A

Industry Oriented Mini Project Report

on

ELLIPTICAL CURVE BASED HYBRID CRYPTOGRAPHY SYSTEM

A report submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

By

Vishwas Gajawada

(20EG105411)

Racharla Sai Chaitanya

(20EG105437)

Abhinav Molugu

(20EG105426)

Under The Guidance Of

B. Ravinder Reddy

Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANURAG UNIVERSITY

VENKATAPUR– 500088

TELANGANA

YEAR 2023-2024

DECLARATION

I Here declare that the Report entitled **Elliptical Curve Based Hybrid Cryptography System** submitted for the award of Bachelor of technology Degree is my own work and the Report has not formed the basis for the award of any degree , diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma

Vishwas Gajawada
(20EG105411)

Racharla Sai Chaitanya
(20EG105437)

Abhinav Molugu
(20EG105426)

ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

CERTIFICATE

This is to certify that the Report / dissertation entitled **Elliptical Curve Based Hybrid Cryptography System** that is being submitted by **Mr. Vishwas Gajawada** bearing the Hall ticket number **20EG105411, Mr. Racharla Sai Chaitanya** bearing the Hall ticket number **20EG105437, Mr. Abhinav Molugu** bearing the Hall ticket number **20EG105426** in partial fulfilment for the award of B.Tech in Computer Science And Engineering_ to the Anurag University is a record of bonafide work carried out by him under our guidance and supervision.

The results embodied in this Report have not been submitted to any other university or Institute for the award of any degree or diploma

Signature of The Supervisor
B. Ravinder Reddy, CSE
Assistant Professor

Head Of The Department

External Examiner

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **B. Ravinder Reddy** for his constant encouragement and inspiring guidance without which this project could not have been completed. His/her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. His/her patience, guidance and encouragement made this project possible.

We would like acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy** , Dean, Department of CSE ,Anurag University. We also express my deep sense of gratitude to **Dr. V V S S S Balaram** , Acadamic co-ordinator. **Dr .Pallam Ravi** Project Co-ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated me during the crucial stage our project work.

We would like express my special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for their encouragement and timely support in our B.Tech program.

Vishwas Gajawada
(20EG105411)

Racharla Sai Chaitanya
(20EG105437)

Abhinav Molugu
(20EG105426)

ABSTRACT

In the ever-changing digital landscape, ensuring the confidentiality and integrity of sensitive information is paramount. This paper addresses the pressing need for robust data protection by introducing hybrid cryptography to bolster the security of internet communications. The primary objective is to thwart unauthorized access and tampering of data.

This study focuses on evaluating key performance indicators, specifically Encryption and Decryption times, as well as key-generation time. These metrics are crucial for assessing the effectiveness of the proposed scheme. The approach outlined combines two cutting-edge cryptographic techniques, Elliptical Curve Cryptography (ECC) and the Diffie-Hellman algorithm, both widely acknowledged for their robust security features. Through the integration of these advanced cryptographic methods, this research aims to make a meaningful contribution to the ongoing evolution of hybrid encryption techniques. Furthermore, it provides a pragmatic solution for strengthening the security of sensitive data transmitted over communication channels and stored within systems.

In conclusion, this research emphasizes the urgent need for heightened data security in the evolving digital landscape. Our hybrid cryptography approach, combining ECC and Diffie-Hellman, demonstrates promising results in protecting sensitive information. This work advances hybrid encryption, enhancing data security for transmission and storage.

Keywords: Diffie-Hellmen, Hybrid Cryptography, Elliptical Curve Cryptography, Key-Generation.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1.INTRODUCTION

Cryptography, in its essence, is the science and art of securing communication and information through the use of codes and algorithms. Its primary objective is to protect the confidentiality, integrity, and authenticity of data in transit or at rest. In today's technologically advanced world, where information is constantly transmitted over networks, the importance of cryptography cannot be overstated. The significance of cryptography has grown exponentially with the proliferation of digital technology and the internet. Advanced technology has made data communication faster and more efficient, but it has also introduced new vulnerabilities. Without proper encryption, sensitive information such as personal data, financial transactions, and confidential business communications could be intercepted and exploited by malicious actors.

There exist two fundamental types of cryptography: symmetric and asymmetric cryptography. Symmetric cryptography, also referred to as secret-key (referred to as say, $\boldsymbol{S_k}$) cryptography, employs a single $\boldsymbol{S_k}$ for both the encryption and decryption processes. It is crucial to maintain the confidentiality of $\boldsymbol{S_k}$ between the sender and receiver. Despite the efficiency and speed of symmetric cryptography, it encounters a challenge when securely sharing the secret key, $\boldsymbol{S_k}$ between parties, as any compromise of the key could result in the compromise of all encrypted data.

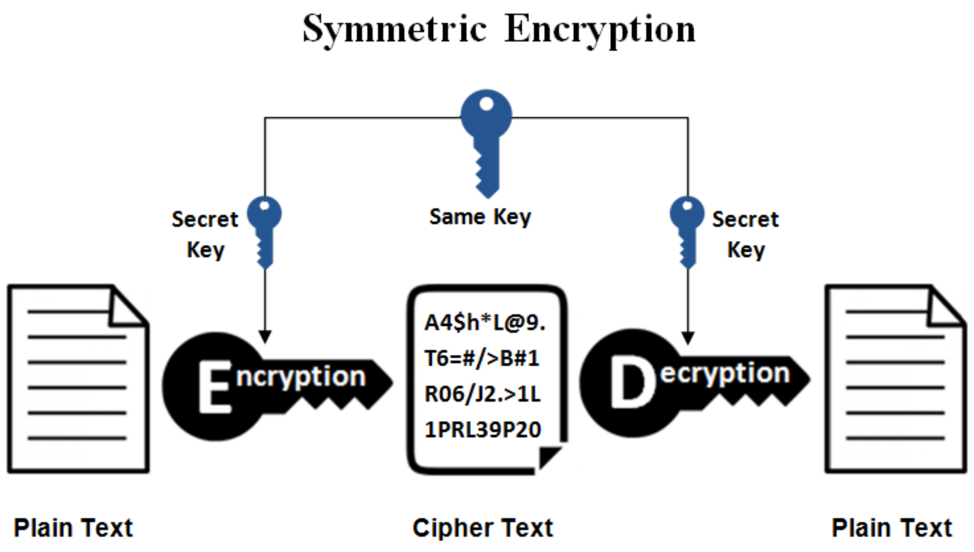


Figure 1.1. Symmetric Encryption

In contrast, asymmetric cryptography employs a key pair—a public key (referred to as say, $\boldsymbol{P_u}$) for encryption and a private key (referred to as say, $\boldsymbol{P_r}$) for decryption. The $\boldsymbol{P_u}$ is shareable without restrictions, while the $\boldsymbol{P_r}$ is kept confidential. This method tackles the key distribution challenge but may involve computational intensity. The security of asymmetric cryptography hinges on the mathematical complexity of specific problems, such as the factorization of large numbers.

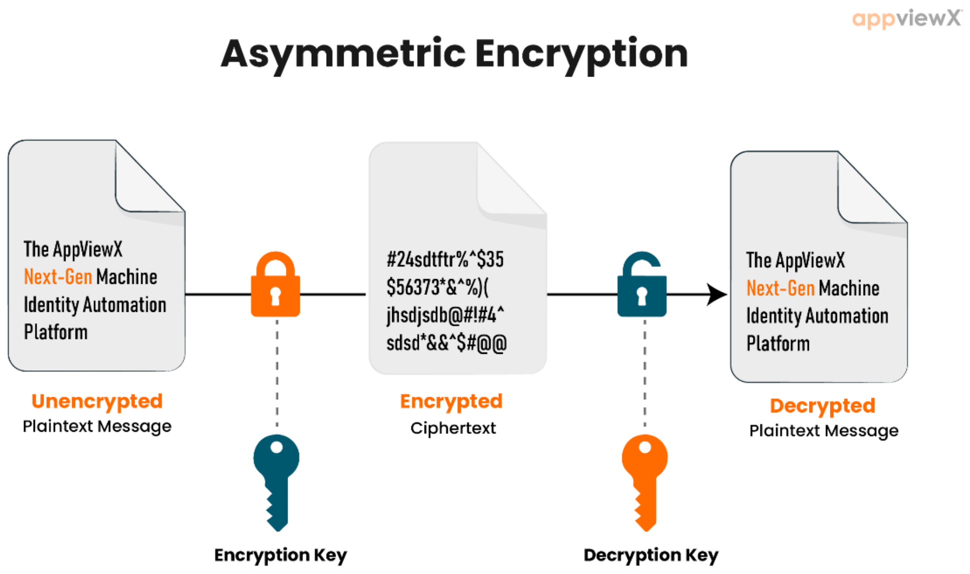

appviewX
Asymmetric Encryption
The AppViewX Next-Gen Machine Identity Automation Platform
Unencrypted
Plaintext Message
#24sdtftr%^$35 $56373*&^%)(jhsdjsdb@#!#4^ sdsd*&&^$#@@
Encrypted
Ciphertext
The AppViewX Next-Gen Machine Identity Automation Platform
Decrypted
Plaintext Message
Encryption Key
Decryption Key

Figure 1.2. Asymmetric Encryption

However, both symmetric and asymmetric cryptography can be vulnerable to attacks if the keys are revealed to an intruder. In the case of symmetric cryptography, if the $\boldsymbol{S_k}$ falls into the wrong hands, all encrypted data can be decrypted. Asymmetric cryptography relies on the secrecy of the $\boldsymbol{P_r}$, and if it is compromised, an attacker can impersonate the key holder. To address these vulnerabilities, cryptographic protocols like the Diffie-Hellman key exchange algorithm were introduced. Diffie-Hellman allows two parties to securely exchange encryption keys over an unsecured channel without directly transmitting the keys. While it provides a secure key exchange mechanism, human intervention in key management can still introduce risks.

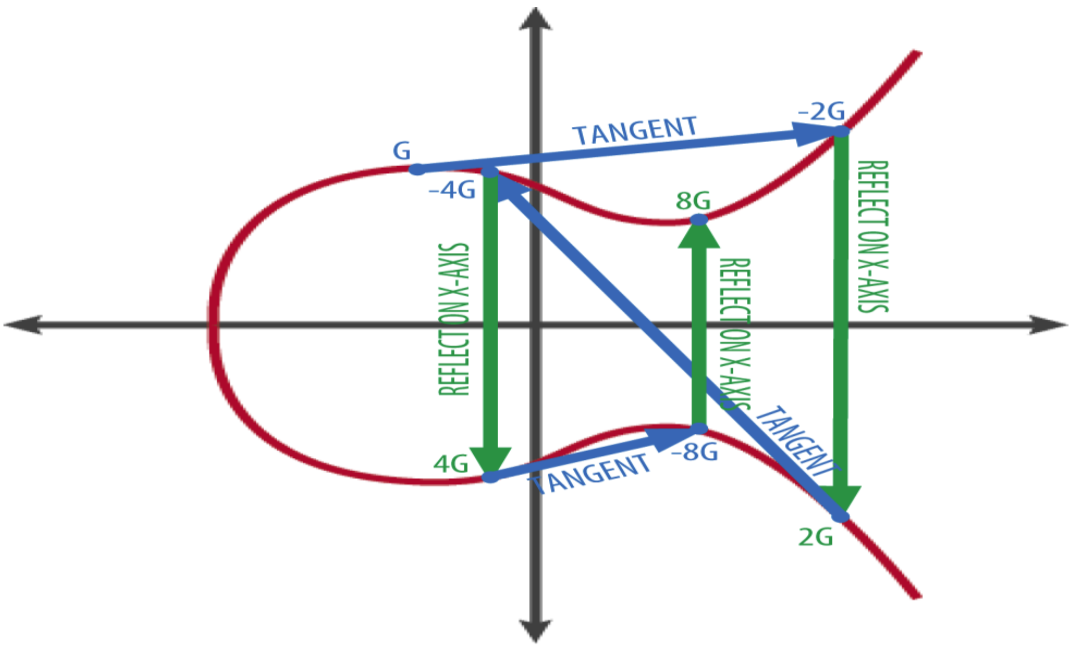


Figure 1.3. Elliptical Curve

To mitigate these risks, a more robust approach involves the integration of elliptic curve cryptography (ECC) into the hybrid system. ECC offers strong security with shorter key lengths, making it less susceptible to attacks. By combining the Diffie-Hellman key exchange with ECC, a highly secure and efficient hybrid cryptography system can be established, providing a robust defense against potential threats.

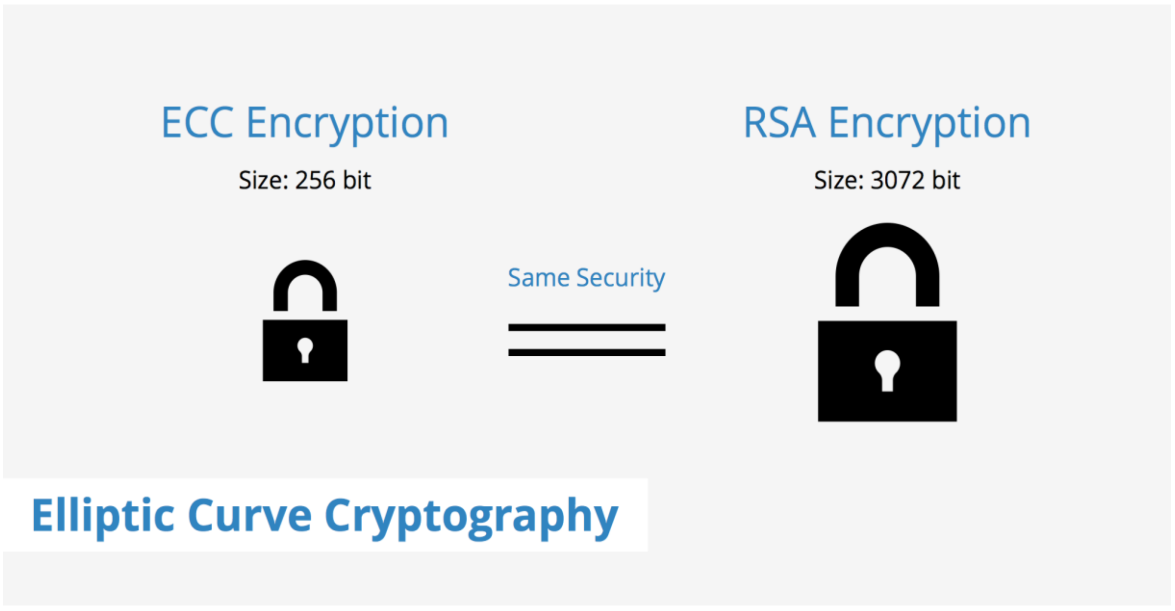


Figure 1.4. Security Comparison

1.1. Rivest-Shamir-Adleman(RSA) with Diffie-Hellman

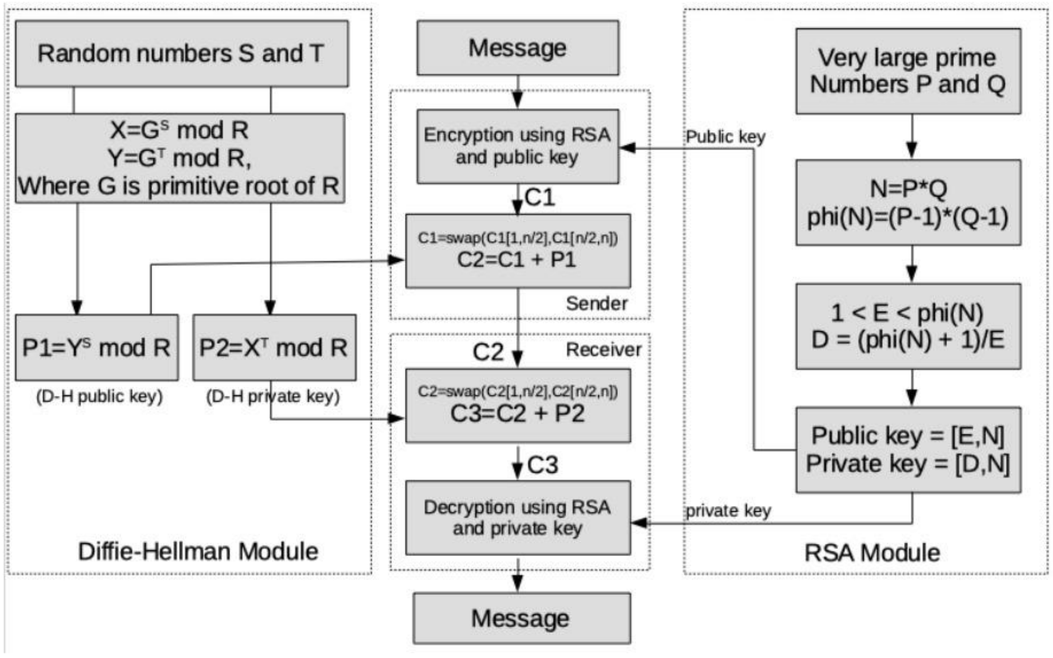


Figure 1.5. Existing Method Program Flow

Now-a-days , the exchange of sensitive information over the internet has become an integral part of our daily lives. Whether it's sending confidential emails, conducting financial transactions, or protecting personal data, ensuring the security and privacy of these communications is paramount. Two fundamental cryptographic algorithms, RSA and Diffie-Hellman, play crucial roles in safeguarding these digital interactions.

RSA and Diffie-Hellman are asymmetric encryption techniques that have revolutionized the way we protect data during transmission and establish secure channels for communication. These algorithms, each with its unique strengths and applications, have become the cornerstones of modern cryptography, enabling secure connections and safeguarding sensitive information against prying eyes

1.2. Numerical Example:

1. Diffie-Hellman Key Exchange:

- Alice and Bob want to securely exchange a symmetric encryption key.
- They agree on public parameters:
  - Prime number (p) = 23
  - Primitive root modulo p (g) = 5
- They each generate their private keys:
  - Alice's private key (a) = 6
  - Bob's private key (b) = 15
- They compute their public keys:
  - Alice's public key (A) = (g^a) mod p = (5^6) mod 23 = 8
  - Bob's public key (B) = (g^b) mod p = (5^15) mod 23 = 19
- They exchange public keys.
- They independently compute the shared secret:
  - Alice: (B^a) mod p = (19^6) mod 23 = 2
  - Bob: (A^b) mod p = (8^15) mod 23 = 2
- Now, Alice and Bob both have the shared secret key (2) to use for symmetric encryption.

2. Rivest-Shamir-Adleman(RSA) Encryption and Decryption:

RSA Encryption and Decryption:

Alice wants to send the plaintext "Vishwas" to Bob using RSA.

Bob has a public key (e, n):

e = 17

n = 253 (product of two large prime numbers).

Alice knows Bob's public key, so she encrypts the message "Vishwas" using Bob's public key:

Convert "Vishwas" to a numerical representation (for example, by using ASCII values): "86 105 115 104 119 97 115".

Encrypt each numerical value separately:

Ciphertext C1 = (86^17) mod 253

Ciphertext C2 = (105^17) mod 253

Ciphertext C3 = (115^17) mod 253

Ciphertext C4 = (104^17) mod 253

Ciphertext C5 = (119^17) mod 253

Ciphertext C6 = (97^17) mod 253

Ciphertext C7 = (115^17) mod 253

The resulting ciphertext values (C1, C2, C3, C4, C5, C6, C7) are sent to Bob.

Bob, who has the corresponding private key (d) for decryption, receives the ciphertext and decrypts it:

Decrypt each ciphertext value separately:

Recovered numerical value R1 = (C1^d) mod 253

Recovered numerical value R2 = (C2^d) mod 253

Recovered numerical value R3 = (C3^d) mod 253

Recovered numerical value R4 = (C4^d) mod 253

Recovered numerical value R5 = (C5^d) mod 253

Recovered numerical value R6 = (C6^d) mod 253

Recovered numerical value R7 = (C7^d) mod 253

Convert these numerical values back to their ASCII representations, resulting in the message "Vishwas."

2. LITERATURE REVIEW

Nguyen Minh Trung, et al [1] a researcher in Vietnam, has created two secure systems using complex math on Elliptic Curves for encrypting messages. These systems rely on the difficulty of solving a math problem, ensuring the security of information. **Gupta et al [2]** suggested a new way to keep internet messages safe by mixing two methods called RSA and Diffie-Hellman. It's like having a secret code (RSA) to lock and unlock messages, and another method (Diffie-Hellman) to safely share the keys used for the code. This makes online communication more secure.

Hazra et al [3] created a double-layered security method for hiding information in files, like text or images. They used special codes to lock the files, making it hard for anyone to peek inside. It's like having two secret layers – one to lock the file, and another to share a key for unlocking it securely. This helps keep sensitive information safe when sent over the internet. **Vidhya et al [4]** proposed a new way to make computer codes safer by combining two existing methods called RSA and ECC. This new approach creates stronger codes that are more resistant to certain types of attacks. It's like having extra layers of protection for your digital information. Their experiments show that this new method works better and is faster than the older method (RSA) alone.

Ermatita and others[5] worked on a way to keep medical images safe by using two techniques: one for hiding the information (AES) and another for exchanging secret keys (Diffie-Hellman). This method helps protect sensitive medical data and ensures that only authorized people can access it. The results show it works well in keeping medical images secure. **Avaestro et al[6]** combined two methods called Modified Diffie Hellman (MDH) and RSA. MDH makes sure the keys used for communication are exchanged safely, and RSA encrypts and decrypts messages. It's like sending secret letters – one person changes the way they write, and the other person knows how to read it. This makes it harder for someone else to understand the message. Their new method is faster and more secure than the old one.

[7]Yassien's project investigates the role of encryption algorithms in enhancing data security within cloud computing, comparing symmetric and asymmetric approaches. It highlights the trade-offs between performance and security, offering insights into the selection of encryption methods for specific cloud applications. The study underscores the importance of safeguarding data in the era of cloud-based services. **Mahibel's et al[8]** proposed method focuses on enhancing the security of key exchange in public channels using Elliptic Curve Cryptography (ECC). It introduces a novel protocol that leverages the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP) to ensure secure and authentic key transport between two parties, mitigating the vulnerabilities associated with man-in-the-middle attacks in traditional Diffie-Hellman key exchange.

Swapnil's et al[9] project explores the critical importance of data security in the age of internet communication and provides an in-depth analysis of both symmetric and asymmetric cryptography techniques. The proposed architecture appears to be robust, emphasizing security and efficiency. However, successful implementation and user adherence will be key factors in its overall effectiveness. **Sadiq's et al[10]** project highlights the growing significance of cloud computing for data storage due to its cost-effectiveness and convenience. However, it addresses the security concerns associated with data breaches in the cloud. The focus on hybrid cryptography as a solution is promising, but the paper identifies gaps in user authentication and algorithm implementation that need attention. Overall, it provides valuable insights into the challenges and solutions related to securing data in the cloud.

One widely recognized hybrid cryptography algorithm is RSA (Rivest-Shamir-Adleman) combined with AES (Advanced Encryption Standard). RSA is used for key exchange and AES for data encryption. This combination offers the security of RSA for key distribution and the efficiency of AES for data encryption. For a detailed discussion, you can refer to the IEEE paper titled "Hybrid Cryptography: RSA and AES for Secure Data Communication" by **S. Gupta et al[2].**

2.1. Comparison of Existing Methods.

S.No.	Author(s)	Method	Advantages	Disadvantages
1.	Sachin rana, Satarupa biswas and Anushika	-ECC algorithm -Diffie – Hellman	Reduced Encryption and Decryption time	Large KeySize
2.	Dr. Vivek Kapoor And Rahul Yadav	-Rsa Algorithm -Des Algorithm -SHA128	-128bit key -Secure data Transmission -Data Integrity	Performance Impact and need high computational power
3.	Arprit Agarwal and Gunjan Patnakar	-RSA Algorithm -Diffie-hellman	-Scalability -Highly Effiecient -Secure Key Sharing	-Require Technical expertise and users need
4.	Y Alkady, M.I. Habib and R.Y	-RSA Algorithm -ECC algorithm	-Better Performancein terms of computation time and the size of text.	-High Complexity, might need more resources
5.	Prakash kuppuswamy and Sayeed Q.Y.Al-khalidi	-Symmetric key Algorithm -Linear block Cipher Algorithm	-Better Performance. - Enhanced Security -Resistance to Attacks	Increased Implementation challenges, Security of Key Exchange. Potential vulnerabilities

3. Elliptical Curve and Diffie-Hellman Using Aes Encryption

The Hybrid scheme proposed is a combination of two cryptographic algorithms that are Elliptical Curve and Diffie-Hellmen which provide better security than the existing method. Here ECC is used for the generation of the public and the private variables as of RSA it has a large key size which lead to more storage and more time taking process so ECC is used to reduce both storage as well as the time. Alongside with the ECC we are using the Diffie - Hellmen to produce a Shared key (referred to as a $\boldsymbol{sh_k}$) between both the parties in a most secure way possible. In this the Message/Data is Encrypted using the AES and the AES 'iv' is encrypted using the ECC based encryption.

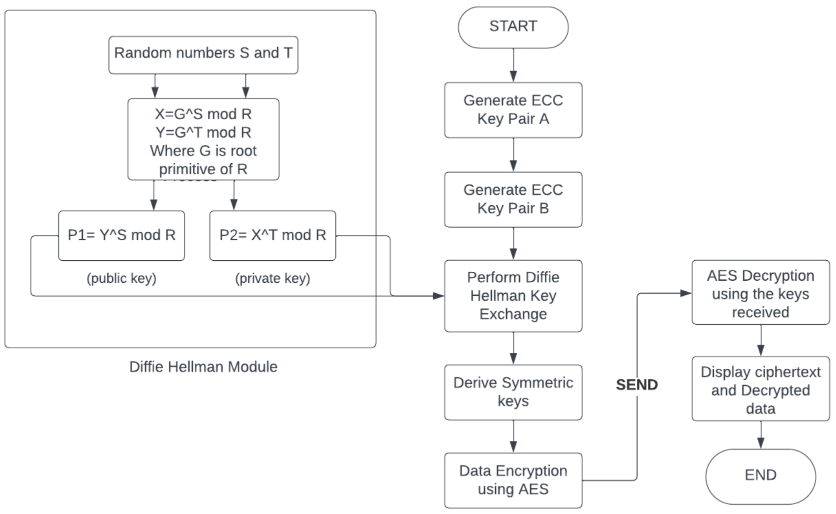


Figure 3.1. Communication using the proposed method

Diffie-Hellman and ECC, being distinct modules, collaborate to enhance efficiency and establish a secure connection for communication between two parties. The flowchart above illustrates the integration of both processes. The left side of the diagram depicts the execution of the Diffie-Hellman algorithm, generating secured $\boldsymbol{sh_k}$ used in subsequent encryption. The center part delineates the flow, commencing with ECC key pair generation. Within this process, encryption and decryption occur using AES, which utilizes the derived symmetric key.

3.1. Encryption Process

Step 1: **Key Generation**

ECC key pairs are independently generated for both parties, A and B**.**

Step 2: **Key Exchange**

Party A's $\boldsymbol{P_r}$ and Party B's $\boldsymbol{P_u}$ are utilized in the Diffie-Hellman key exchange process, yielding shared_key_A(referred to as $\boldsymbol{sh_{ka}}$). Simultaneously, Party B's $\boldsymbol{P_r}$ and Party A's $\boldsymbol{P_u}$ are employed in the Diffie-Hellman key exchange, resulting in shared_key_A(referred to as $\boldsymbol{sh_{kb}}$).

Step 3: **Key Derivation**

The HMAC Key Derivation Function (HKDF) is applied to derive symmetric keys, namely symmetric_key_A and symmetric_key_B, from the $\boldsymbol{sh_{ka}}$ and $\boldsymbol{sh_{kb}}$ obtained through the Diffie-Hellman key exchange process.

3.2.Decryption Process

Step 1: **Key Exchange (for Party B)**

To initiate the decryption process, Party A's public key and Party B's private key are employed in the Diffie-Hellman key exchange. This operation yields $\boldsymbol{sh_{kb}}$.

Step 2: **Key Derivation (for Party B)**

The HMAC Key Derivation Function (HKDF) is subsequently applied to derive the symmetric key (symmetric_key_B) from the $\boldsymbol{sh_{kb}}$.

Step 3: **Decryption:**

The encrypted data (ciphertext) is provided as input. Initializing an AES cipher in CFB mode with symmetric_key_B and the same Initialization Vector (IV) used during encryption (zeros), the ciphertext undergoes decryption using the AES cipher. The resultant plaintext is then extracted and printed

3.3.Numerical Example for the above process:

Step 1: **Key Generation**

1. Generate Private Key A and Private Key B:
    a. Private Key A (a): 123
    b. Private Key B (b): 456

2. Compute Public Key A and Public Key B (Point Multiplication):
    a. Public Key A (PA) = a * G (Generator Point)
    b. Public Key B (PB) = b * G

Step 2: **Key Exchange**

1. Calculate the Shared Secret for A and B (Point Multiplication):
    i. Shared Key A (SA) = a * PB
    ii. Shared Key B (SB) = b * PA

Step 3: **Key Derivation**

1) Derive Symmetric Keys using HKDF:

i) Symmetric Key A (KA) = HKDF(SA, SHA256, length=32, info='symmetric key')
ii) Symmetric Key B (KB) = HKDF(SB, SHA256, length=32, info='symmetric key')

2) Numerical Example of HKDF Process:

i) Using the HKDF process with SHA-256, derive the symmetric keys:
ii) Shared Secret SA (Example Value): `0x36187E0AB15F7CF46EC3B5A20DE650E0B8A7A631A94DE6B358D586ED0EE74D18

iii) Extract Step (HMAC Key Extraction):

iv) Salt: None (no salt is used in this example).

v) Info: 'symmetric key' (additional data for key derivation).

vi) Extracted Key (PRK) = HMAC(SHA-256, None, SA) = \`0xB0A67213D5E044A2A10D7DE219D60D47A1575ABE2ABD56D030 4E11A816D7A75F9\`

vii) Expand Step (Key Expansion):

viii) Expand the derived pseudorandom key into the desired length (32 bytes).

ix) Derived Symmetric Key A (KA) = HMAC(SHA-256, PRK, Info='symmetric key') = \`0x9434D570FAB3E954CB7C7F DF01FC0E2706BBD27DE28A71DFFEF91D7622F4A26D\`

3.4. Encryption and Decryption

1) Let's encrypt and decrypt a message:

i) Plaintext (P): "HELLO" (represented as bytes)

3.4.1. Encryption

2) Encrypt the plaintext using Symmetric Key A (AES Encryption in CFB Mode):

i) Initialization Vector (IV): All zeros (for the first block)

ii) AES Encryption:

iii) Ciphertext (C) = AES_Encrypt(P, KA, IV)

3) Numerical Example of AES Encryption (Block 1):

i) Initialization:

i. Plaintext Block (P[0:3]): \`0x48454C4C4F4F\`

ii. IV: \`0x00000000000000000000000000000000\`

iii. Symmetric Key A (KA): \`0x9434D570FAB3E954CB7C7FDF01FC0E2706BBD27DE 28A71DFFEF91D7622F4A26D

ii) AES Encryption (CFB Mode):

i. XOR the first plaintext block (P[0:3]) with the IV:
ii. XOR1 = `P[0:3] XOR IV = 0x48454C4C4F4F XOR 0x000000000000 = 0x48454C4C4F4F`
iii. Encrypt XOR1 using AES and KA:
iv. AES_Encrypt(XOR1, KA) = `0xACEFFD72B8767DAF3E883703E577E2F6`

3.4.2. Decryption

1) **Decrypt the ciphertext using Symmetric Key B (AES Decryption in CFB Mode):**

   i. AES Decryption:
   ii. Decrypted Text (P') = AES_Decrypt(C, KB, I)

2) **Numerical Example of AES Decryption (Block 1):**

   i. Initialization:
   ii. Ciphertext Block (C[0:3]): `0xACEFFD72B8767DAF3E883703E577E2F6`
   iii. IV: `0x00000000000000000000000000000000`
   iv. Symmetric Key B (KB): `0x9434D570FAB3E954CB7C7FDF01FC0E2706BBD27DE28A71DFFEF91D7622F4A26D`
   v. AES Decryption (CFB Mode):
   vi. Decrypt the first ciphertext block (C[0:3]) using AES and KB:
   vii. AES_Decrypt(C[0:3], KB) = `0x48454C4C4F4F`
   viii. XOR the decrypted result with the IV to obtain the first plaintext block:
   ix. XOR1 = `Decrypted_Block XOR IV = 0x48454C4C4F4F XOR 0x000000000000 = 0x48454C4C4F4F`

3) **Block 2 (Encryption and Decryption):**

   i. Continue the encryption and decryption process for each subsequent block, using the previous ciphertext block as the XOR input for encryption and the previous ciphertext as input.

4.IMPLEMENTATION

4.1. Packages Used

1. **cryptography.hazmat.primitives.serialization:** This package provides methods for serializing private and public keys, including encoding formats (e.g., PEM).
2. **cryptography.hazmat.primitives.hashes:** This package is used for cryptographic hashing algorithms. In the script, it's used with SHA256 for key derivation.
3. **cryptography.hazmat.primitives.asymmetric.ec:** It deals with elliptic curve cryptography (ECC) and provides methods for generating ECC key pairs, performing key exchange, and more.
4. **cryptography.hazmat.primitives.kdf.hkdf:** This package is used for key derivation functions (HKDF) to derive a symmetric key from a shared secret.
5. **cryptography.hazmat.primitives.ciphers:** It includes methods for working with cryptographic ciphers, like AES. In the script, it's used for encryption and decryption.
6. **cryptography.hazmat.backends.default_backend:** The default_backend package is used to specify the default cryptographic backend for the operations.

4.2. Attributes

1. **generate_ecc_key_pair():** Generates ECC key pairs, consisting of private and public keys using the SECP256R1 elliptic curve.
2. **perform_diffie_hellman():** Performs the Diffie-Hellman key exchange using ECC, producing a shared key.
3. **derive_symmetric_key():** Derives a symmetric key using the shared key and HKDF with SHA256.
4. **encrypt_data():** Encrypts input data using AES encryption in CFB (Cipher Feedback) mode.
5. **decrypt_data():** Decrypts ciphertext using AES decryption in CFB mode.

4.3. Variables

2. **private_key_A and public_key_A:** ECC private and public key pair for party A.
3. **private_key_B and public_key_B:** ECC private and public key pair for party B.
4. **shared_key_A and shared_key_B:** Shared secret keys derived through the Diffie-Hellman key exchange for parties A and B, respectively.
5. **symmetric_key_A and symmetric_key_B:** Symmetric keys derived from the shared secrets for parties A and B, respectively.
6. **plaintext:** The input data (string) to be encrypted.
7. **ciphertext:** The encrypted data generated from the plaintext using the symmetric key.
8. **decrypted_text:** The decrypted data obtained from the ciphertext using the symmetric key.
9. **iv:** Initialization vector (IV) used in AES encryption and decryption.
10. **enc_start_time, enc_end_time, enc_elapsed_time:** Variables to measure the encryption time.
11. **dec_start_time, dec_end_time, dec_elapsed_time:** Variables to measure the decryption time.

4.4. Sample Code

#Key Pair Generation

```
def generate_ecc_key_pair():
    private_key = ec.generate_private_key(ec.SECP256R1(), default_backend())
    public_key = private_key.public_key()
    return private_key, public_key
```

#diffie-hellman Key Exchange

```
def perform_diffie_hellman(private_key, peer_public_key):
    shared_key = private_key.exchange(ec.ECDH(), peer_public_key)
    return shared_key
```

```
#Derivation of the symmetric key
def derive_symmetric_key(shared_key):
    derived_key = HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b'symmetric key',
        backend=default_backend()
    ).derive(shared_key)
    return derived_key
def encrypt_data(plaintext, symmetric_key):
    iv = b'\x00' * 16
cipher = Cipher(algorithms.AES(symmetric_key), modes.CFB(iv),
backend=default_backend())
encryptor = cipher.encryptor()
ciphertext = encryptor.update(plaintext) + encryptor.finalize()
return ciphertext

#Decryption
def decrypt_data(ciphertext, symmetric_key):
    iv = b'\x00' * 16
    cipher = Cipher(algorithms.AES(symmetric_key), modes.CFB(iv),
            backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    return plaintext

# Display Keys
'''print("Private Key A:", private_key_A.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncryption()
).decode('utf-8'))
```

```
print("Public Key A:", public_key_A.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
).decode('utf-8'))

print("Private Key B:", private_key_B.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncryption()
).decode('utf-8'))
print("Public Key B:", public_key_B.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
).decode('utf-8'))'''
```

4.5. Dataset:

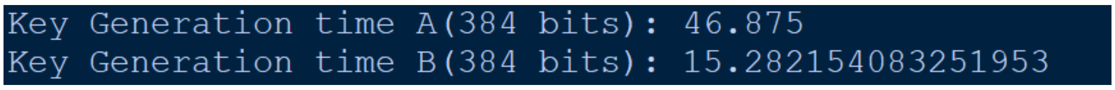
The below following dataset consists of the plaintexts that can be encrypted And decrypted using the ecc and Diffie-hellman and there are various data lengths are Present like starting from the 128 bit to the 7186 bit and the method is tested with this Data so as, to check the accuracy of the algorithm
https://www.kaggle.com/datasets/alihusseinalwan/dataset-aes-des-bluefishetc/

1. **Input Text:** this consists of the data inputs of a text which is taken as a plain text by the ecc_encrypt.py file to process the plain text and as to generate the cipher text. The message/input text size is used for the comparison of the encryption and the decryption time with already existing rsa algorithm.

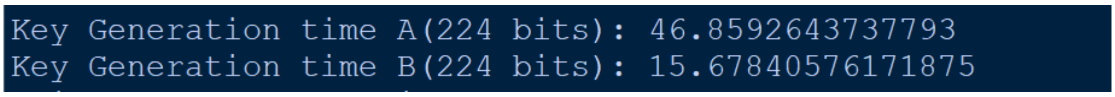
2. **Input Text Size:** This field consists of the size of the input text i.e the size of the plain text that the algorithm will process through it. This will be helpfull to determine the objectives of the algorithm and also will be helpful in evolving the performance of the algorithm proposed.

5.EXPERMENT RESULTS

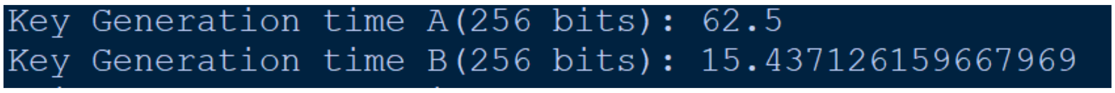
5.1. Experiment Screen Shots



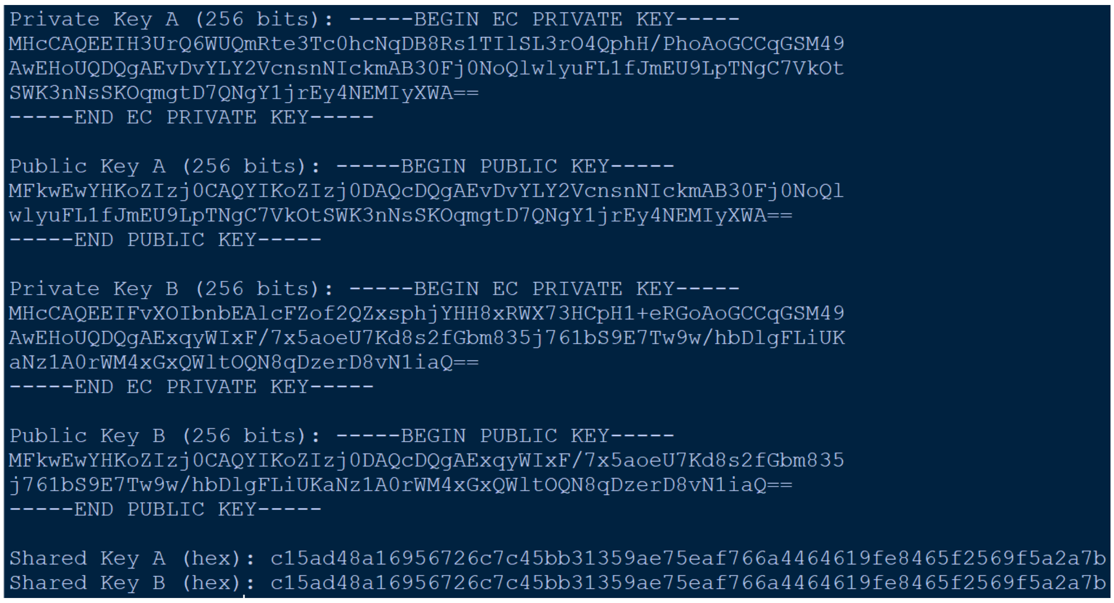
```
Key Generation time A(384 bits): 46.875
Key Generation time B(384 bits): 15.282154083251953
```



```
Key Generation time A(224 bits): 46.8592643737793
Key Generation time B(224 bits): 15.67840576171875
```



```
Key Generation time A(256 bits): 62.5
Key Generation time B(256 bits): 15.437126159667969
```



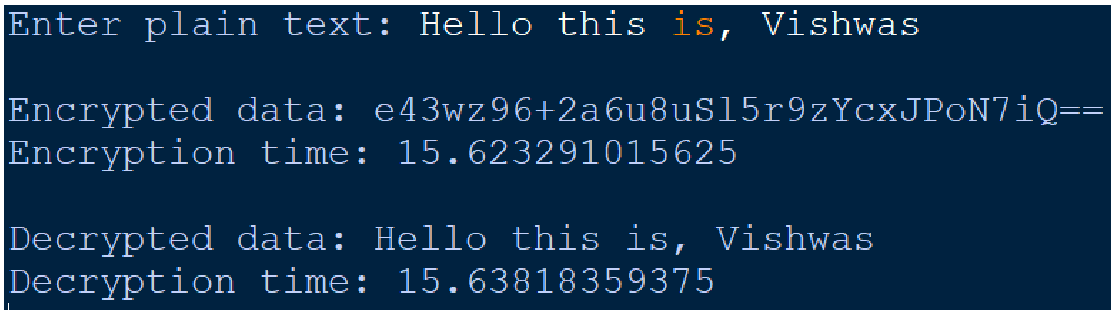
```
Private Key A (256 bits): -----BEGIN EC PRIVATE KEY-----
MHcCAQEEIH3UrQ6WUQmRte3Tc0hcNqDB8Rs1TIlSL3rO4QphH/PhoAoGCCqGSM49
AwEHoUQDQgAEvDvYLY2VcnsnNIckmAB30Fj0NoQlwlyuFL1fJmEU9LpTNgC7VkOt
SWK3nNsSKOqmgtD7QNgY1jrEy4NEMIyXWA==
-----END EC PRIVATE KEY-----

Public Key A (256 bits): -----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEvDvYLY2VcnsnNIckmAB30Fj0NoQl
wlyuFL1fJmEU9LpTNgC7VkOtSWK3nNsSKOqmgtD7QNgY1jrEy4NEMIyXWA==
-----END PUBLIC KEY-----

Private Key B (256 bits): -----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFvXOIbnbEAlcFZof2QZxsphjYHH8xRWX73HCpH1+eRGoAoGCCqGSM49
AwEHoUQDQgAExqyWIxF/7x5aoeU7Kd8s2fGbm835j761bS9E7Tw9w/hbDlgFLiUK
aNz1A0rWM4xGxQWltOQN8qDzerD8vN1iaQ==
-----END EC PRIVATE KEY-----

Public Key B (256 bits): -----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExqyWIxF/7x5aoeU7Kd8s2fGbm835
j761bS9E7Tw9w/hbDlgFLiUKaNz1A0rWM4xGxQWltOQN8qDzerD8vN1iaQ==
-----END PUBLIC KEY-----

Shared Key A (hex): c15ad48a16956726c7c45bb31359ae75eaf766a4464619fe8465f2569f5a2a7b
Shared Key B (hex): c15ad48a16956726c7c45bb31359ae75eaf766a4464619fe8465f2569f5a2a7b
```



```
Enter plain text: Hello this is, Vishwas

Encrypted data: e43wz96+2a6u8uSl5r9zYcxJPoN7iQ==
Encryption time: 15.623291015625

Decrypted data: Hello this is, Vishwas
Decryption time: 15.63818359375
```

5.2. Parameters

The ECC (Elliptic Curve Cryptography) and RSA (Rivest-Shamir-Adleman) are two widely used public-key encryption algorithms, and they differ significantly in terms of key sizes, security, and performance. In this discussion, we'll explore the key size considerations for ECC and RSA, and how they impact the security and efficiency of these encryption methods.

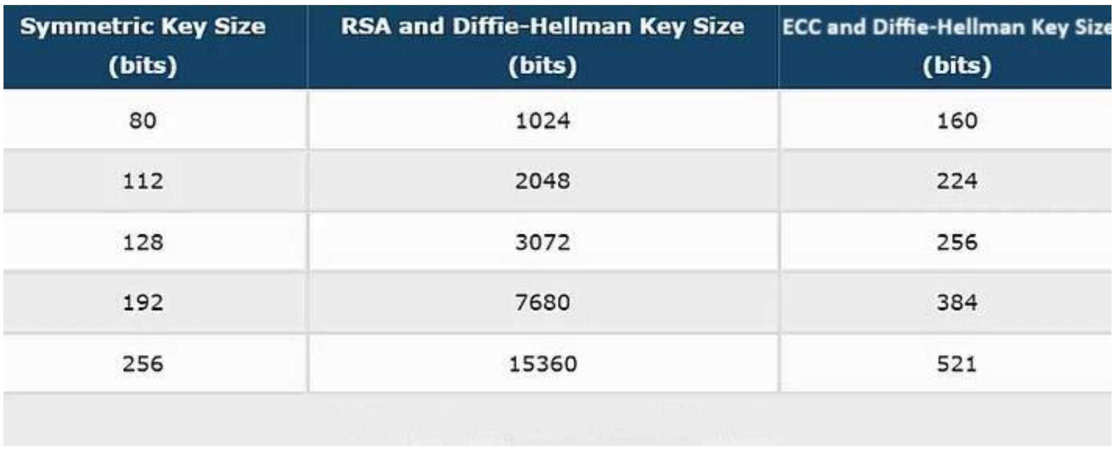
5.2.1 Elliptic Curve Cryptography(ECC):

Key Size: ECC offers strong security with relatively shorter key sizes compared to RSA. It achieves this by exploiting the mathematical properties of elliptic curves. For instance, a 256-bit ECC key can provide the same level of security as a 3072-bit RSA key.

Security: ECC is considered highly secure, and its security is based on the difficulty of the elliptic curve discrete logarithm problem. ECC's smaller key sizes make it more resistant to brute force and quantum attacks compared to RSA. In the post-quantum era, ECC is a strong contender for maintaining secure communication.

Performance: ECC is known for its efficiency in terms of both key generation and encryption/decryption processes. This efficiency is particularly advantageous in resource-constrained environments, such as mobile devices and IoT devices, where computational resources are limited.

Bandwidth Efficiency: ECC keys are shorter, which leads to smaller encrypted data sizes. This can be beneficial for bandwidth-constrained applications, as it reduces the overhead of transmitting cryptographic data.



Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	ECC and Diffie-Hellman Key Siz (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 5.1. Key Sizes Comparison

5.2.2. Rivest-Shamir-Adleman (RSA):

Key Size: RSA traditionally requires longer key sizes to achieve the same level of security compared to ECC. For example, a 2048-bit RSA key is roughly equivalent in security to a 224-bit ECC key. As computational power increases and new attacks emerge, the recommended RSA key sizes have been growing over time.

Security: RSA relies on the difficulty of factoring large composite numbers. While it is still secure, it faces challenges from advances in factoring algorithms, and larger key sizes are necessary to maintain security. In a post-quantum world, RSA might become less viable as it is more susceptible to quantum attacks compared to ECC.

Performance: RSA is computationally more intensive for both key generation and encryption/decryption compared to ECC. This can be a concern for systems where computational efficiency is crucial.

Legacy Support: RSA is widely supported and remains in use in many legacy systems. Transitioning to ECC can be challenging for organizations that have already deployed RSA-based encryption.

S.No.	Key Length		Time (s)	
1.	**RSA**	**ECC**	**RSA**	**ECC**
2.	1024	160	1.61	0.8
3.	2048	224	7.47	0.061
4.	3072	256	9.80	0.077
5.	7680	384	133.30	0.082

Table 5.2. Time Comparison

The choice between ECC and RSA depends on various factors, including the desired security level, performance requirements, available resources, and the need for legacy system support. ECC is a strong candidate for modern cryptographic applications, offering strong security with smaller key sizes and better efficiency, while RSA is still used in legacy systems and some traditional applications. However, as quantum computing advances, ECC's inherent security advantages may make it a more future-proof choice in the long run.

6. Discussion Of Results

Analysis and comparisons were conducted between the conventional AES algorithm and the newly proposed hybrid algorithm, assessing various parameters such as key size, message size, encryption, and decryption time. The experimentation utilized an Intel i5 processor with 16GB main memory and another setup with an i5 processor featuring 8GB of memory. The implementation of these algorithms was carried out using Python IDLE 3.10.

6.1. Comparison Encryption Time and Message size

The table below shows the data collected for different message sizes input to the existing and the proposed algorithm.

Table 6.1. Encryption time comparison for different message sizes

Message Size In Chars In ms	**Encryption Time** Rsa with Diffie-hellman (Existing)	**Encryption Time** Ecc with Diffie-hellman (Proposed)
100	91	10
200	93	14
500	97	27
1000	102	39

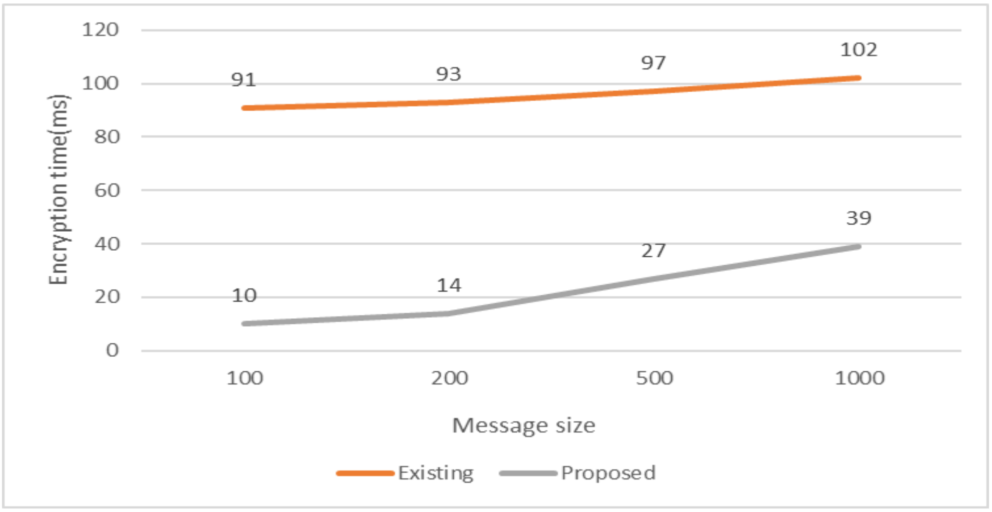


Figure 6.1. Performance of ECC Algorithm on encryption

6.2. Comparison Decryption Time and Message size

The table below shows the data collected for different message sizes input to the existing and the proposed algorithm.

Table 6.2. Decryption time comparison for different message sizes

Message Size In Chars In ms	**Decryption Time** RSA with Diffie-hellmn (Existing)	**Decryption Time** Ecc with Diffie-hellman (Proposed)
100	81	9
200	82	22
500	84	29
1000	91	31

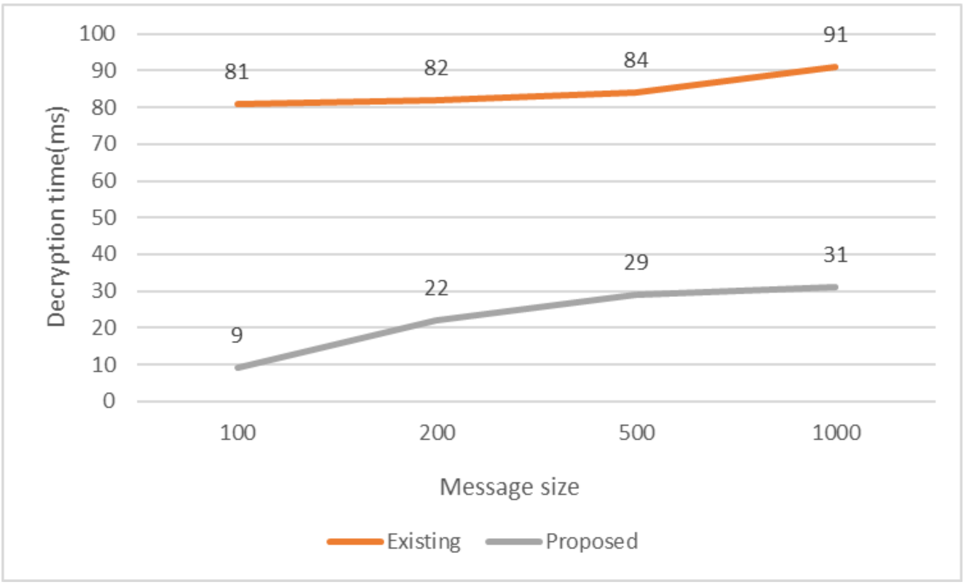


Figure 6.2. Performance of ECC Algorithm on decryption

7.CONCLUSION

In conclusion, our proposed hybrid cryptography system, combining Elliptical Curve Cryptography (ECC) and Diffie-Hellman, presents a robust and efficient solution for secure data communication. The unique integration of ECC for key generation, addressing the limitations of larger key sizes in traditional RSA, ensures a balance between enhanced security, reduced storage requirements, and faster processing times. By incorporating the Diffie-Hellman key exchange protocol, we establish a $\boldsymbol{sh_k}$ between communicating parties, providing an additional layer of security.

The encryption process utilizes the Advanced Encryption Standard (AES) in Cipher Feedback (CFB) mode, ensuring confidentiality and integrity of the transmitted data.

Compared to other hybrid algorithms, our method stands out in terms of efficiency and security. The reduced key size and faster computation, thanks to ECC, contribute to a more agile and resource-efficient cryptographic system. The integration of Diffie-Hellman enhances the security of key exchange, minimizing the risk of interception and unauthorized access.

REFERENCES

[1] Nguyen Minh Trung and Nguyen Binh , The 2014 International Conference on Advanced Technologies for Communications (ATC'14)

[2] Shilpi Gupta and Jaya Sharma , 2012 IEEE International Conference on Computational Intelligence and Computing Research

[3] Tapan Kumar Hazra, Anisha Mahato, Arghyadeep Mandal and Ajoy Kumar Chakraborty , 978-1-5386-2215-5/17/$31.00 ©2017 IEEE

[4] E.VIDHYA, S.SIVABALAN and R.RATHIPRIYA , Proceedings of the Fourth International Conference on Communication and Electronics Systems (ICCES 2019) IEEE Conference Record # 45898; IEEE Xplore ISBN: 978-1-7281-1261-9

[5] Ermatita , Yugo Bayu Prastyo and I Wayan Widi Pradnyana, 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)

[6] Junnel E Avaestro , Ariel M. Sison and Ruji P. Medina , "IEEE 4th International Conference on Technology, Informatics, Management, Engineering & Environment" (TIME-E) Bali, Indonesia, November 13-15, 2019

[7] Muneer Bani Yassein , Shadi Aljawarneh and Ethar Qawasmeh ,"The International Conference on Engineering & Technology" ICET2017, Antalya, Turkey

[8] Nissa Mehibel and M'hamed Hamadouche , "The 5th International Conference on Electrical Engineering – Boumerdes "(ICEE-B) October 29-31, 2017, Boumerdes, Algeria.

[9] Swapnil Chaudhari, "INTERNATIONAL JOURNAL FOR RESEARCH & DEVELOPMENT IN TECHNOLOGY", (2018) Volume-9,Issue-5(May-18) ISSN (O) :- 2349-3585

[10] Sadiq Aliyu Ahmad and Sadiq Aliyu Ahmad , 2019 15th International Conference on Electronics, Computer and Computation (ICECCO)

[11] Ugbedeojo Musa , Marion O. Adebiyi, Oyeranmi Adigun, Ayodele A. Adebiyi and Charity O. Aremu, 2023 International Conference on Science, Engineering and Business for Sustainable Development Goals (SEB-SDG)

[12] Hema Srivarshini Chilakala , N Preeti and Murali K , 2022 IEEE North Karnataka Subsection Flagship International Conference (NKCon)

[13] Harshit Sharma, Rakesh Kumar and Meenu Gupta , 2023 2nd International Conference for Innovation in Technology (INOCON) ,(3-5 March)

[14] G. Swain, Object-Oriented Analysis and Design Through Unified Modelling Language. Laxmi Publication, Ltd., 2010.

[15] G. Booch, D. L. Bryan, and C. G. Peterson, Software engineering with Ada. Redwood city, Calif.: Benjamin/Cummings, 1994.

[16] J. M. Zelle, Python Programming: an introduction to computer science. Portland, Oregon: Franklin, Beedle & Associates Inc, 2017.