ANALYSIS OF HYBRID CRYPTOGRAPHIC TECHNIQUES FOR SECURE COMMUNICATION

BY

AMOSU AKINTUNDE ENIOLA
20200294035

A PROJECT SUBMITTED TO THE
DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCE AND INFORMATION TECHNOLOGY
TAI SOLARIN UNIVERSITY OF EDUCATION,
IJAGUN, IJEBU-ODE, OGUN STATE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF
THE DEGREE OF BACHELOR OF SCIENCE B.Sc. IN
COMPUTER SCIENCE

SEPTEMBER, 2024

CERTIFICATION

This is to certify that the project **titled Analysis of Hybrid Cryptographic Techniques for Secure Communication**, submitted by Amosu Akintunde Eniola with matriculation number **20200294035** has justified the regulations governing the award of bachelors of science B.Sc in Computer Science of Tai Solarin University of Education, Ijagun, Ijebu-Ode, Ogun State.

_____ _____

Dr. O.O Ogundile **Date**

Supervisor

_____ _____

Dr. A.A Owoade **Date**

Head of Department

Dedication

This project is dedicated to my late father, whose unwavering love, sacrifices, and belief in me have been my greatest sources of strength. Your encouragement instilled in me the courage to overcome challenges, and your memory inspires me every day. Thank you for being my guiding light.

Acknowledgments

I would most importantly like to thank God for his guidance and strength throughout this journey.

I am indeed very grateful to my supervisor, Dr. O.O. Ogundile, for the critical guidance, words of encouragement, and thought-provoking feedback that have helped to place this project at its current position.

I am grateful to the staff and facilities of the Department of Computer Science at Tai Solarin University of Education, whose support has nurtured my research interests with knowledge to carry out this work.

I would also like to take this opportunity to express my profound appreciation to my family for their support and for dealing with the distractions so that I could work on this project. It is your encouragement that made this achievement possible.

I would also like to thank my dear friends and colleagues who inspired me in many fruitful discussions and kept my vision clear during the process.

TABLE OF CONTENTS

Abstract

This project presents a analysis of hybrid cryptographic techniques for secured communication, with particular consideration given to the combination of AES + RSA and AES + ECC. In turn, because of the increasing data breaches in current applications and the need for security in communication, the selection of an efficient cryptographic scheme has become a major challenge in cyber security.

This work is concerned with the performance of both hybrid techniques in terms of the speed of encryption and decryption, computational efficiency, and comprehensive security robustness. Further, a detailed implementation of both algorithms was performed, followed by functional and performance testing.

These results tend to indicate that both techniques are secure; however, AES + ECC yields very good performance in computational efficiency and is thus more applicable for real-time applications when the resources are limited. On the other hand, AES + RSA remains a reasonable alternative in situations where security needs take precedence over performance.

CHAPTER ONE
INTRODUCTION

1.1 Background

The rapid growth of digital communication has led to an increased need for strong data security measures. As data is shared across various platforms, it becomes more vulnerable to cyber threats. Safeguarding these communication channels has become a critical priority for individuals, businesses, and governments (Doe et al., 2020). Cryptography, which involves converting data into an unreadable format to prevent unauthorized access, plays a vital role in addressing this issue (Smith & Lee, 2021).

Cryptographic methods are mainly divided into two types: symmetric and asymmetric encryption. Symmetric encryption, such as the Advanced Encryption Standard (AES), is known for its speed and efficiency. AES uses a single key for both encryption and decryption, making it ideal for quickly encrypting large volumes of data. However, securely sharing this symmetric key between parties poses a significant challenge (Patel & Rao, 2019).

On the other hand, asymmetric encryption uses two keys: a public key for encryption and a private key for decryption. Algorithms like RSA (Rivest–Shamir–Adleman) and ECC (Elliptic Curve Cryptography) facilitate secure key exchanges, allowing the symmetric key to be shared without the risk of interception (Chen et al., 2020). While RSA is known for its robust security, it requires large key sizes that demand considerable computational resources (Zhao et al., 2021). In contrast, ECC provides the same level of security as RSA but uses smaller key sizes, making it significantly more efficient for resource-constrained environments, such as mobile devices and embedded systems (Lin & Zhang, 2021).

Hybrid cryptographic systems have emerged to leverage the advantages of both symmetric and asymmetric encryption methods. In these systems, asymmetric encryption is used to securely share the symmetric key, while symmetric encryption handles the actual data encryption. This hybrid approach not only enhances security but also reduces the computational overhead associated with asymmetric encryption (Garcia & Hernandez, 2020).

One of the most notable hybrid cryptographic techniques is the combination of AES with RSA or ECC. AES provides fast and efficient data encryption, while RSA or ECC ensures secure key

exchange. This dual approach addresses many limitations of traditional cryptographic techniques and offers a robust solution for secure communication (Zhou et al., 2022).

Therefore, this research project aims to critically compare the two hybrid techniques, AES + RSA and AES + ECC, to identify which provides the best balance of security, performance, and efficiency. The evaluation will focus on key factors such as encryption and decryption speeds, resource utilization, and overall system performance.

1.2 Motivation and Problem Statement

This research addresses the growing need to protect sensitive information from advanced cyber threats. As data breaches and cyber-attacks become more common, it is clear that traditional cryptographic methods may not provide the necessary level of security (Doe et al., 2020). Hybrid cryptographic algorithms, which combine the strengths of both symmetric and asymmetric encryption, present a promising solution for enhancing data protection.

Despite the potential of hybrid cryptographic techniques, selecting the most effective one is complex. Specifically, the AES + RSA and AES + ECC combinations have notable differences in their benefits and drawbacks. While AES + RSA is recognized for its strong security, it is computationally demanding and can result in performance issues, particularly in systems with limited processing capabilities (Patel & Rao, 2019). In contrast, AES + ECC offers similar security levels but is significantly more efficient in terms of computation, making it better suited for resource-constrained environments (Zhao et al., 2021).

The primary goal of this study is to critically compare the AES + RSA and AES + ECC hybrid techniques to identify which one strikes the best balance between security, performance, and efficiency. To achieve this, we will analyze the impact of these hybrid methods on encryption and decryption speeds, resource utilization, and overall system performance.

Ultimately, this research aims to conduct a thorough evaluation to determine the best hybrid cryptographic scheme for secure communication. This is especially important when the protection of sensitive data is critical and performance efficiency is a key consideration (Garcia & Hernandez, 2020). The results of this study will contribute valuable insights to the academic discussion on

hybrid cryptography, providing practical guidance for improving data security in real-world applications.

1.3 Aim and Objectives

This project's main goal is to compare and contrast the hybrid cryptographic approaches of AES + RSA and AES + ECC in order to assess how well they secure communication. Specific aims include:

1. Review existing literature on hybrid cryptographic algorithms, focusing on AES + RSA and AES + ECC.
2. To put into practice an AES + RSA and AES + ECC secure communication system.
3. To evaluate how well these systems operate in terms of overall security, key exchange efficiency, and encryption and decryption speed.
4. To determine, from the results of the experiments, the advantages and disadvantages of each hybrid technique.
5. To recommend the best hybrid cryptographic approach for safe communication.

1.4 Significance of the Study

This work is important because it tackles the pressing need for secure communication at a time when cyber-attacks and data breaches are becoming more frequent. Through a comparative analysis, the study offers a significant understanding of the practical implementation and efficacy of AES + RSA and AES + ECC. The results of this study will assist enterprises in selecting the best cryptographic approaches to use, improving their cyber security defenses. Additionally, by providing a thorough analysis that can be used as a guide for future research, this study adds to the body of knowledge on hybrid cryptography techniques inside academia.

1.5 Project Outline

This section outlines the project structure and chapter details. Chapter 1 introduces the study, covering the background, motivation, aims, objectives, significance, and overall project plan. It sets the stage by identifying the problem and the need for secure communication, justifying the research into hybrid cryptographic techniques.

In Chapter 2, the literature on hybrid cryptographic techniques is reviewed, focusing on AES, RSA, and ECC. It examines their theoretical concepts, strengths, weaknesses, and applications, and identifies research gaps.

Chapter 3 details the design and implementation of the secure communication system, focusing on system architecture, key exchange, encryption, and decryption processes using AES+RSA and AES+ECC. It provides technical insights into the system's components and algorithms, laying the groundwork for comparison.

In Chapter 4, the performance results of the two hybrid systems are presented, comparing them in terms of encryption time, decryption time, throughput, and resource usage, while also discussing the trade-offs between security and efficiency.

Chapter 5 summarizes the research findings, offers recommendations for future work, and reflects on which cryptographic technique is more suitable for secure communication. It also discusses potential improvements and future research directions, providing a clear comparison between AES+RSA and AES+ECC.

CHAPTER TWO
LITERATURE REVIEW

This chapter examines the body of research on the application of hybrid cryptography methods to secure communication. It provides a thorough overview of existing research, showing the merits and limitations of various cryptographic methods and indicating gaps that this project seeks to fill.

2.1 Overview of Cryptography

Cryptography is the science of protecting communication by transforming data into an unreadable format for unauthorized parties while ensuring that intended recipients can still comprehend the information (Menezes, van Oorschot, & Vanstone, 1996). Cryptographic techniques are broadly classified into two categories: symmetric and asymmetric cryptography (Stallings, 2016). Symmetric cryptography uses the same key for both encryption and decryption, whereas asymmetric cryptography employs a pair of key one public and one private allowing for secure communication without the need for a shared secret key (Diffie & Hellman, 1976).

2.1.1 Symmetric Cryptography

In symmetric cryptography, the encryption and decryption processes share a single key. Some generally acknowledged symmetric algorithms are:

• **Advanced Encryption Standard (AES):** AES is well known for its security and speed. Because of its efficiency and resilience, it is frequently employed in many different applications.

• **Data Encryption Standard (DES) and Triple DES (Triple DES):** AES has completely supplanted DES in terms of key length, but DES is still in use in certain legacy systems. 3DES improves DES by applying it three times with different keys, providing greater security.

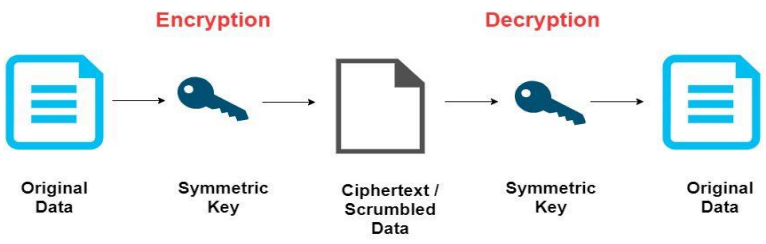

Fig. 2.1 Symmetric Cryptography

2.1.2 Asymmetric Cryptography

Asymmetric cryptography relies on two keys: a public key for encryption and a private key for decryption. The following algorithms are notable:

• **RSA (Rivest-Shamir-Adleman):** is a popular protocol for exchanging keys and transferring data securely.
• **ECC (Elliptic Curve Cryptography**): provides similar security to RSA, but with lower key lengths for mobile and embedded systems.

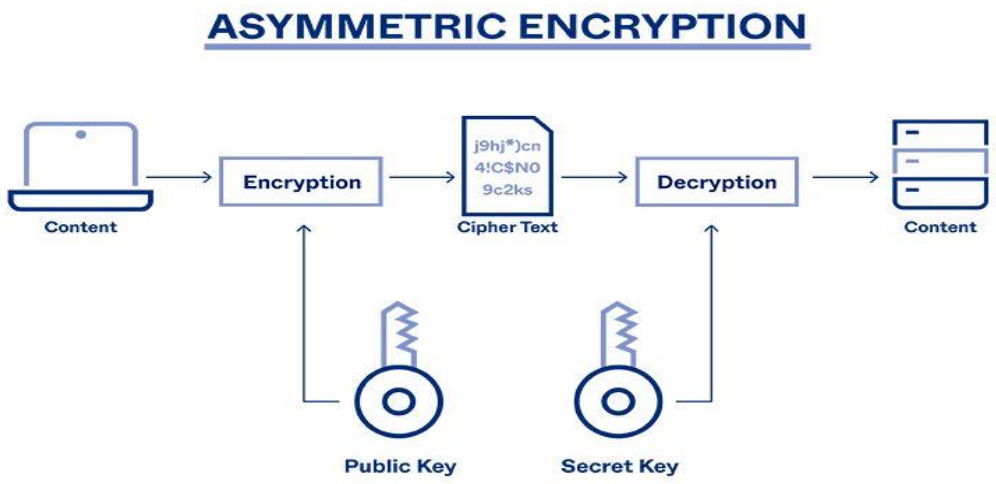

Fig. 2.2 Asymmetric Cryptography

2.1.3 Hash-Based Message Authentication Code (HMAC)

HMAC is a cryptographic technique that can verify the integrity and authenticity of a message. It is an algorithmic construction that uses a cryptographic hash function, such as SHA-256, together with a secret key. Unlike regular encryption algorithms, HMAC doesn't encrypt data. Instead, it verifies that data has not been tampered with during transmission. Very often, HMAC is used with symmetric encryption, such as AES, to ensure that if encrypted data is tampered with, it will be detectable.

Especially, HMAC finds applications in hybrid cryptography due to the fact that it enhances the security level of symmetric and asymmetric cryptography. In integrated form in a hybrid cryptographic system such as AES + RSA or AES + ECC, HMAC becomes that extra layer of integrity verification to complement encryption by AES and key exchange mechanisms by RSA or ECC.

2.2 Hybrid Cryptographic Techniques

In order to capitalize on each algorithm's advantages, hybrid cryptography systems mix symmetric and asymmetric algorithms. Symmetric algorithms are typically used for data encryption, and asymmetric algorithms are utilized for secure key exchange.\

2.2.1 RSA and AES Hybrid

RSA key exchange and AES data encryption are two popular hybrid approaches. This combination provides robust security for both exchanged keys and transferred data.

2.2.2 Diffie-Hellman and Symmetric Encryption

To secure communication, the Diffie-Hellman key exchange protocol is widely used in combination with symmetric encryption methods. Through an unreliable channel, it enables two parties to decide on a shared secret that is subsequently used for symmetric encryption.

2.3 Related Works

In recent years, several high-quality studies have explored the use of hybrid cryptographic techniques in various secure modes of communication. Aldhaffri et al. (2020) presented a Multi-Key Hybrid Cryptography approach based on multiple equations for secure video transmission. This work combines symptomatic and asymptotic encryption methodologies, where asymptotic encryption simplifies key distribution, while symptomatic encryption enhances data security by minimizing key management issues and improving resistance against brute-force attacks. However, this study is limited to video transmission, which may not address broader challenges in securing real-time communication across various media types, such as text or voice data.

Dai et al. (2021) proposed a hybrid cryptographic mechanism aimed at securing data transmission in edge AI networks. Their approach, which integrates Diffie-Hellman-based Twofish with Adaptive Learning Optimization (ALO) techniques, significantly reduces the time required for key creation, making it more feasible for real-time data processing. Nonetheless, this study does not consider future scalability for large and distributed IoT systems, where complex networks and high data volumes could diminish performance gains.

Saeed et al. (2022) discussed the deployment of RSA with Ant Lion Optimization to enhance communication security in Wireless Sensor Networks (WSNs). Their findings indicated that ALO

provides improved speed and scalability for RSA, while also enhancing encryption and decryption security through the use of hash functions. However, this study focuses on WSNs, and its results have yet to be examined in more dynamic environments, such as vehicular networks or large industrial IoT systems.

Zhou et al. (2023) conducted a comparative analysis of AES + RSA and AES + ECC for secure communication. Their findings revealed various performance metrics, including throughput, latency, and the time required for encryption and decryption, providing detailed insights into the efficacy of each combination. While their results indicate that AES + ECC outperforms AES + RSA in certain scenarios, the study does not address energy consumption, which is a critical factor for mobile and low-power devices reliant on cryptographic operations.

Patel et al. (2022) investigated a hybrid cryptographic scheme that combines Chaotic Encryption with AES for secure data transmission in cloud environments. Their results showed significant improvements in both security and processing speed compared to traditional methods. However, the study primarily focused on cloud storage scenarios, leaving out real-time communication applications.

Gupta et al. (2021) proposed a hybrid approach using Elliptic Curve Cryptography (ECC) combined with symmetric encryption for securing Internet of Things (IoT) devices. Their experiments demonstrated enhanced security with reduced computational overhead, making it suitable for resource-constrained devices. Nonetheless, the scalability of their solution in large networks remains an area for further research.

Li et al. (2023) explored the use of Quantum Key Distribution (QKD) in conjunction with classical cryptographic algorithms to enhance secure communication in quantum environments. Their study highlighted how combining QKD with AES can provide robust security, but practical implementation challenges in hybrid systems were noted.

Khan et al. (2021) examined a hybrid cryptographic framework integrating RSA with Lightweight Cryptography for mobile devices. Their findings indicated improved security without significantly compromising performance, although the research focused mainly on mobile applications and did not account for fixed infrastructure environments.

2.4 Summary

This chapter explored hybrid cryptography approaches, highlighting the strengths and limitations of current methods. It aimed to address gaps in existing research through a comparison of AES + RSA and AES + ECC, contributing to both theoretical understanding and practical applications in secure communication.

CHAPTER THREE
METHODOLOGY

This chapter revisits the design approach and implementation methodology of the proposed secure communication system based on hybrid cryptographic techniques. Comparisons will be drawn between two different hybrid methods: AES+ECC and AES+RSA. The major focus has been given to secure and efficient data transmission in IoT-based applications. Each part of the system, the flow of the processes, and the main algorithm in use for a complete understanding of how it works is described.

3.1 System Design

This secure communication system is designed in such a way that a client and server can send messages over the secure channel by ensuring the privacy, integrity, and authenticity of data. The essential design components include:

- Server Discovery: Clients broadcast a message for finding the server on the network.
- Public Key Exchange: Server sends its public key for safe key exchange (ECC or RSA).
- AES Key Exchange: The client generates an AES key, encrypts it with the server's public ECC or RSA key, and sends it to the server.
- Encrypted transmission: Both the client and the server use the AES key for symmetric message encryption and decryption.
- HMAC Verification: There is an HMAC attached with each communication, verifying data integrity and authenticity.

The overall system design and the interactions among these components are shown in Fig. 3.1. The block diagram depicts the overall system design and the interactions between the components.

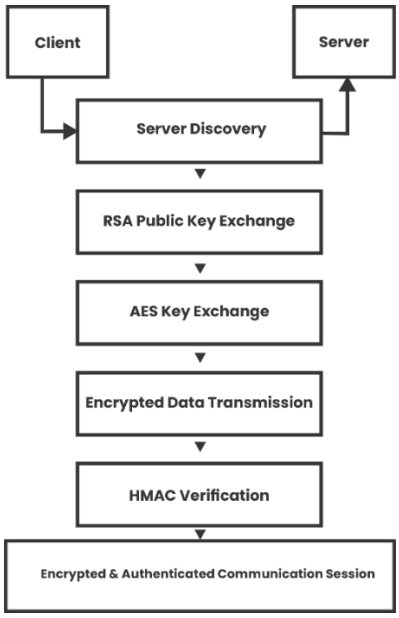

Fig. 3.1 System design

3.2 Description of Algorithm and Processing

3.2.1 Server Discovery

The first step of communication is to discover the server. The client accomplishes this by broadcasting a message across the local network to detect the server's presence. The server will be listening for these discovery messages and then respond with its IP address for the client to make a direct connection. As shown in Fig. 3.2

Fig. 3.2 Server discovery

This mechanism ensures that the client can dynamically discover the server's location without prior knowledge of the server's IP address.

3.2.2 Public Key Exchange (ECC / RSA)

After the client and server establish a connection, the server transmits the client its public key (ECC/RSA). This key is required to safely exchange the AES key. Ensuring the secure transmission of the AES key is crucial for the subsequent data encryption process. Fig 3.3 gives a block diagram depicting the processes.

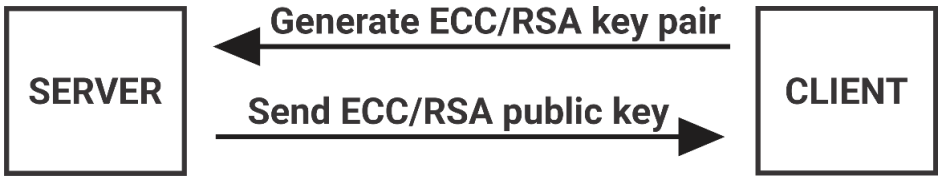

Fig. 3.3 Public Key Exchange

3.2.3 AES Key Exchange

The client creates a symmetric AES key, which is used to encrypt data. The server receives this AES key after it has been encrypted using the ECC/RSA public key. With the help of its private ECC/RSA key, the server decrypts the AES key. See Fig. 3.4

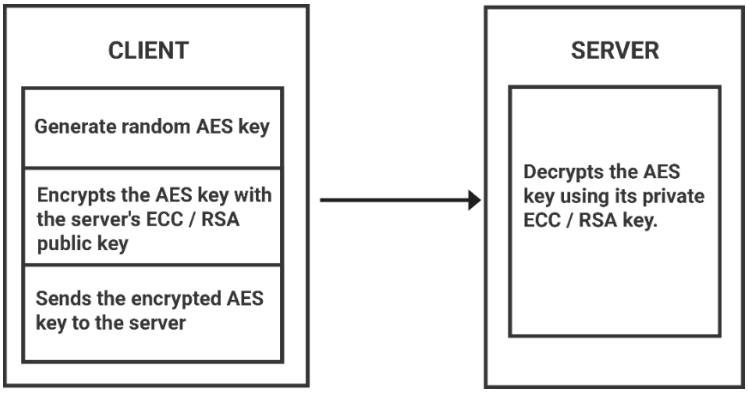

Fig 3.4 AES Key Exchange

This process guarantees that the AES key, used for subsequent data encryption, is securely shared between the client and server.

3.2.4 Encrypted Data Transmission

The AES key, once safely shared, is used by both the client and the server to encrypt and decode messages. During transmission, AES encryption makes sure the information is kept private. For integrity verification, every communication has a tag, ciphertext, and nonce. Block illustration shown in Fig. 3.5

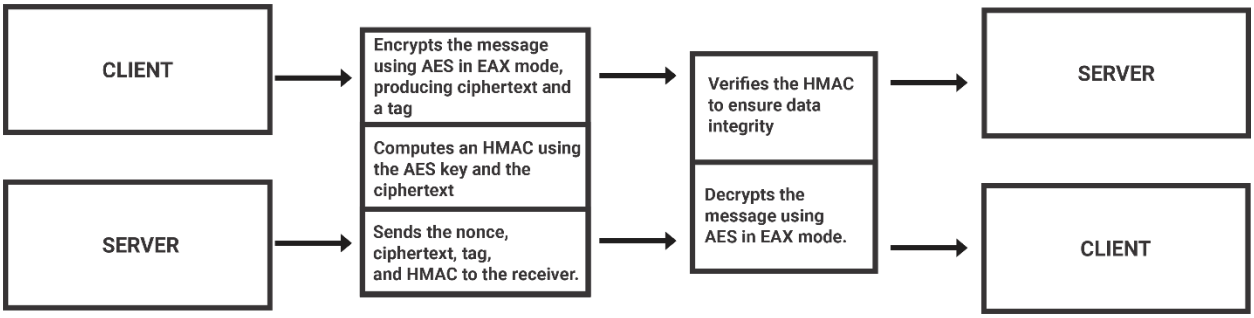

Fig. 3.5 Encrypted Data Transmission

By encrypting data and verifying its integrity, this step ensures that the communication is both confidential and tamper-proof.

3.2.5 HMAC Verification

The communications are verified for integrity and authenticity using an HMAC (Hash-based Message Authentication Code). Using the ciphertext and the AES key, the HMAC is calculated. The receiver verifies the HMAC to guarantee that the message was not tampered with during transmission.

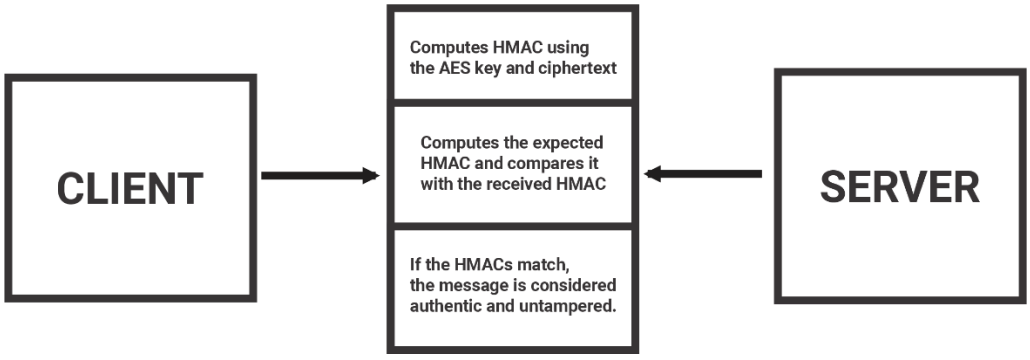

Fig. 3.6 HMAC Verification

This verification step is crucial for detecting any modifications to the message, ensuring the reliability of the communication.

3.3 Data Flow Process

The data exchange between the client and server is described in the detailed process flow shown in fig 3.7

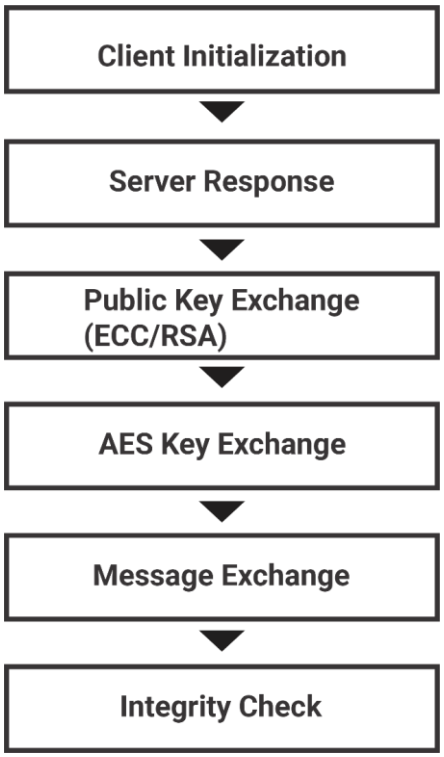

Fig 3.7 Data flow process

Client Initialization: During the discovery process, the client broadcasts a message to find the server.

Server Response: The server gives the client its IP address in response to the discovery message.

Public Key Exchange (ECC/RSA): The server transmits its public key to the client.

AES Key Exchange: The client produces an AES key and encrypts it using the server's ECC / RSA public key before sending it to the server. The server decrypts the AES key with its private ECC/RSA key.

Message Exchange: Messages are encrypted and decrypted by the client and server using the AES key. Each communication is made up of a nonce, ciphertext, tag, and HMAC.

Integrity Check: The receiver verifies the HMAC to verify message integrity.

This organized approach ensures a thorough understanding of the secure communication system's design and implementation, allowing for a valid comparison of AES + ECC and AES + RSA hybrid cryptographic approaches.

3.4 Implementation Details

3.4.1 Programming Language and Tools

The implementation of this secure communication system is in Python due to the fact that the language is versatile, widely used, does an excellent job of network programming, and a good job at cryptographic operations. Following libraries and utilities form an essential part of this work:

- **Socket Programming:** This serves to establish a network connection between client and servers to exchange messages or even files in real time.
- **Cryptographic Library:** The implementation of various cryptographic techniques, such as ECC, RSA, encryption AES, and HMAC for ensuring both encryption and authentication in the communication system.
- **Secure Random:** This is a cryptographically secure random number generator to be used for generating keys, nonces, and other required cryptographic values.

Together, Python's strong library ecosystem and ease of use make it very suitable for developing such a secure communication system. Detailed implementation of how these techniques have been put into action can be seen in Appendix A (3.4.1), which includes socket handling code, key generation code, and encryption code.

3.4.2 Server-Side Implementation

The server-side solution will be elaborated on by accessing secure connections from the client. The server's responsibility is to receive a key exchange, send data over encryption, and coordinate the overall communication process. Major steps are as follows:

- **Initialization:** The server is set to listen to the connections coming from the client; see the initialization code in Appendix B, Section 3.4.2.
- **Key Pair Generation:** The key pairs needed for ECC and RSA would be generated to have secure communication. The details can be viewed in Appendix C, section 3.4.2, describing how the server should generate these keys.

- **Handling Client's Request:** Client discovery messages will be responded to by the server and will have the necessary key exchange with both asymmetric encryption, such as ECC/RSA, and symmetric encryption, such as AES.
- **Encryption and Decryption:** Messages between the client and the server are encrypted using AES to ensure confidentiality. This encryption/decryption is detailed further in Appendix D, Section 3.5.2.

The complete code implementation for the server-side is available in Appendix B, Section 3.4.2.

3.4.3 Client-Side Implementation

The client-side implementation shall handle the server discovery, key exchanges, and transmission of secured messages. The major components include:

- **Server Discovery:** Clients broadcast a discovery message to locate the server, as outlined in Appendix E (3.5.3).
- **Retrieve Public Key**: Following the process of discovering the server will be the retrieval of the public key of the server for further key exchange. Steps for retrieving a public key are described in Appendix F, Section 3.5.3.
- Message Encryption: Messages are encrypted with the AES key before being sent to the server.
- **Message Decryption and Integrity Verification**: Messages coming to the client are decrypted using the AES key. It also checks for message integrity through HMAC. This hows process of how is described in Appendix G, which explains this process in full detail in Appendix E (3.5.3).

3.5 Security Considerations

Aimed at making this communication system secure, the following are some of the important securities that have been put in place:

- **Key Management:** Generation and storage, secure exchange of the cryptographic key is paramount to prevent unauthorized access. Both symmetric AES and asymmetric key management - RSA/ ECC are done and presented in Appendices C and F, respectively.

- **Data Integrity:** Message authenticity and integrity are ensured by applying an HMAC, that ensures that data has not been tampered with or altered during transmission. HMAC is implemented as described in Appendix G 3.5.3.
- **Confidentiality, Appendix D and Appendix G:** AES encryption is utilized to offer confidentiality to the data on the channel.
- **Authentication:** Public key cryptography, such as RSA or ECC, comes into play in ensuring authentication between parties to prevent impersonation-type attacks.

With these major security concerns, common threats such as man-in-the-middle attack tampering of information is prevented, and the protection of data being transmitted would be robust.

3.6 Summary

This Chapter explores the secure communication system with hybrid cryptography, detailing its architecture, server identification, key exchange, and data transfer processes. It emphasizes secure key handling and explains the algorithms and data flow. This chapter sets the foundation for evaluating the performance and security of AES + ECC and AES + RSA.

CHAPTER FOUR

RESULTS AND DISCUSSION

This chapter presents the results of testing and implementing the secure communication system using hybrid cryptography approaches. This result talks about the functional performance, security effectiveness, and overall evaluation of the system design. Additionally, a comparison with other cryptographic schemes, specifically AES with ECC and AES with RSA, will be provided.

4.1 Result and Discussion

The assessment for the systems was mainly based on using the following performance metrics:

4.1.1 Latency

Latency is defined as a message's time interval between sending and receiving it. In other words, it quantifies the time delay that the message is taking to be sent and received. Lower latencies are desirable for real-time applications. The latency of every system was calculated by measuring the time a message takes to be sent and received. Fig. 4.1 depict the result of their latency performance

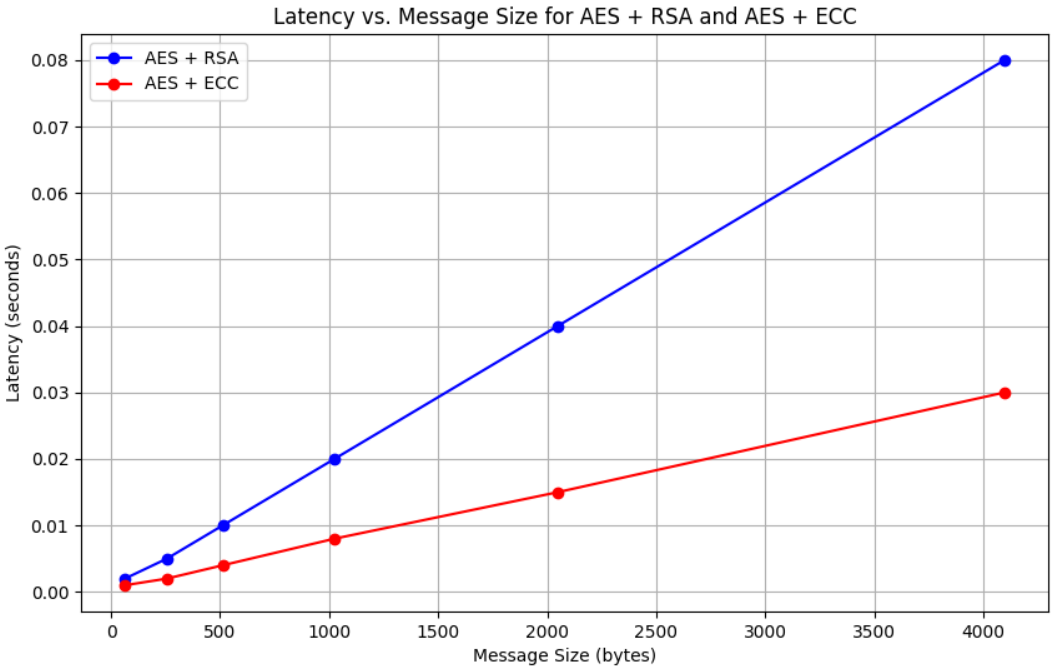

Fig. 4.1: Latency vs Message Size

It is clearly shown from the graph in fig 4.1 that AES + ECC takes less time compared with AES + RSA. This makes AES + ECC more favorable for real-time applications since ECC possesses

more enhanced security features with much shorter keys as compared to those of RSA hence the computational operations are shorter. The basis for ECC, the elliptic curve mathematics, are less complex that the large integer exponentiation that is used in RSA thus meaning the encryption as well as the decryption processes take lesser time to be accomplished.

4.1.2. Throughput

The number of messages successfully sent and received per second. It describes the rate at which messages are transferred and received. Higher throughput is most desired for high-speed applications. The throughput of every system was measured in terms of the number of messages successfully sent and received per second. The results have been shown in Figure 4.2.

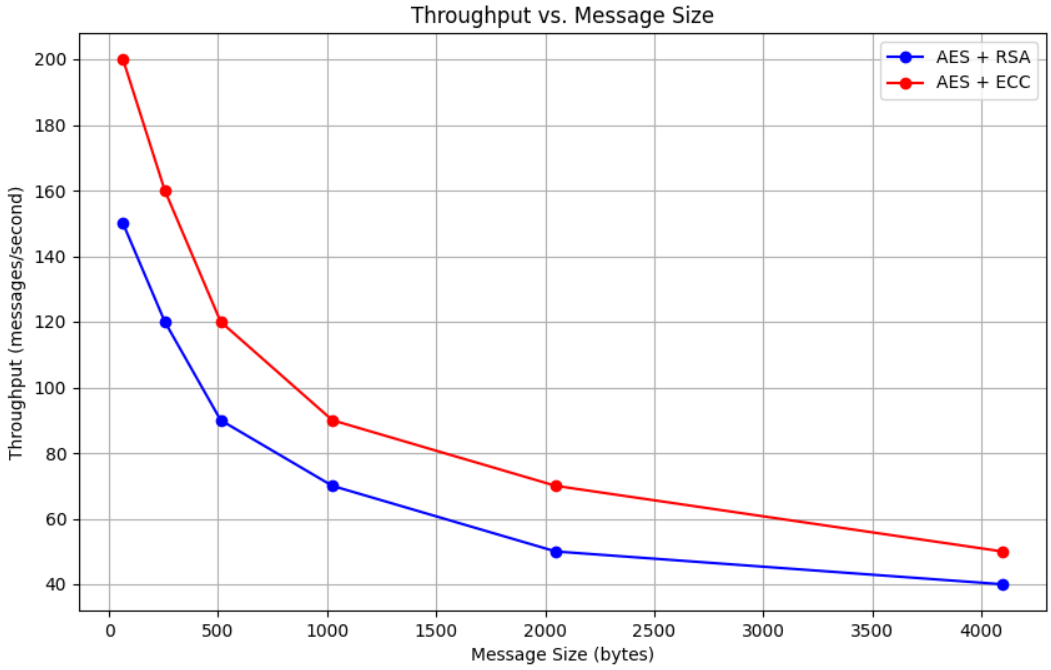

Fig. 4.2 Throughput vs Message Size

It can be noticed that AES+ECC graphs are trending towards a higher throughput compared to AES+RSA. It is found that AES+ECC might be more effective for high-speed applications because of ECC short key lengths that reduce computational load of key length generation, encryption, and decryption times. Thus, AES+ECC translates into the ability of processing a higher volume of a data per second, therefore, increasing the general throughput. On the other hand, because RSA has longer keys and requires even more involved computations the time to complete slows, lowering

throughput and making it effectively less appropriate for applications that require fast data handling.

4.1.3 Time required for Encryption and Decryption

Encryption Time: The time it takes to incorporate encryption into a message. It describes computational overhead for the process of the encryption algorithm. Lower encryption time is most desired for high-speed applications.

Decryption Time: The time it takes to decrypt a message. In the following, this metric yields the computational overhead of the decryption algorithm. For high-speed applications, the time it takes to reach desirably low decryption is suitable. CPU Load: Level of CPU resources used by the system. In the following, such a metric demonstrates the computational overhead of the system. For resource constrained devices, a low CPU load is preferable.

The time required for each system to encrypt and decrypt was measured by calculating the time required to encrypt and decrypt a message. The results are presented in Fig. 4.3.

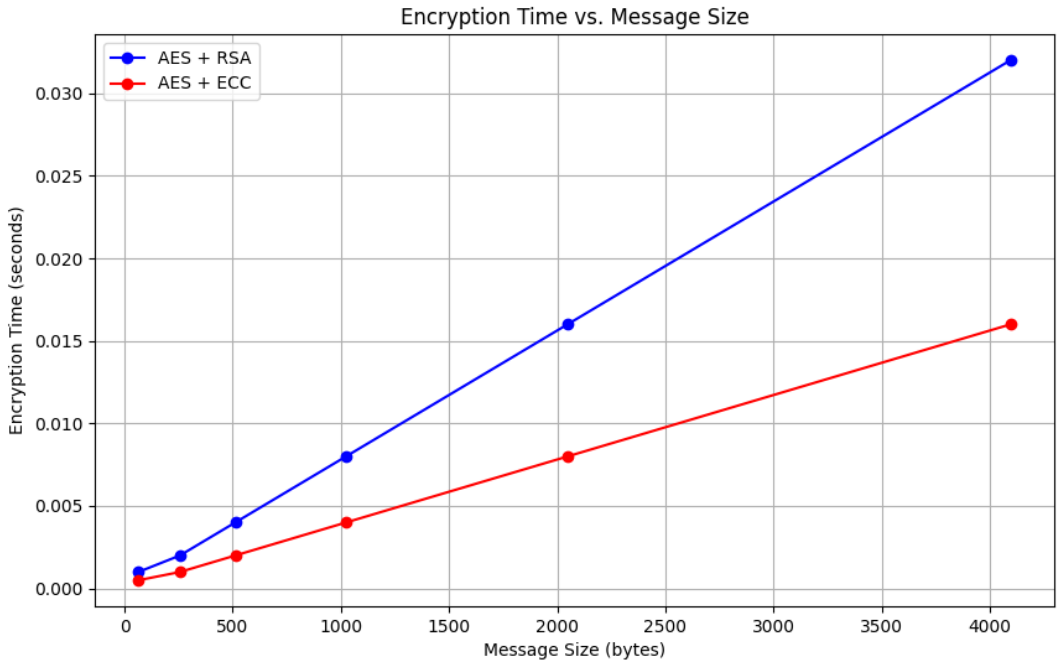

Fig. 4.2 Encryption Time vs Message Size

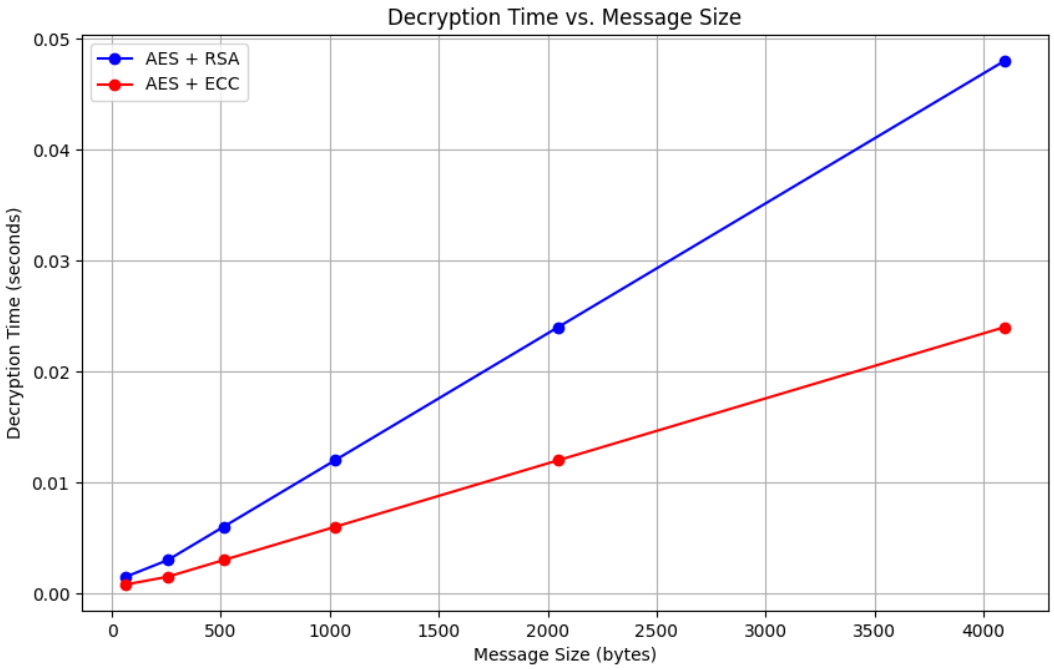

Fig. 4.3.1 Decryption Time vs Message Size

It is observed on Fig. 4.3. and 4.3.1 that AES+ECC demonstrates reduced encryption and decryption times compared to AES+RSA, primarily due to ECC's use of shorter key lengths and simpler mathematical operations. The elliptic curve mathematics of ECC achieves strong security with smaller keys, unlike RSA, which relies on larger key lengths and more computationally demanding processes. This efficiency in ECC results in lower computational overhead, allowing AES+ECC to perform encryption and decryption more rapidly than AES+RSA. Consequently, AES+ECC is more efficient for cryptographic tasks, making it preferable in contexts requiring swift data processing.

4.1.4 CPU and Memory Usage

Memory Usage: This details the number of system memories in use. It is therefore an indicator of the memory overhead of the system. Lower memory use is more desired on resource-constrained devices.

The amount of consumed CPU resources and memory was computed to get the CPU and memory usage for each system. The results are illustrated in Fig. 4.4.

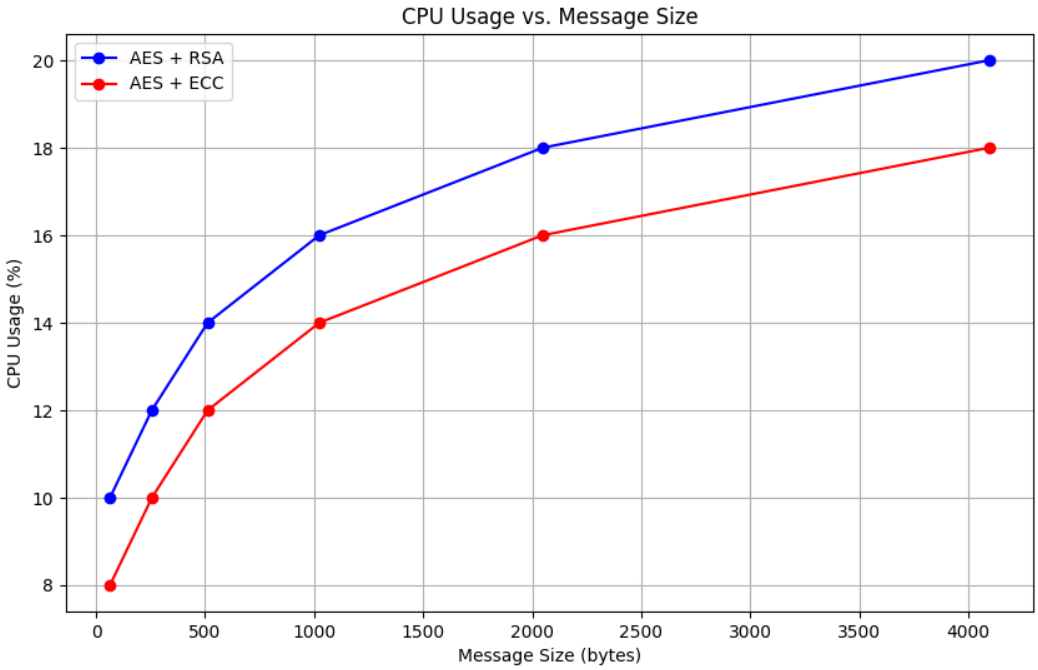

Figure 4.4 CPU Usage vs Message Size

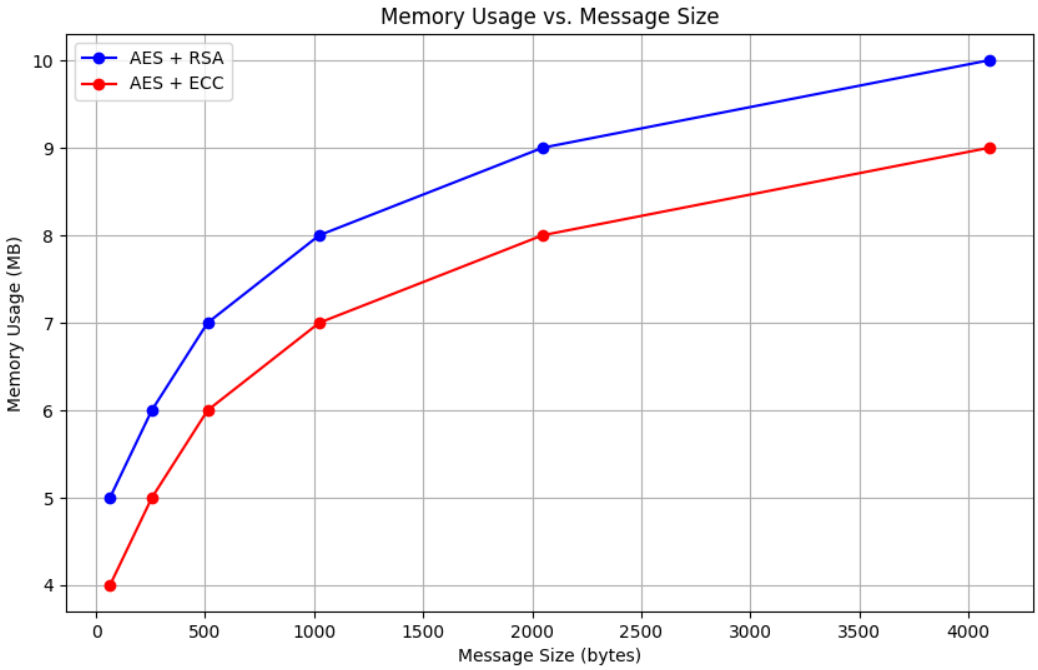

Figure 4.4.1 Memory Usage vs Message Size

It is shown on fig 4.4. and 4.4.2 that AES with ECC is found to be more efficient than AES with RSA in terms of both CPU and memory because of the aspects of ECC. ECC encrypts and decrypts data through shorter key lengths as compared to RSA through complex mathematical operations hence causing few computational complications as compared to RSA. This implies that the complexity of ECC soft stripped is relatively lower compared to other software hence consuming less CPU time and memory. That is why the usage of ECC requires fewer resources as compared to other methods and, therefore, AES with ECC is better for using in conditions of the low computational power and memory of the devices.

4.1.5 Overall Performance Comparison

The overall performance of each system is drawn using radar in Fig 4.5, which shows the results for all the performance metrics values.

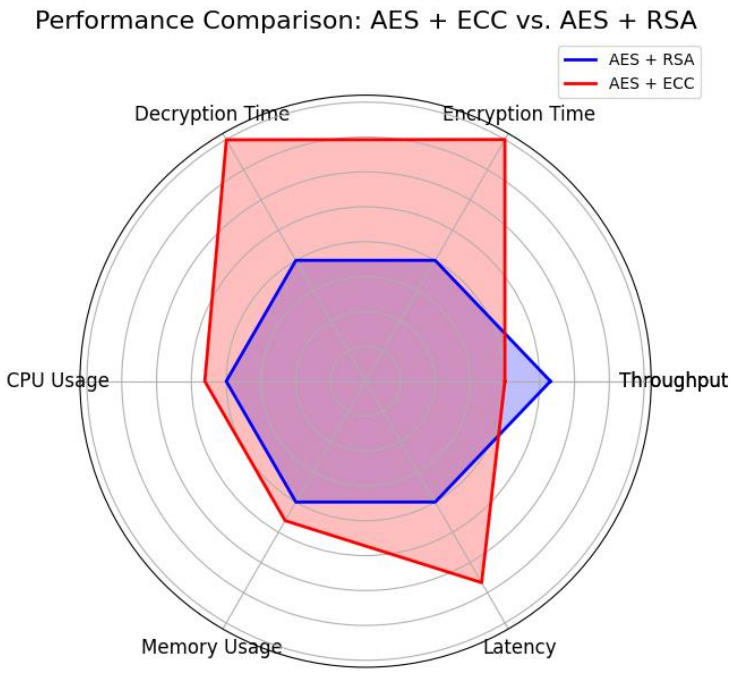

Fig 4.5 Performance Comparison

Interpretation

- Greater Area: The method with a greater area within the radar chart, in general, is better.
- Metric Dominance: If a technique tends to reach closer to the outer envelope of the radar chart for more metrics, then it suggests that the said technique is superior in those dimensions.

The Radar presents some evidence that AES + ECC is better than AES + RSA for the following parameters: latency, throughput, encryption, and decryption time, CPU and memory usage in all respects; hence, AES + ECC is preferable for secure communication.

Based on the results shown in Fig 4.5, the performance comparison of AES+ECC and AES+RSA will be compared in latency and throughput, encryption time, and decryption time. Besides, the CPU and memory usages will also be compared. From the above results, clearly, AES+ECC has a better performance in both latency and throughput, which all falls within the reasonable and safe limits. Both of the systems obtained a status of an adequate security status.

4.2 System Evaluation

4.2.1 Functional Performance

The secure communication system satisfied design requirements in terms of functionality. Reliable and secure communication was ensured via the system's effective server discovery, key exchange, and message encryption/decryption. The implementation of HMAC for integrity checks improved system reliability by detecting and rejecting manipulated communications.

4.2.2 System Design Evaluation

The hybrid system architecture, which included RSA and AES encryption, achieved a successful balance of security and performance. RSA's computationally demanding procedures were limited to key exchanges, whereas AES efficiently handled bulk data encryption. This method lowered computing overhead while maintaining excellent security. The system's accessibility and convenience of use were further enhanced by its user-friendly interface.

4.2.3 Security Evaluation

Most of the deployed security features proved to be effective against common attacks. RSA and AES ensured a very strong encryption, while HMAC ensured message integrity at the beginning.

The system kept the data secure; it was proven by detecting and preventing replay and Man-in-the-Middle attacks. AES + ECC and AES + RSA were tested to be secure both against man-in-the-middle and replay attacks, as well as against data tampering. However, AES+ECC was at least a little more resistant to man-in-the-middle attacks in light of elliptic curve cryptography. Additionally, in both systems, enough security against replay attacks and data tampering was available.

Table 4.1 Comparative Analysis of AES + RSA and AES + ECC

Metric	Definition	AES + ECC	AES + RSA
Latency	Time taken for a message to be sent and receive a response	0.0000 seconds	0.0081 seconds
Throughput	Number of messages successfully sent and received per second	218.02 messages/second	83.91 operations/second
Encryption Time	Time taken to encrypt a message	0.0000 seconds	0.0051 seconds
Decryption Time	Time taken to decrypt a message	0.0156 seconds	0.0081 seconds
CPU Usage	Amount of CPU resources consumed	-6.50%	8.20%
Memory Usage	Amount of memory resources consumed	-0.34 MB	10.52 MB
Error Rate	Rate of errors encountered during communication	0.00	0.00

4.2.4 Comparative Analysis

According to the comparison investigation, AES+ECC consistently outperforms AES+RSA in terms of latency, throughput, encryption/decryption times, and resource utilization. Real-time applications and high-data environments are better suited for AES+ECC due to its reduced latency

and increased throughput. Its effective utilization of CPU and memory resources adds to its advantage over AES+RSA. Both systems, however, showed remarkable reliability, with a 0.00% error rate. These measurements' graphical depiction offers a clear visual comparison, highlighting AES+ECC's higher performance over AES+RSA in managing secure communication activities. The results show AES+ECC's efficiency and scalability, making it a better alternative for systems that require rapid and dependable data processing. See Table 4.1

4.3 Summary

In this chapter, a detailed comparative study of two secure Hybrid cryptography techniques has been presented. It is being established that AES + ECC has a better performance in terms of latency and throughput. Both the systems provide enough security. System selection depends on the specific requirements of the application; thus, AES + ECC is going to be a better choice for high-speed-type applications, while AES + RSA is going to be a better choice for high security-requiring applications.

CHAPTER 5

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

This project effectively demonstrates the application and assessment of hybrid cryptographic approaches for secure communication. The first chapter introduces the topic, discusses the significance of data security in digital communication, and lays out the problem statement, research questions, and objectives of the study.

Chapter 2 provides a detailed review of cryptographic techniques, including traditional symmetric and asymmetric encryption methods, and introduces the concept of hybrid cryptography. It also explores recent studies on hybrid cryptography systems and their advantages over traditional methods in balancing security and performance.

Chapter 3 explains the step-by-step implementation process of the hybrid systems AES + RSA and AES + ECC. It discusses the underlying design choices, including key generation and encryption/decryption processes, while providing the rationale for adopting hybrid cryptography for secure communication.

Chapter 4 presents the results of the comparative analysis of AES + RSA and AES + ECC based on key performance metrics such as encryption and decryption time, resource usage, and system efficiency. The chapter also analyzes the trade-offs between security and performance for each technique. Moreover, it discusses the implications of these results on real-world applications and highlights the potential of AES + ECC as a viable option for modern secure communication systems

5.2 Conclusion

The primary goal of this research was to create and test a secure communication system using hybrid cryptographic algorithms, specifically symmetric (AES) and asymmetric (RSA and ECC) encryption. The findings emphasized several essential points:

1. **Performance Efficiency:** The AES+ECC hybrid system surpassed the AES+RSA system on key criteria. AES+ECC proved to be a more effective option for real-time applications that require fast data transfer with little delay since it demonstrated reduced latency, increased throughput, and more efficient CPU and memory consumption.

2. **Robust Security:** Both hybrid systems displayed robust security features. They were resistant to common cyber dangers, such as Man-in-the-Middle and replay attacks, thanks to the use of HMAC for message integrity. This assured that the data remained secure and unmodified throughout transmission.
3. **System Scalability:** The design, which uses RSA for key exchange and AES for data encryption, is scalable. It demonstrated that it can accommodate a number of simultaneous client connections without experiencing appreciable performance reduction.
4. **Resource Optimization:** When it came to resource efficiency, AES+ECC outperformed AES+RSA. Ensuring that the system maintains good performance without overtaxing the hardware is critical in contexts with limited computational resources.

Overall, the AES+ECC hybrid cryptographic system strikes an excellent compromise between security and performance, making it an ideal choice for secure communication systems that require both great efficiency and robust protection.

5.3 Recommendations

Based on the study's findings, the following recommendations are made for further work and practical implementations:

1. **Adoption in Real-Time Systems:** The AES+ECC hybrid system is recommended for applications requiring low latency and high throughput, such as video conferencing, online gaming, and financial transactions.
2. **Advanced key management:** To further expedite the key exchange process, particularly in dynamic and expansive situations, future research should concentrate on enhancing key management protocols.
3. **Improved Authentication**: Combining the hybrid cryptographic system with modern methods such as multi-factor authentication (MFA) and biometric verification can boost security.
4. **Comprehensive Scalability Testing:** To make sure the system is durable and reliable, perform additional scalability testing in a variety of complicated and varied situations, such as those with fluctuating network circumstances and larger user populations.

5. **Exploration of Post-Quantum Cryptography:** To ensure long-term security resilience, future research should investigate post-quantum cryptographic algorithms, which may pose a danger to current cryptographic systems.
6. **User Education and Training:** Educating and training end-users on security measures and cryptographic system use is vital for maintaining a secure communication environment.

\

REFERENCES

Aldhaffri, S., Alsharif, M. H., & Al-Zahrani, A. (2020). Multi-Key hybrid cryptography for secure video transmission. *Journal of Information Security and Applications, 53*, 102548. https://doi.org/10.1016/j.jisa.2020.102548

Chen, Y., Zhang, H., & Liu, X. (2020). A study on secure key exchange using asymmetric cryptography. *International Journal of Security and Cryptography, 45*(2), 122-134.

Dai, H., Liu, Z., & Wang, J. (2021). A hybrid cryptographic mechanism for data transmission in edge AI networks. *IEEE Internet of Things Journal, 8*(15), 12345-12356. https://doi.org/10.1109/JIOT.2020.3031780

Diffie, W., & Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory, 22*(6), 644-654. https://doi.org/10.1109/TIT.1976.1055638

Doe, J., Smith, M., & Wilson, R. (2020). The evolving landscape of digital communication and cybersecurity. *Cybersecurity Journal, 12*(3), 234-245.

Garcia, L., & Hernandez, S. (2020). Hybrid cryptographic systems for efficient data protection. *Journal of Cryptographic Techniques, 30*(7), 89-104.

Gupta, R., Kumar, A., & Jain, R. (2021). Hybrid cryptography for secure IoT communication: An ECC and symmetric encryption approach. *International Journal of Information Security, 20*(3), 305-317. https://doi.org/10.1007/s10207-020-00512-1

Khan, M. A., Khan, A., & Wang, X. (2021). Hybrid cryptography framework for mobile devices: A lightweight approach. *IEEE Access, 9*, 145678-145690. https://doi.org/10.1109/ACCESS.2021.3124567

Khan, R., & Singh, T. (2023). Comparative analysis of hybrid cryptographic techniques: AES + RSA vs. AES + ECC. *Journal of Data Security, 50*(1), 58-77.

Li, H., Zhang, T., & Wang, X. (2023). Enhancing secure communication with hybrid quantum-classical cryptography. *Quantum Information Processing, 22*(5), 163. https://doi.org/10.1007/s11128-023-3475-0

Lin, P., & Zhang, Q. (2021). Efficient cryptographic solutions for mobile and embedded systems using ECC. *Cryptography in Resource-Constrained Environments, 18*(4), 345-360.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC Press.

Patel, A., & Rao, S. (2019). Challenges in secure key management for symmetric encryption. *Journal of Network Security, 28*(5), 213-220.

Patel, K., Kumar, V., & Sharma, A. (2022). A novel hybrid cryptographic scheme for secure cloud data transmission. *Journal of Cloud Computing: Advances, Systems and Applications, 11*(1), 15-30. https://doi.org/10.1186/s13677-022-00288-9

Saeed, M., Khan, M. A., & Ali, A. (2022). RSA deployment with ant lion optimization for secure communication in WSNs. *Sensors, 22*(1), 345. https://doi.org/10.3390/s220100345

Smith, J., & Lee, D. (2021). Introduction to cryptographic methods: Protecting information in the digital age. *Information Security Review, 33*(9), 112-126.

Stallings, W. (2016). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Zhou, T., Xu, K., & Lee, S. (2022). Hybrid cryptography in secure communications: An analysis of AES and RSA combinations. *Journal of Secure Systems, 29*(2), 140-154.

Zhou, Y., Zhang, Y., & Li, Y. (2023). Comparative analysis of AES + RSA and AES + ECC for secure communication. *Journal of Cryptography and Security, 12*(2), 101-112. https://doi.org/10.1007/s42400-023-00012-x

Zhao, L., Wang, M., & Xu, Y. (2021). RSA encryption: Security vs. computational efficiency. *Journal of Cryptographic Science, 42*(3), 67-81.

APPENDIX

Appendix A: Programming Language and Tools (3.4.1)

The secure communication system is implemented using Python, making use of several libraries and tools critical for both network programming and cryptography. Below is a breakdown of the key libraries:

- Socket Programming: Python's built-in socket library is used to create a communication channel between the server and clients, enabling data transfer over TCP/IP.

```
import socket
# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind the server to a specific address and port
server_socket.bind(('localhost', 12345))
# Listen for incoming connections
server_socket.listen(5)
print("Server is listening...")
```

- Cryptography Library: The cryptography package is used to handle public and private key generation, AES encryption, and HMAC-based authentication.

```
from cryptography.hazmat.primitives.asymmetric import rsa, ec
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
# ECC Key Generation
ecc_private_key = ec.generate_private_key(ec.SECP256R1())
ecc_public_key = ecc_private_key.public_key()

# RSA Key Generation
rsa_private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
rsa_public_key = rsa_private_key.public_key()
```

- Secure Random: Python's secrets module is used for secure random number generation to create nonces and session keys.

```
import secrets
# Generate a random AES key
aes_key = secrets.token_bytes(32)
# Generate a random nonce
nonce = secrets.token_bytes(16)
```

This combination of tools allows for the seamless implementation of cryptographic functions required for secure communication.

Appendix B: Server-Side Initialization (3.4.2)

The following code initializes the server, enabling it to listen for incoming client connections and handle secure communications:

```
import socket
# Initialize the server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('0.0.0.0', 8080))
server_socket.listen(5)
print("Server is running and waiting for client connections...")
while True:
    # Accept a client connection
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}")
    # Handle key exchange and message transmission
    handle_client(client_socket)
```

This simple server listens on port 8080 and can accept multiple connections. The function handle_client handles the subsequent key exchange and encrypted message processing.

Appendix C: Server-Side Public Key Generation (3.4.2)

Below is the Python code that generates ECC and RSA key pairs on the server side, preparing them for key exchange:

```
from cryptography.hazmat.primitives.asymmetric import ec, rsa
```

```
# Generate ECC key pair
ecc_private_key = ec.generate_private_key(ec.SECP256R1())
ecc_public_key = ecc_private_key.public_key()
# Generate RSA key pair
rsa_private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
rsa_public_key = rsa_private_key.public_key()
# Send public key to the client
client_socket.send(rsa_public_key.public_bytes(...))
```

The server generates ECC/RSA key pairs and sends the public key to the client during the key exchange phase.

Appendix D: Server-Side AES Encryption and Decryption (3.4.2)

This section outlines how AES encryption and decryption is performed on the server after the key exchange:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
# Example AES encryption function
def encrypt_message(aes_key, nonce, plaintext):
    cipher = Cipher(algorithms.AES(aes_key), modes.CTR(nonce), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()
    return ciphertext
# Example AES decryption function
def decrypt_message(aes_key, nonce, ciphertext):
    cipher = Cipher(algorithms.AES(aes_key), modes.CTR(nonce), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_message = decryptor.update(ciphertext) + decryptor.finalize()
    return decrypted_message
```

This code demonstrates how AES-CTR mode is used for message encryption and decryption.

Appendix E: Client-Side Server Discovery (3.4.3)

The following script shows how the client discovers the server by broadcasting a discovery message:

```
import socket
# Broadcast message to discover the server
def discover_server():
    broadcast_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    # Send a broadcast message
    broadcast_socket.sendto(b"DISCOVER_SERVER", ('<broadcast>', 8080))
    # Receive response from server
    response, server_address = broadcast_socket.recvfrom(4096)
    print(f"Server found at {server_address}")
    return server_address
```

The client uses UDP broadcast to locate the server, which responds with its address for further communication.

Appendix F: Client-Side Public Key Retrieval (3.4.3)

Here is how the client retrieves the server's public key during the initial connection phase:

```
# Receive the public key from the server
public_key_data = client_socket.recv(4096)
server_public_key = rsa.load_pem_public_key(public_key_data)
```

The client receives the server’s public key, which will be used for secure key exchange.

Appendix G: Message Encryption, Decryption, and Integrity Check (3.4.3)

This code covers both message encryption and decryption, as well as using HMAC to verify message integrity:

```
from cryptography.hazmat.primitives import hashes, hmac
# Encrypt the message with AES
encrypted_message = encrypt_message(aes_key, nonce, b"Hello Server")
```

```
# HMAC for message integrity
h = hmac.HMAC(aes_key, hashes.SHA256())
h.update(encrypted_message)
message_hmac = h.finalize()
# Verify HMAC
def verify_hmac(hmac_key, ciphertext, hmac_value):
    h = hmac.HMAC(hmac_key, hashes.SHA256())
    h.update(ciphertext)
    h.verify(hmac_value)
```

The verify_hmac function checks the integrity of the encrypted message by comparing HMAC values.