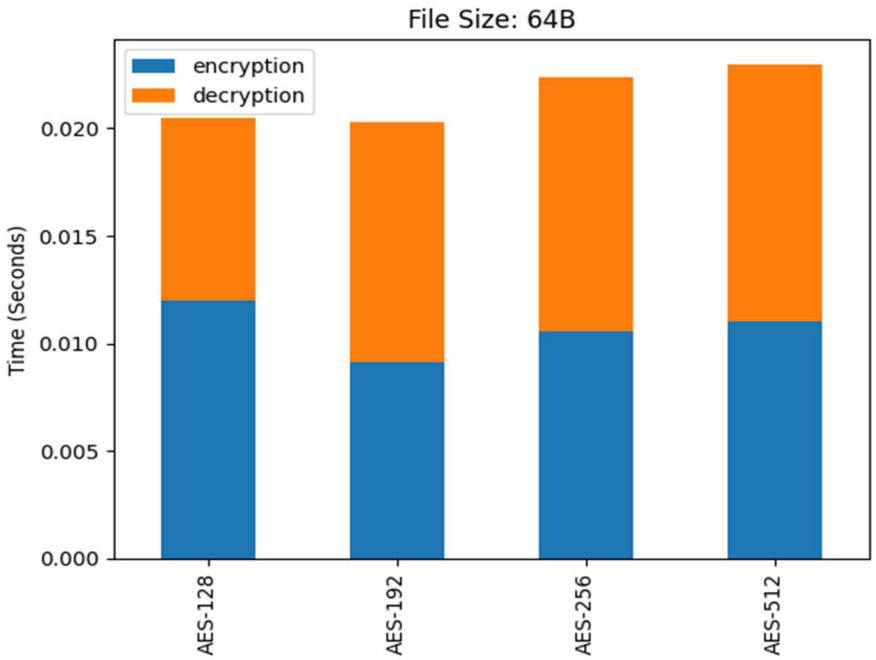
3. AES-512 has been found to exhibit slightly more correlation in its key scheduling than AES-128. This may make it somewhat more predictable than AES-128, but it still offers a high level of security due to its large key size.

4. Based on the order of key variations, it can be observed that AES-128 has the highest variation during key scheduling, followed by AES-512, and then AES-192 and AES-256, which exhibit similar levels of variation. This order of key variations can have important implications for the security and efficiency of these algorithms in various applications.

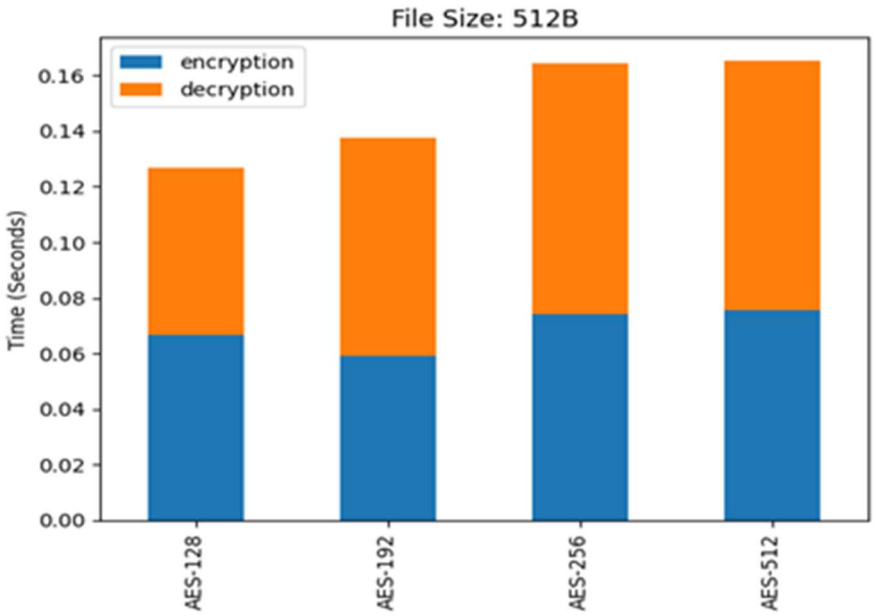
5.3 Encryption & Decryption Time

The analysis of the time taken by the different AES algorithm to perform encryption and decryption for different file sizes is observed in a graph plot.

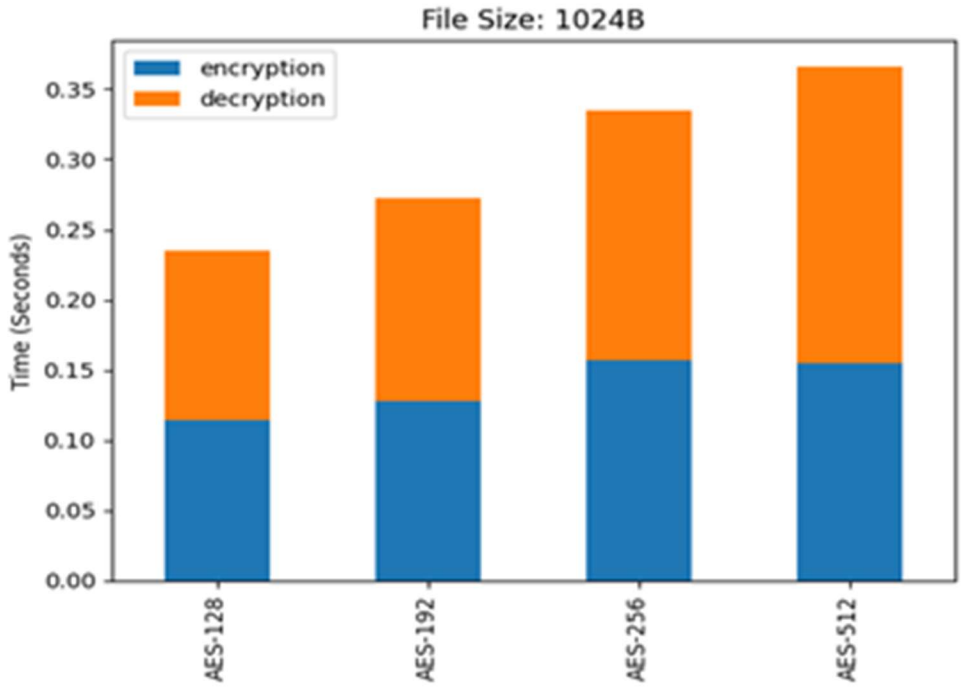
File Size: 64 bytes



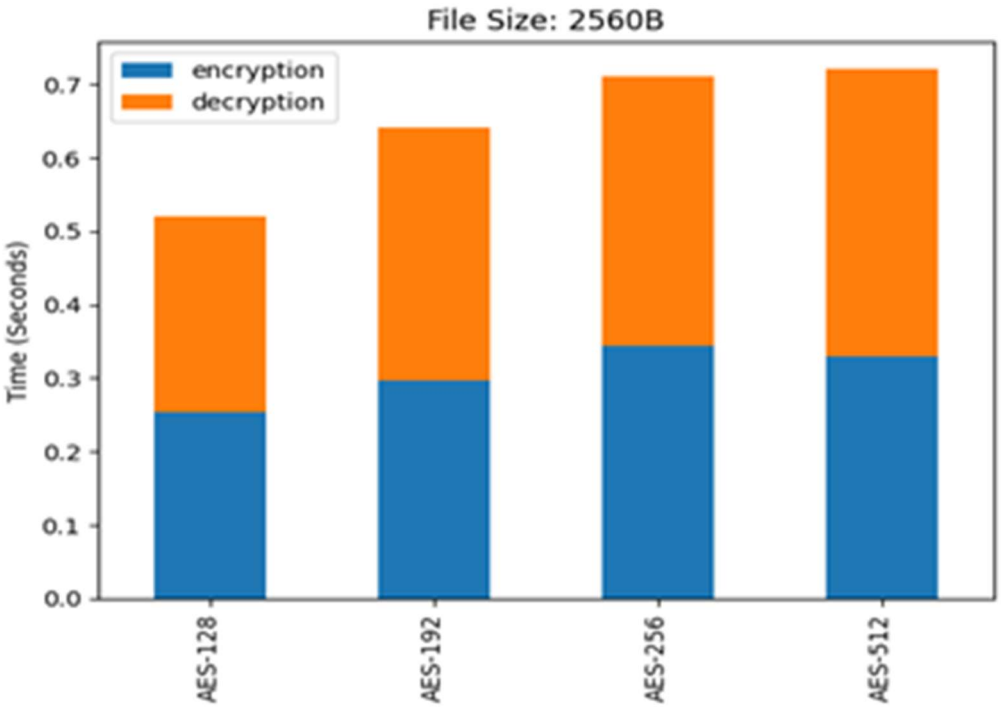
File Size: 512 bytes / 0.5 KB



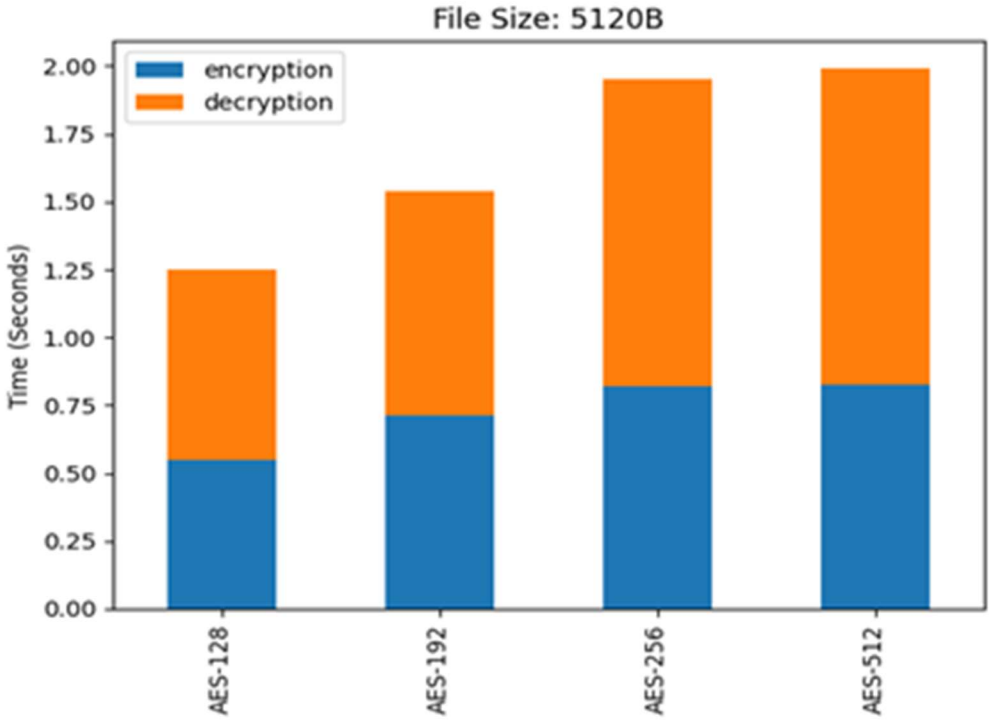
File Size: 1024 bytes / 1 KB



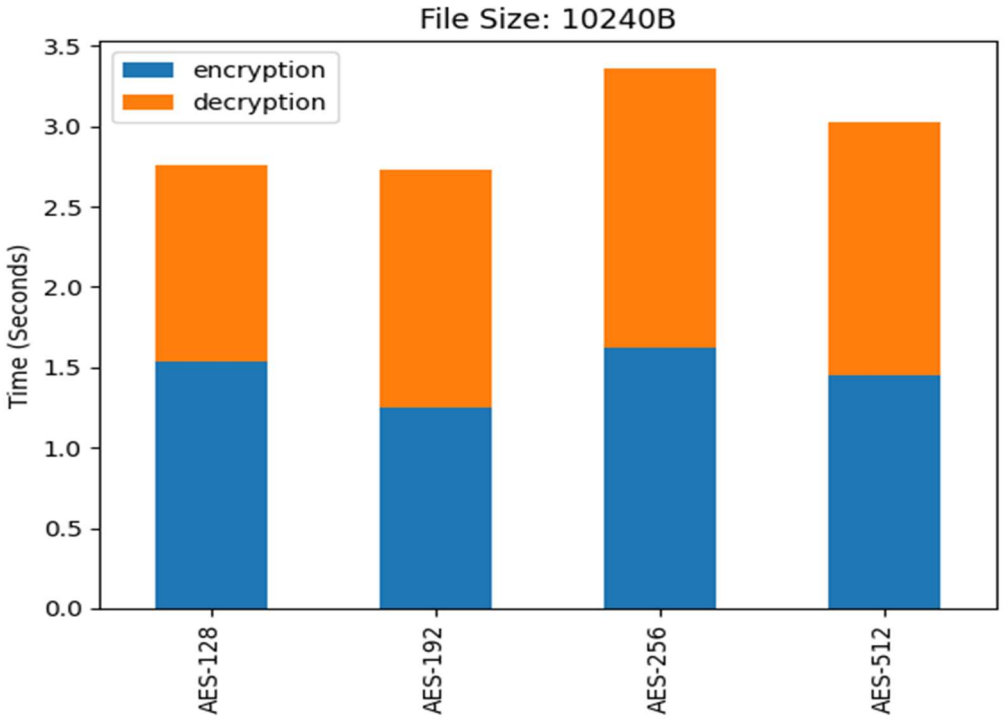
File Size: 2560 bytes / 2.5 KB



File Size: 5120 bytes / 5 KB



File Size: 10240 bytes / 10 KB



Observation

1. AES-512 takes nearly the same amount of time as compared to other AES variants, especially AES-256.

2. However, the performance of AES-512 improves as the size of data increases. This is because AES-512 has a larger key size and is able to handle larger data sizes more efficiently.

Chapter 6

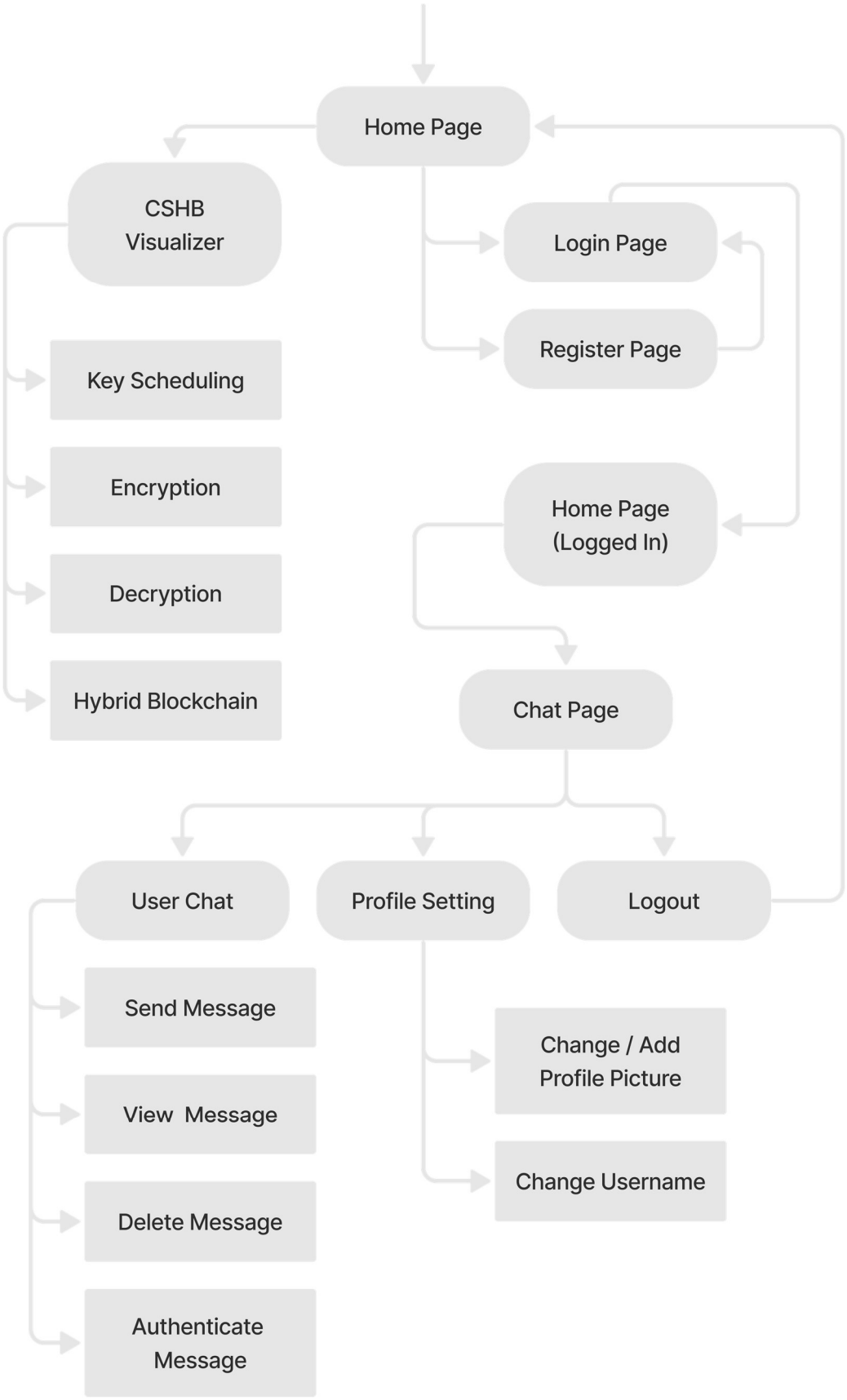
Software Design

We will be using this Crypto-Steganographic Hybrid Blockchain Model to implement a web application name *Saṁvāda* for text messaging which is not only end-to-end encrypted but also preserve integrity of data. This application ensures that all messages exchanged between users are protected against unauthorized access and manipulation. Just like CSHB Model it also employs hash-based error detection techniques to verify the integrity of messages, preventing tampering and ensuring data integrity. The hybrid blockchain algorithm used in *Saṁvāda* App securely links the hash of the previous message to the current message, making it resistant to manipulation and ensuring secure communication.

Saṁvāda App a beta version of a secure end-to-end chat application. The application is designed using React Typescript and uses Firebase as the backend, providing reliable and efficient service to its users. *Saṁvāda* App is also a Progressive Web App that can be easily installed on mobile devices. With its advanced security features, users can communicate without the fear of their data being compromised. *Saṁvāda* App is currently in beta, with ongoing improvements and updates to enhance its functionality and user experience. Along with messaging feature it also provides the CSHB Model visualizer, which depicts at each stage how the data is passed. It includes feature to view Key Scheduling of AES-512, Encryption, Decryption and working of Hybrid Blockchain.

6.1 Data Flow Diagram

The flow chart representation of the *Saṁvāda* application and web flow among various interface is given below


Home Page
CSHB Visualizer
Key Scheduling
Encryption
Decryption
Hybrid Blockchain
Login Page
Register Page
Home Page (Logged In)
Chat Page
User Chat
Send Message
View Message
Delete Message
Authenticate Message
Profile Setting
Change / Add Profile Picture
Change Username
Logout

6.2 Software Development Information

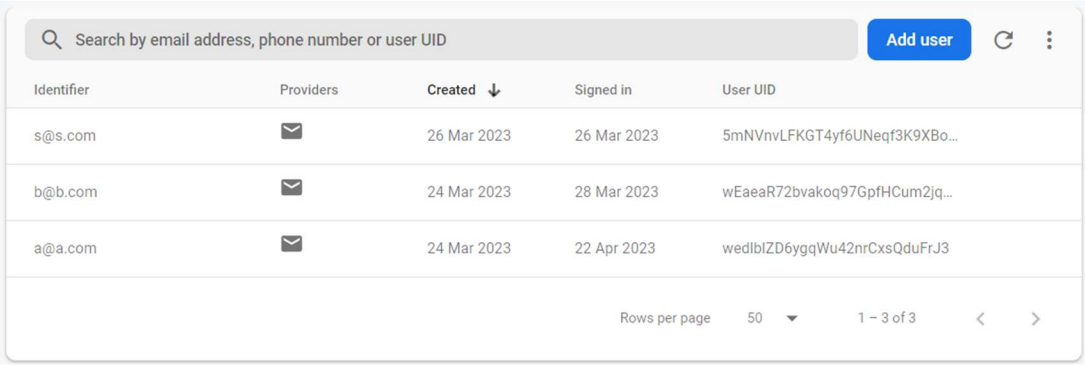
1. **Software Development Life Cycle**: Iterative Incremental Model with Continuous Integration and Continuous Deployment (CI/CD).
2. **Current Software Testing Stage**: Beta Testing with Test-Driven Development (TDD)
3. **Version Control & Management**: Github with Git branching and merging strategies, pull requests, and code review process
4. **Analysis**: Python with Jupyter Notebook, NumPy, Pandas, Seaborn, and statistical analysis techniques like hypothesis testing and regression analysis
5. **Coding**: ViteJS with TypeScript for faster development and better type safety, React for building reusable components, TailwindCSS for responsive and efficient styling, and ESLint for code quality assurance
6. **Deployment**: Github-Pages with Webpack bundling and optimization, and caching strategies for faster load times and better user experience
7. **Database & Storage**: Firebase Storage & Realtime Database for cloud-based storage and real-time synchronization, and device local storage for offline caching and data persistence
8. **Hardware System Requirements**: Minimum system requirements including processor, RAM, and storage specifications, and recommended system configurations for optimal performance and scalability.
    - CPU: Intel Core i5 or equivalent AMD processor
    - RAM: 8 GB or more
    - Storage: 256 GB SSD or higher
    - Display: 1080p or higher resolution
    - Graphics: NVIDIA or AMD dedicated graphics card with 2 GB VRAM or higher (for better performance)
    - Operating System: Windows 10 or macOS (latest version)
    - Network: Wi-Fi or Ethernet connectivity (for online features)

6.3 Database and Storage

6.3.1 Firebase Authentication

Firebase Authentication provides an easy-to-use user authentication system that helps developers to secure their web or mobile applications. In *Saṁvāda* App, Firebase Authentication is used to handle user registration and login. When a user registers in the app, Firebase Authentication creates a new user account and stores the user's credentials securely in its backend servers. In *Saṁvāda* App, users can register and login using their email and password. The password is hashed and stored securely in Firebase servers using salted hash algorithms, which ensure that passwords are never stored in plain text.

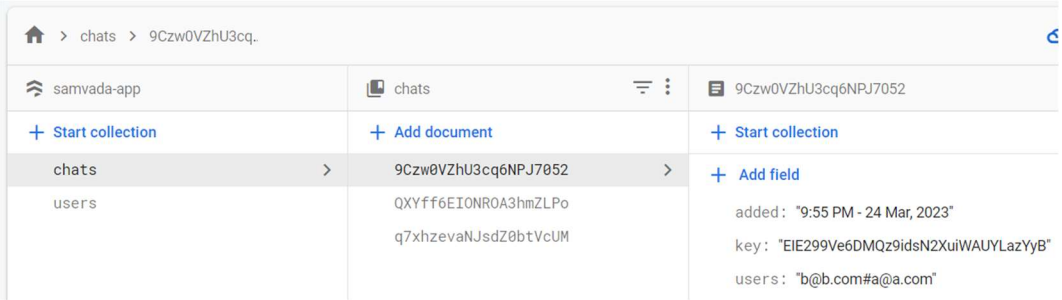
When a user logs in to the app, Firebase Authentication verifies the user's identity by checking the credentials stored in its backend servers. If the user's credentials are valid, it generates an access token, which is sent to the client-side app. This access token is used to authenticate the user for all future requests to Firebase services, ensuring that the user is authorized to access the app's resources. Firebase Authentication also provides features like two-factor authentication, account recovery, and user management tools for developers to manage their user base efficiently.



Registered User Data for Authentication

6.3.2 Firestore Database

Firebase Storage is used to store the constant keys generated by the application's AES encryption algorithm. These keys are generated dynamically for every chatroom created, and are used to encrypt and decrypt the messages. Since the keys are used for cryptography purposes, it is important to ensure their security and confidentiality. Firebase Storage provides a secure and reliable way to store these keys, ensuring that they are protected from unauthorized access and tampering. Moreover, Firebase Storage also allows for easy and efficient management of these keys, making it easy for the application to retrieve and use them as needed.



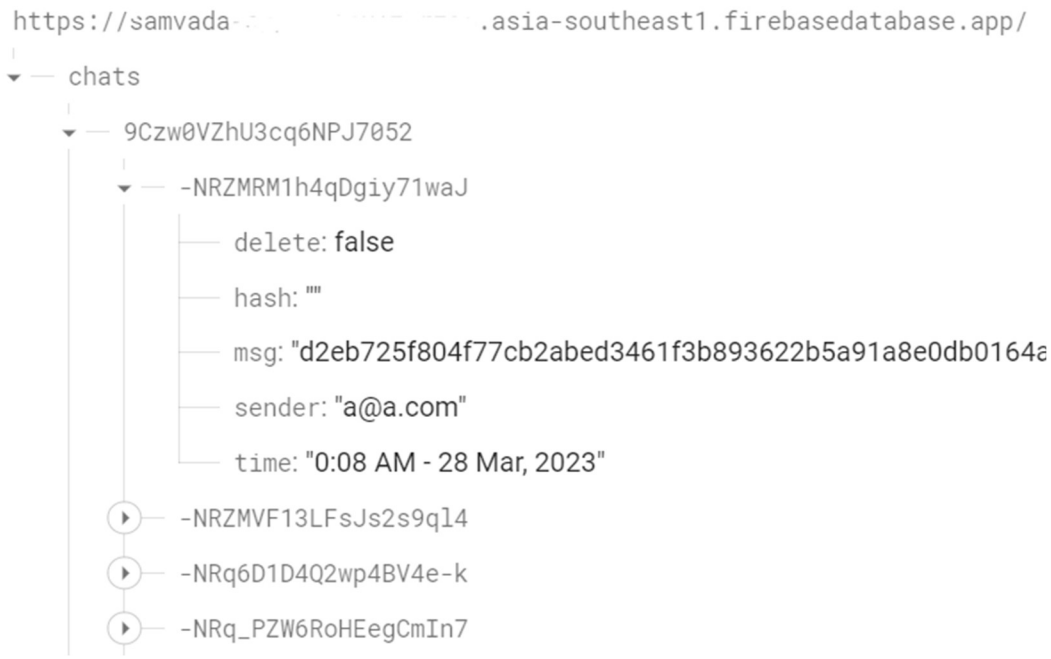
Store User and Chat data including dynamically generated key

6.3.3 Firebase Realtime Database

Firebase Realtime Database is a NoSQL cloud-hosted database that allows for real-time data synchronization between clients. It provides a fast and reliable way to store and synchronize data across multiple clients, making it a perfect choice for real-time communication applications. It uses web sockets to provide real-time updates to clients, which means that users can receive updates in real-time without needing to refresh the page.

Security Rules are a set of conditions and permissions that one can use to control access. These rules allow you to restrict access to read or write data to only the authorized users and devices. The security rules for this real-time database is given as:

```
{
 "rules": {
  "chats": {
   "$chat_id": {
    ".read": "auth != null",
    ".write": "auth != null"
   }
  }
 }
}
```

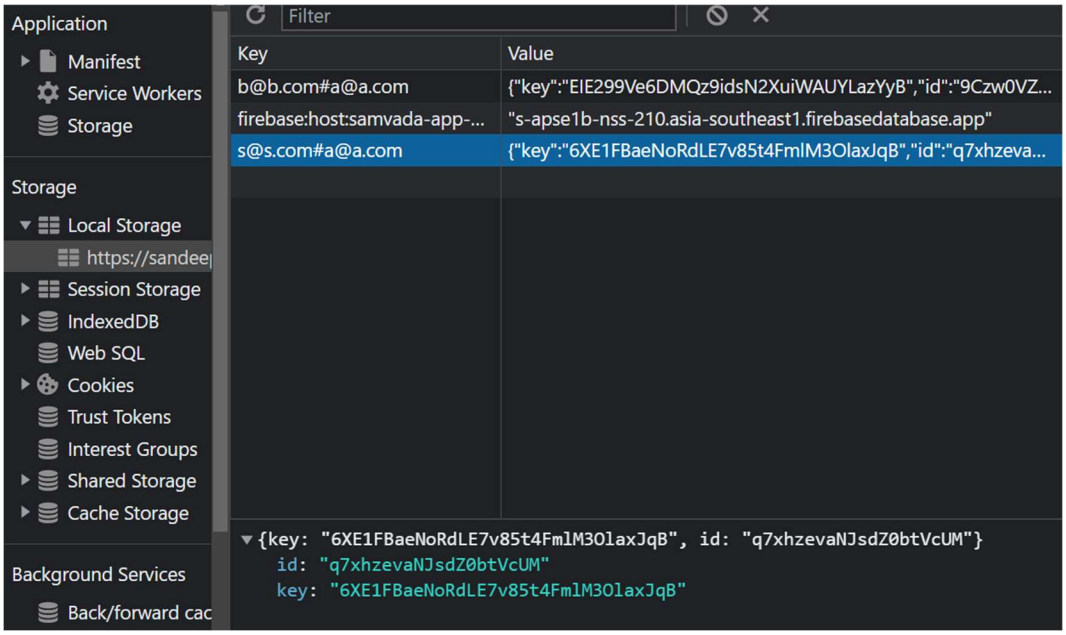


NoSQL structure for Chat Storage

6.3.3 Device Local Storage

This app also utilizes local storage in user's device in order to prevent multiple key fetching from Firestore database. This feature ensures that the key required for encryption and decryption of chat messages is securely stored on the user's device, and does not have to be frequently fetched from the database. Before the start of a session, the app matches the user's device key with the one stored in the Firestore database. If a match is found, the session proceeds with the local key, else the local

device key is updated with the key from the database. By restricting frequent key fetches, this feature helps prevent potential attacks, as attackers could potentially track and reverse engineer the chat if frequent key fetches are observed. Overall, this feature enhances the security of the app and ensures that user data is kept safe and secure. Apart from Local Storage there are many secure options available depending on which the key could be stored.



Local Storge from Chrome Developer Tools

6.4 Application Preview

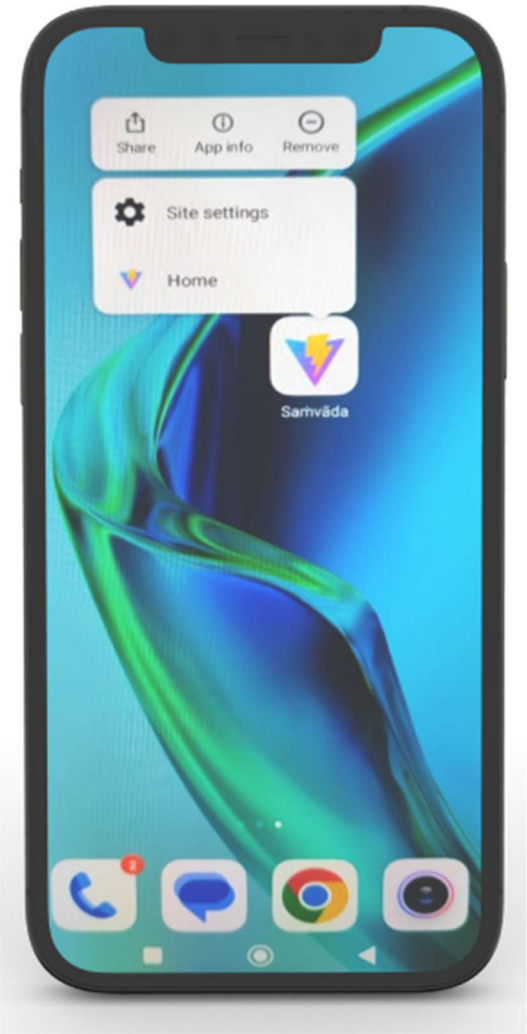
The GitHub repository of this project including data analysis and application source code is given:

1. CSHB Model: *https://github.com/sandeep-shaw10/cshb-model*
2. *Saṁvāda* App: *https://github.com/sandeep-shaw10/samvada*

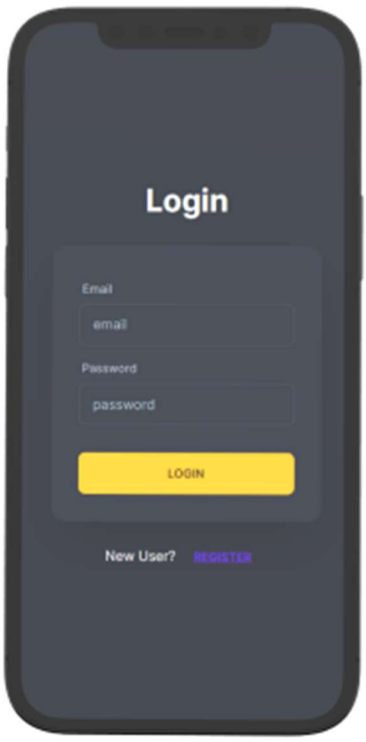
This application provides a basic but a clean and intuitive user interface that allows users to easily navigate through its various features. It is also designed to be fully responsive and compatible across all devices. This ensures that users can access the app from any device, whether it's a desktop computer, a tablet or a

mobile phone. Additionally, the app supports different themes based on the user's device theme, making it visually appealing and customizable.

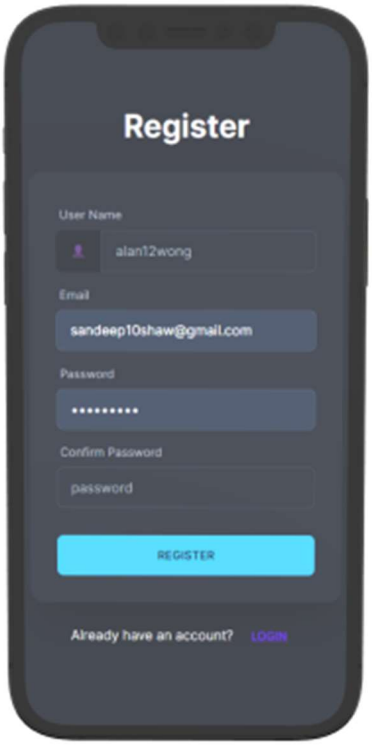
Furthermore, the app is a Progressive Web App (PWA), which means it can be installed on users' devices like a native app, allowing them to access the app's features and functionalities even when they are offline. This feature enhances the user experience by providing instant access to the app without the need to download and install it from an app store.



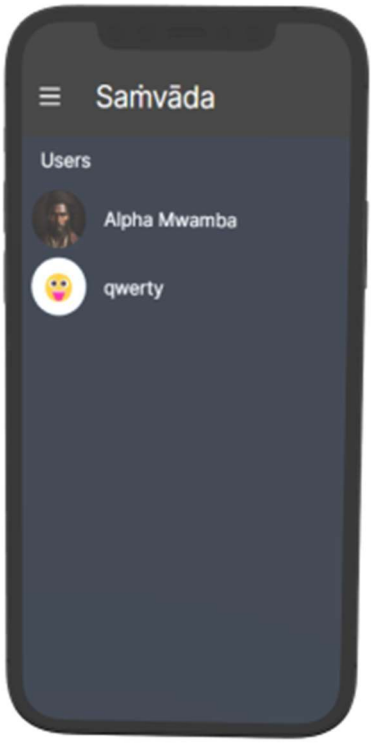
Application Preview on Mobile



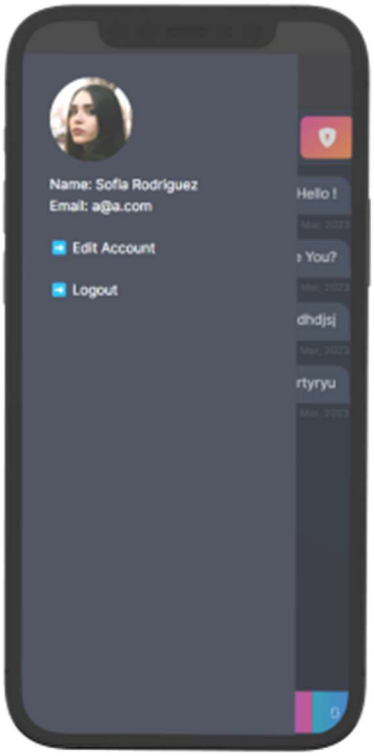
/login



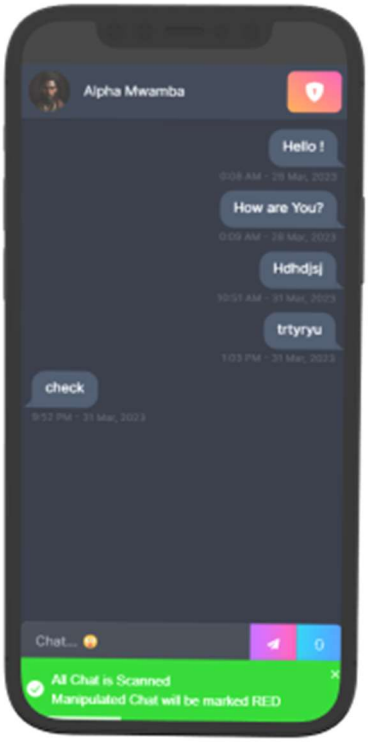
/register



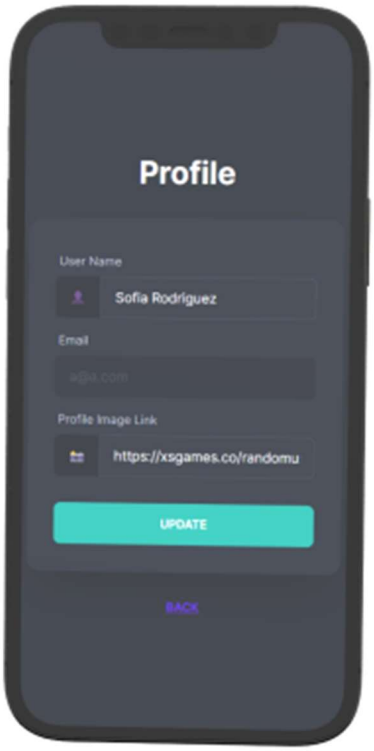
/home



/home



/home/:chatid



/profile

6.5 CSHB Model Visualizer

This visualiser is a part of the application which help us to understand how the data is processed at each stage of the CSHB Model. The visualiser provides a step-by-step breakdown of how the data is processed through the model, including how the key is generated and how each block is encrypted. This is helpful for users who are interested in understanding the technical details of the CSHB Model.

CSHB Model Visualizer Preview on Tablet

6.5.1 Key Scheduling

It demonstrates the working of how AES-512 algorithm, which expect 64 character key and outputs the state matrix, all the round keys along with the variation of keys generated at each rounds.

CSHB > Key Schedule

Ty7dWpwssyKQ3HFBTpYqDR | START

64/64

INPUT

KEY:

Ty7dWpwssyKQ3HFBTpYqDR3LMM3iFiLEMKK2Ye61vuSKd3uCstBGdWIyFmvkBVkW

STATE MATRIX:

84	121	55	100	87	112	119	115
115	121	75	81	51	72	70	66
84	112	89	113	68	82	51	76
77	77	51	105	70	105	76	69
77	75	75	50	89	101	54	49
118	117	83	75	100	51	117	67
115	116	66	71	100	87	108	121
70	109	118	107	66	86	107	87

OUTPUT

Key Scheduling Time	Rounds	Total Bytes	Throughput
400 microseconds	10 Expansions	640 State(8x8) * Rounds	1601 KBps

Select the Rounds

Round 8

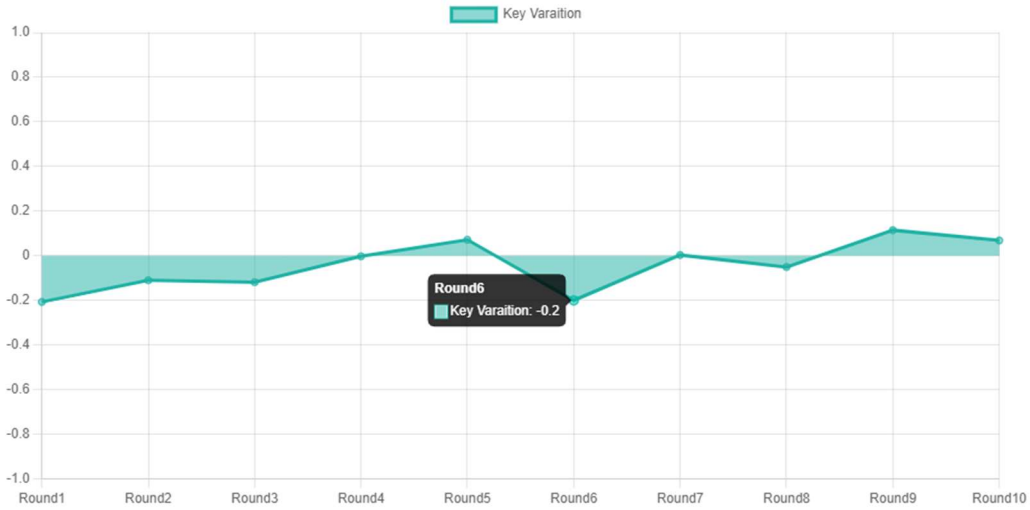
Round 1
Round 2
Round 3
Round 4
Round 5
Round 6
Round 7
Round 8
Round 9
Round 10

?ß▒▒i¾ê²t5³êö¥°ÿ)ÓòÀ▒¤h-µB^▒¨©z¦&æ

			74	63	91	138	69
43	95	159	60	242	174	91	38
197	12	27	92	211	196	71	122
100	18	5	15	147	160	63	223
31	28	238	190	234	178	116	53
179	234	244	165	176	255	41	211
242	192	15	164	104	45	181	66
94	139	168	169	122	166	38	230

KEY VARIATION (Pearson's correlation):
-0.050512609399928084

ALL ROUNDS KEY VARIATION (Pearson's correlation):



6.5.2 Crypto-Steganography Encryption

This step combines cryptography and steganography to encrypt messages into an image file using AES-512. The message and cipher text are first encrypted and then represented in binary, base encoding, and hex format. In addition, error bits are added to the encrypted data to increase its security. Once encrypted, the data is then converted into a Scalable Vector Graphic (SVG) file format, which allows it to be hidden within an image without changing the image's appearance. The use of AES-512 ensures that the encrypted data is extremely secure and difficult to crack. Overall, Crypto-Steganography Encryption is an effective method for securely transmitting confidential information.

The protected string and SVG generated after providing this value are

KEY:
iiCX6ln8rAApFog5MxWIiT6XTiNOUVHKJchN3VdkAJIFiCrd6PsPSVfnKQ0u1Bw3

MESSAGE:
This is a secret

CSHB > Crypto-Steganography Encrypt

iiCX6In8rAApFog5MxWIiT6XTiNOUVH RANDOM

64/64

This is a Secret

Enter Message

ENCRYPTION

INPUT

KEY:
iiCX6In8rAApFog5MxWIiT6XTiNOUVHKJchN3VdkAJIFiCrd6PsPSVfnKQ0u1Bw3

MESSAGE:
This is a Secret

OUTPUT: AES-512 Encrypt

BASE64
gtw3mwQV6LIRDGRAuOhzzH8lh3kx07q6XI7VbXkRYHfD0Z2hpL5TV1B40X8zde+4gAe50+jt2i7KwjCE568cVA==

HEX
82dc379b0415e8b9510c6440b8e873cc7f08877931d3baba5e5ed56d79116077c3d19da1a4be53575078d17f3375efb88007b9...

BINARY
1000001011011100001101111001101100000100000101011110100010111001010100010000110001100100010000001011100011...

Encryption Time	Throughput
1301 microseconds	13 KB of Message per seconds

OUTPUT: Stego-Image Generation

PROTECTED HEX STRING
82dc379b0415e8b9510c6440b8e873cc7f08877931d3baba5e5ed56d79116077c3d19da1a4be53575078d17f3375efb88007b9...

SVG Image

DOWNLOAD COPY

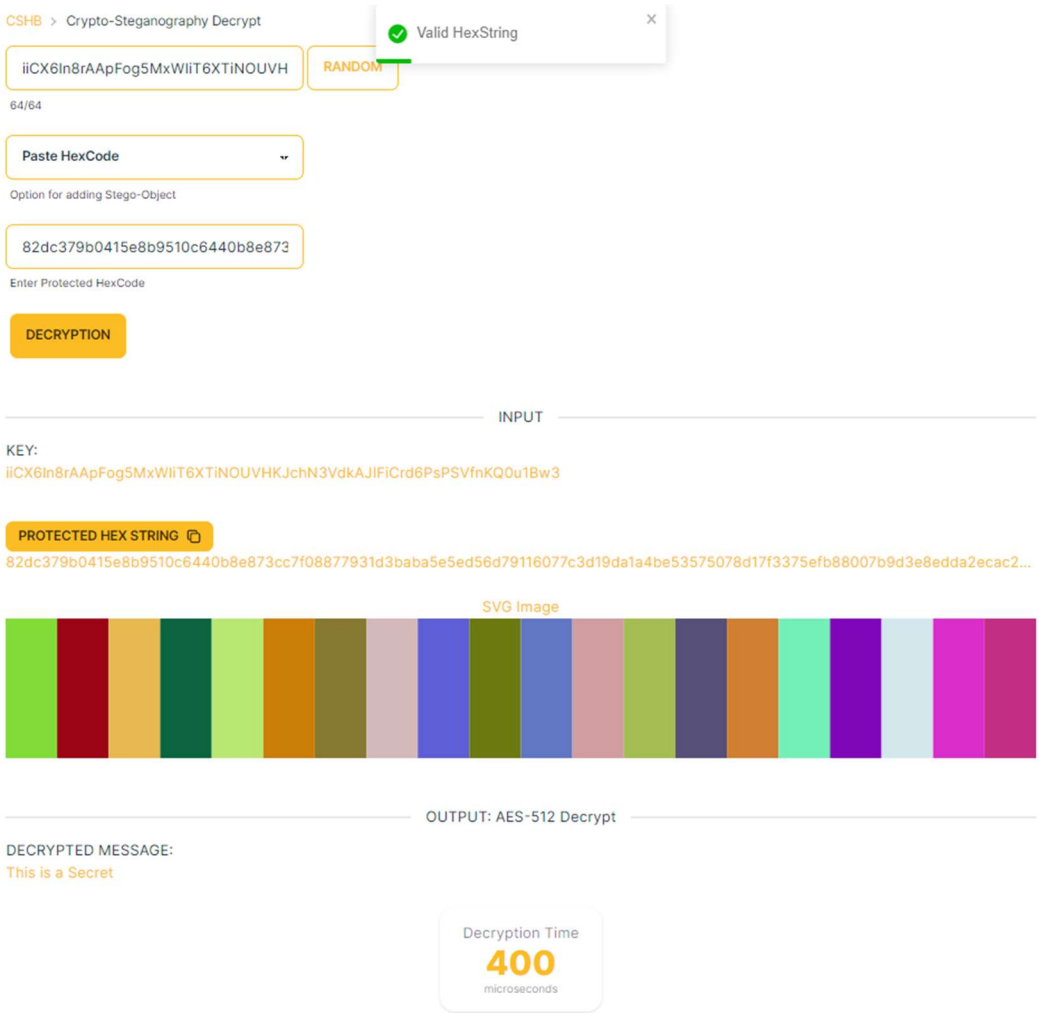
Generation Time	Encrypt Data Size	Stego-Image Size	Embedded Data
300 microseconds	0.068 KB (approx size)	1.34 KB (approx size)	5.08 % (Percentage)

6.5.3 Crypto-Steganography Decryption

Similar to the previous process, this involves the decryption process involves parsing the data from the colour codes of the SVG using regular expressions. The error bits are then checked and the data is validated before performing AES decryption to generate the original message.

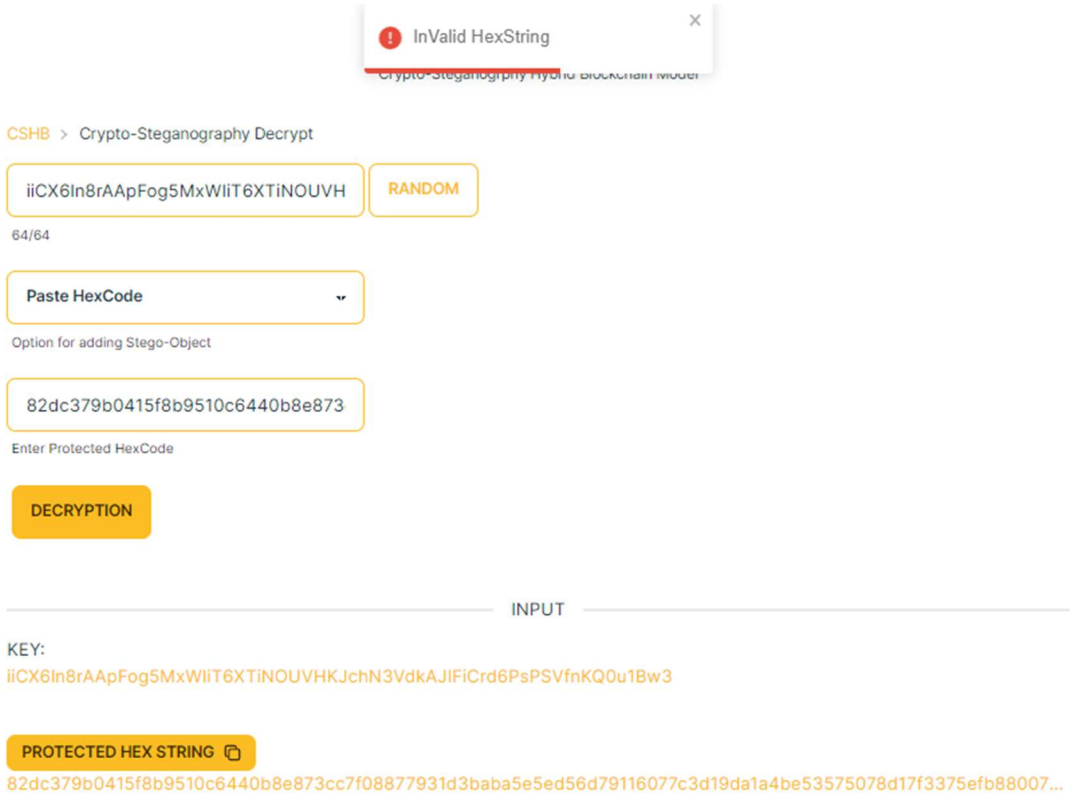
We provided the option either to upload SVG, copy-paste SVG or to paste the protected hex string.

The protected hex string generated in previous step was:
82dc379b0415e8b9510c6440b8e873cc7f08877931d3baba5e5ed56d79116077c3d19da1a4be53575078d17f3375efb88007b9d3e8edda2ecac23084e7af1c54f4c0474d



If we change the protected hex string slightly by changing “5e8” to “5f8” then:

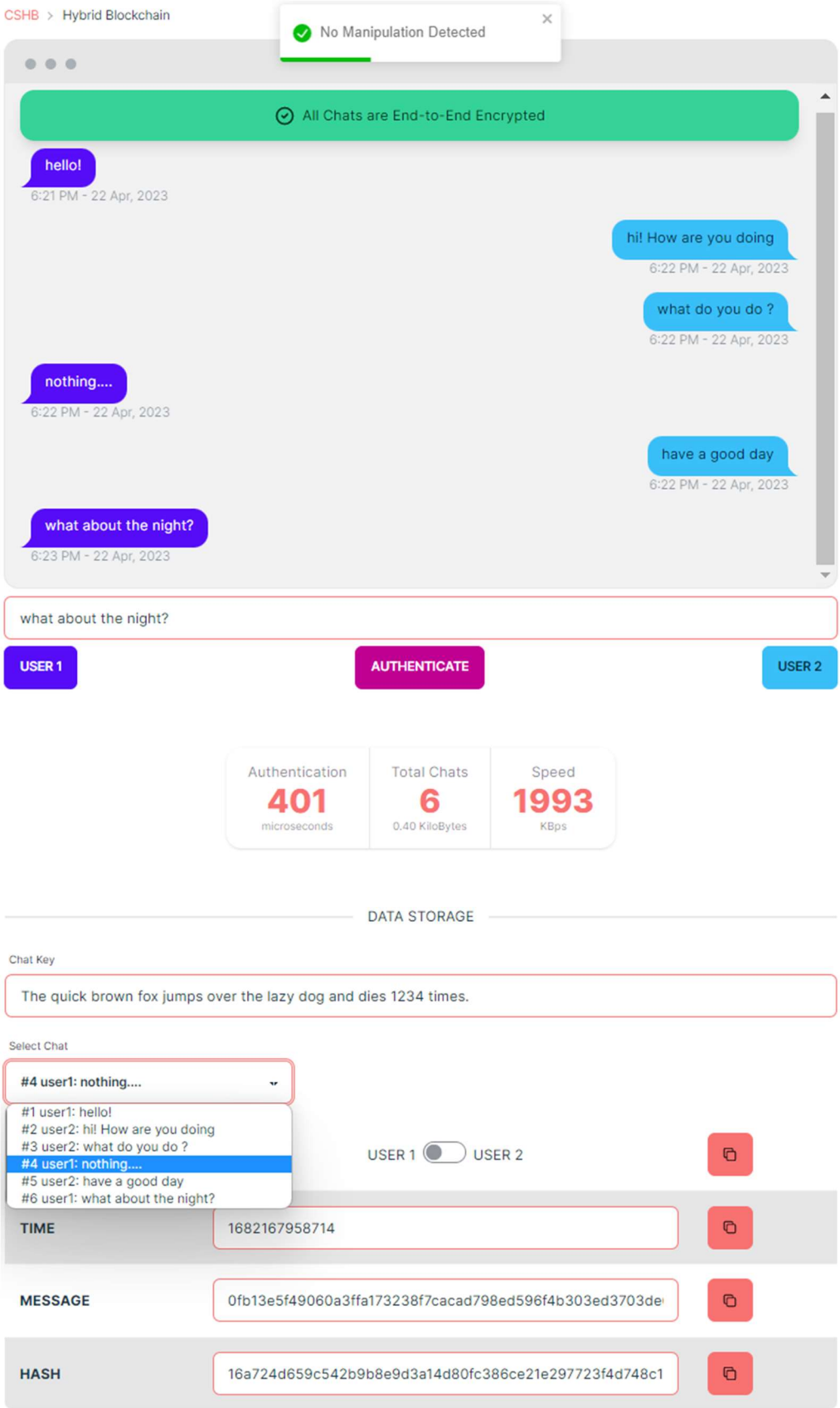
82dc379b0415f8b9510c6440b8e873cc7f08877931d3baba5e5ed56d79116077c3d
19da1a4be53575078d17f3375efb88007b9d3e8edda2ecac23084e7af1c54f4c0474d



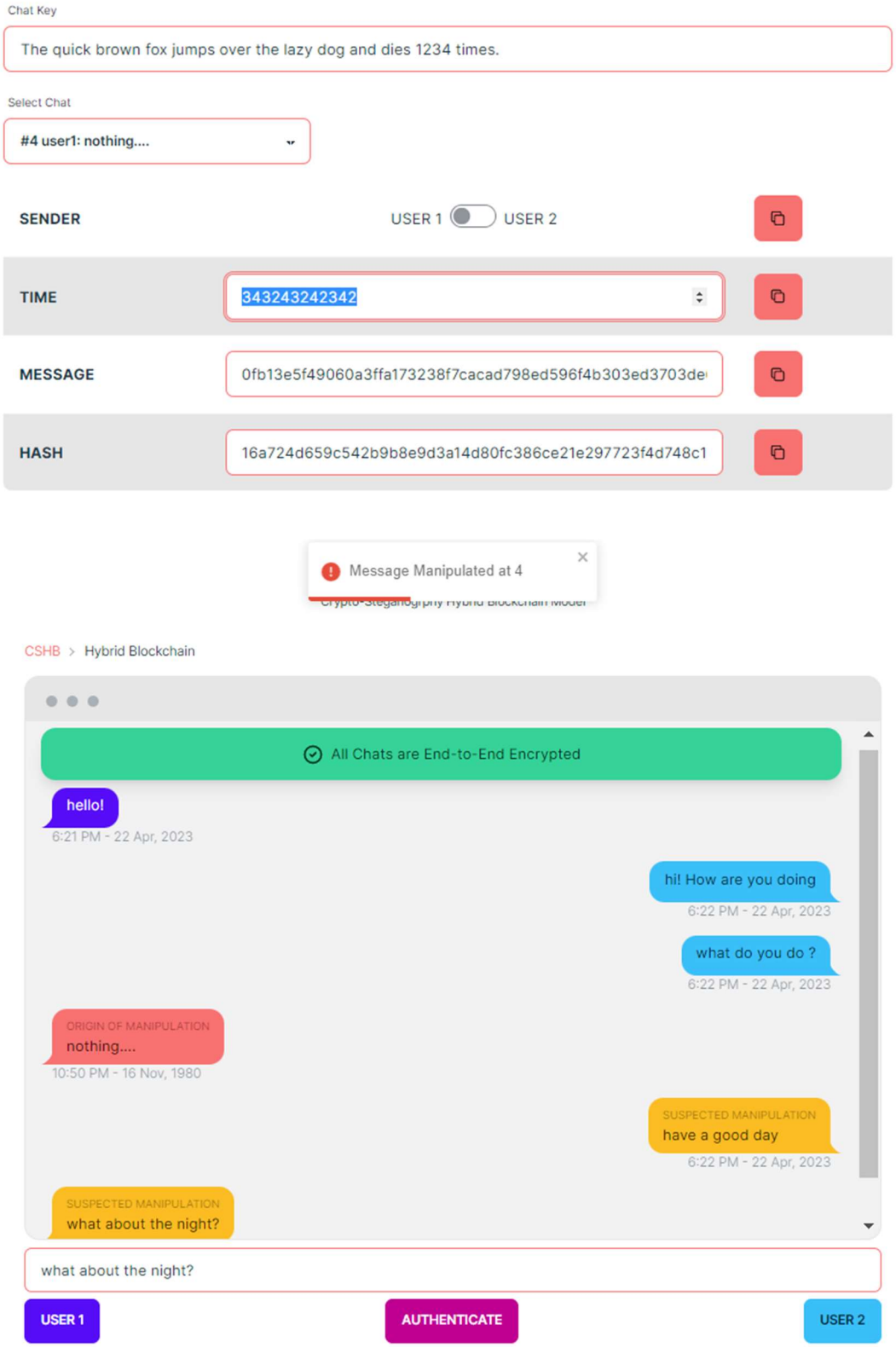
6.5.4 Hybrid Blockchain

The Hybrid Blockchain Visualizer is a tool that demonstrates how the chat is block chained in a database. This tool is designed to provide a visualization of the real process, and as such, the data is not saved and is non-persistent in nature. Whenever the page is refreshed, all data state is lost.

Through this visualizer, users can mimic the chat app and view how data is stored in the database. The tool also allows for authentication and checks the integrity of data by manipulating some data and checking how it is detected. Overall, the Hybrid Blockchain Visualizer is an important tool for understanding the process of block chaining and its use in chat applications.


CSHB > Hybrid Blockchain
No Manipulation Detected
All Chats are End-to-End Encrypted
hello!
6:21 PM - 22 Apr, 2023
hi! How are you doing
6:22 PM - 22 Apr, 2023
what do you do ?
6:22 PM - 22 Apr, 2023
nothing....
6:22 PM - 22 Apr, 2023
have a good day
6:22 PM - 22 Apr, 2023
what about the night?
6:23 PM - 22 Apr, 2023
what about the night?
USER 1
AUTHENTICATE
USER 2
Authentication
401
microseconds
Total Chats
6
0.40 KiloBytes
Speed
1993
KBps
DATA STORAGE
Chat Key
The quick brown fox jumps over the lazy dog and dies 1234 times.
Select Chat
#4 user1: nothing....
#1 user1: hello!
#2 user2: hi! How are you doing
#3 user2: what do you do ?
#4 user1: nothing....
#5 user2: have a good day
#6 user1: what about the night?
USER 1
USER 2
TIME
1682167958714
MESSAGE
0fb13e5f49060a3ffa173238f7cacad798ed596f4b303ed3703de
HASH
16a724d659c542b9b8e9d3a14d80fc386ce21e297723f4d748c1

If we try to change the time of the 4th chat and click on authenticate then it will detect the origin of manipulation.



Chapter 7

Conclusion

The *Saṁvāda* app provides a secure platform for communication, making it a valuable addition to the market of chat applications. The app demonstrates the potential of combining different technologies and techniques to create a secure and efficient chat application.

In conclusion, this project provided a comprehensive understanding of several crucial concepts such as AES algorithm, steganography, hooks in React, and Firebase security rules. We were successful in implementing the AES algorithm from scratch and developing Python and Typescript libraries for standard AES along with AES-512. The project also provided insights into improving the working of steganography by making it dynamic and compressive. However, we faced challenges in implementing dynamic LSB steganography due to its impact on the application's speed, leading us to use SVG, which proved faster and less susceptible to noise and cropping.

Furthermore, the project taught us about constant key storing, which was an essential aspect of the model, and we successfully overcame this challenge using local device storage. Overall, the experience of experimenting and analysing the data, gaining insights, and applying them to the project was invaluable and has equipped us for future research. This project provided a clear roadmap for carrying out research work and has helped us to gain a better understanding of the stages involved in research.

Chapter 8

Future Scope

There exist some of the potential areas that can be explored to enhance the functionality and security of the app. With further development and refinement, the *Saṁvāda* app has the potential to become a reliable and secure communication tool for individuals and organizations alike.

1. Adding more features to the app, such as sending emojis and multimedia support, to make the chat experience more engaging and expressive.
2. Further optimizing the *Saṁvāda* app and reducing the application size by developing a specific native app for Android and iOS devices to enhance user experience.
3. Implementing better strategies for storing the dynamically generated key and developing a more efficient approach to detect the validity of the key.
4. Exploring more efficient ways to detect errors using error bits to improve the data validation process.
5. Making efforts to increase the key variation of AES-512 to make it at par with AES-128 in terms of security.

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