UNIVERSITY
INSTITUTE
OF LISBON

PRISEC-III-Cryptographic-Techniques-for-Enhanced-Security

Humza Sohail

Master in Computer Engineering

Supervisors:
PhD Valderi Reis Quietinho Leithardt, Assistant Professor,
Iscte-IUL

PhD António Rui Trigo Ribeiro, Assistant Professor,
Iscte-IUL

May, 2025

TECHNOLOGY
AND ARCHITECTURE

Department of Information Science and Technology

PRISEC-III-Cryptographic-Techniques-for-Enhanced-Security

Humza Sohail

Master in Computer Engineering

Supervisors:
PhD Valderi Reis Quietinho Leithardt, Assistant Professor,
Iscte-IUL

PhD António Rui Trigo Ribeiro, Assistant Professor,
Iscte-IUL

May, 2025

Acknowledge

I would like to express my sincere gratitude to Professor Valderi Reis Quietinho Leithardt, my advisor, for his constant enthusiasm, unwavering support, and invaluable guidance throughout this project. His expertise and guidance have been invaluable in shaping the direction of my research.

I would also like to thank Professor António Rui Trigo Ribeiro, my co-supervisor, for his valuable feedback. The development of this thesis has been greatly aided by their collaborative mentoring.

Finally, I would like to thank everyone who has supported me throughout this journey, especially my loved ones, for their constant encouragement and belief in me.

Resumo

Este projeto apresenta o PRISEC III, uma estrutura criptográfica concebida para melhorar a segurança e a eficiência em ambientes IoT com recursos limitados. Com um modelo de criptografia em quatro camadas — "Convidado", "Básico", "Avançado" e "Administrador" —, equilibra segurança e desempenho, utilizando técnicas de criptografia progressivamente mais sofisticadas. O PRISEC III integra criptografia simétrica (AES, ChaCha20 e Blowfish) e assimétrica (ECC e RSA) para otimizar a eficiência e garantir a troca segura de chaves. As camadas inferiores utilizam criptografia leve para reduzir a sobrecarga computacional, enquanto as camadas superiores aplicam criptografia em várias camadas para operações críticas. A criptografia híbrida e a computação de borda melhoram ainda mais a escalabilidade e a otimização de recursos. Avaliações de desempenho confirmam a capacidade do PRISEC III de fornecer uma transmissão de dados segura e escalável, ao mesmo tempo que responde às limitações da IoT. Esta investigação oferece uma abordagem estruturada para a seleção de protocolos criptográficos, contribuindo para o desenvolvimento de soluções de segurança eficientes e adaptáveis para aplicações na Internet das Coisas.

Palavras-chave: algoritmos criptográficos, criptografia assimétrica e simétrica, criptografia em múltiplas camadas, eficiência computacional, técnicas de criptografia híbrida.

Abstract

This project presents PRISEC III, a cryptographic framework designed to improve the security and efficiency of IoT environments with limited resources. It features a four-tier encryption model — Guest, Basic, Advanced, and Admin — which balances security and performance by using increasingly sophisticated encryption techniques. PRISEC III integrates both symmetric (AES, ChaCha20 and Blowfish) and asymmetric (ECC and RSA) encryption to optimise efficiency and ensure secure key exchange. Lower tiers use lightweight encryption to reduce computational overhead, while higher tiers apply multi-layer encryption for critical operations. Hybrid encryption and edge computing further enhance scalability and resource optimisation. Performance evaluations confirm PRISEC III's ability to provide secure, scalable data transmission while addressing IoT limitations. This research offers a structured approach to selecting cryptographic protocols, contributing to the development of efficient, adaptive security solutions for IoT applications.

Keywords: Cryptographic algorithms, asymmetric and symmetric encryption, multi-level encryption, computational efficiency, hybrid encryption techniques.

Contents

List of Figures

List of Tables

List of Acronyms

AES-CTR: The Advanced Encryption Standard (AES) Counter Mode is a cryptographic technique utilized by numerous companies and organizations to safeguard their data.

AES-CCM: CBC-MAC is a mode of operation for the Advanced Encryption Standard.

AES-GCM: The Advanced Encryption Standard Galois/Counter Mode is a cryptographic algorithm.

Blowfish: An encryption algorithm for symmetric block ciphers

ChaCha20: A high-speed stream cipher for secure data encryption and decryption

XChaCha20: Advanced ChaCha20 with a longer nonce

ChaCha20-Poly1305: Combines ChaCha20 stream cipher with Poly1305 MAC

ECC: Elliptic Curve Cryptography

Curve25519: An elliptic curve that is widely used for high-performance key exchange

HMAC: Hash Message Authentication Key

HMAC-SHA512: HMAC with SHA-512 hash algorithm

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IoT: Internet of things

PKI: Public key infrastructure

RSA: Asymmetric Encryption Algorithm (Rivest-Shamir-Adleman)

SHA: Secure Hashing Algorithm

TLS: Transport Layer Safety

XOR: Exclusive OR logical operation

Poly1305: Message Authentication Code (MAC), used to ensure the integrity of the data

CTR: Block cipher operating mode known as counter mode.

CBC: Cipher Block chaining

GCM: Galois/Counter Mode (ciphering mode used to ensure confidentiality and integrity)

VM: Virtual machine

LAN: Local Area Network (LAN)

List of Software

Git: A System for Version Control

Microsoft Excel: Spreadsheet for analyzing and visualizing data

Microsoft Word: Text editor for writing and editing

Python: Cryptographic algorithm implementation programming language

Visual Studio Code (VS Code): A Code editor with extensive extensions for developing Python

GitHub: Code-hosting platform for revision control and cooperation

Adobe Photoshop: Software for image editing

VMware Workstation: Virtual machine software for virtual environment creation

Terraform: Software for infrastructure as code

Figma: Tool for UI/UX design and prototyping

PyCryptodome: Python library for cryptography functions (AES, Blowfish, ECC, etc.)

Highcharts: JavaScript-based chart library for visualization of encryption algorithm performance

Matplotlib: Static, animated, and interactive visualization Python library

NumPy: Numerical cryptographic testing Python library

Pandas: Python library for manipulating data sets

Wireshark: Network protocol analyzer for monitoring encrypted network traffic

Jupyter Notebook: Python code writing and testing environment

OpenSSL: Toolkit for implementation of SSL/TLS protocols and cryptographic functions

Docker: Containerization platform for deployment of lightweight, isolated application environments

Seaborn: Library used in conjunction with Matplotlib for data visualization

PyTest: Cryptographic function output validation framework

CHAPTER 1

Introduction

1.1. Synopsis

Recently, new security issues have emerged as Internet of Things (IoT) technologies have become an integral part of daily life and critical business operations. IoT applications link billions of devices, ranging from home automation systems to medical equipment. However, many IoT devices have limited processing power, which leaves them vulnerable to security flaws. These gadgets are vulnerable to malicious attacks as they often have limited hardware and are unable to execute sophisticated cryptographic protocols. This predicament assumes particular significance in the context of real-time processing applications, such as smart grids, healthcare systems, and autonomous vehicle technologies, where the safeguarding of sensitive data during its transmission becomes a matter of paramount importance [1].

The security of data in IoT ecosystems largely depends on cryptography. Cryptographic algorithms can be used to encrypt data, ensuring that only authorised individuals can access it. Cryptographic techniques are essential to prevent sensitive data, including medical and personal information, from being intercepted or altered [2]. The three main security objectives of confidentiality, integrity and authenticity are met by effective cryptography [3].

The project's investigation of cryptographic algorithms in edge computing is aimed at securing IoT data. Edge computing processes data closer to its source, thereby lowering latency and network congestion. This improves security by limiting the exposure of sensitive information over long distances, while also speeding up real-time processing [4]. The aim of integrating cryptographic algorithms directly into edge devices is to enhance the security and computational efficiency of IoT systems [5].

1.2. Background and Motivation

As IoT applications proliferate, their vulnerabilities have become increasingly apparent. Various attackers can hack IoT networks, including those who want to access data illegally, those who want to steal data, and those who want to make the network unavailable.Many IoT applications focus on network security but neglect the protection of sensitive data transmitted between devices, especially in scenarios involving high-value personal data [6]. Furthermore, the adoption of new data transmission architectures like Named Data Networking (NDN) has exposed IoT applications to new

risks, such as increased susceptibility to DoS and DDoS attacks [7]. These emerging threats, combined with the rapid expansion of IoT networks, highlight the need for effective cryptographic solutions [8].

The motivation for this project lies in the increasing need to secure IoT systems through cryptographic solutions that balance robustness with computational efficiency. The project aims to investigate cryptographic algorithms suitable for low-resource IoT devices, including symmetric (AES, ChaCha20) and asymmetric (RSA,ECC) encryption techniques, and to implement them within an edge computing framework. This will ensure that IoT devices can secure data transmissions without sacrificing performance due to computational constraints [9].

1.3. Objective of the Project

1.3.1. Cryptographic algorithms evaluation

Finding and assessing cryptographic algorithms that offer the best speed for encryption and decryption while preserving high security is the aim of this project. Performance studies have been conducted on symmetric algorithms such as AES (and its variants: AES-128-CTR, AES-256-GCM, AES-128-CCM, and AES-192-CTR), ChaCha20, and XChaCha20. Hybrid combinations (AES + ChaCha20 + ECC, AES + Blowfish, etc.) guarantee improved performance for IoT environments, while asymmetric algorithms like RSA and ECC (Curve25519) are included for secure key exchange and access control [10].

1.3.2. Integration with edge computing

To enable safe and effective data transfer between IoT systems, the project suggests integrating a cryptographic model into an edge computing framework. The system seeks to lower latency and computational overhead on IoT devices by handling encryption operations at the edge [11].

1.3.3. Performance analysis

Real-world Internet of Things applications will be used to test the chosen cryptographic algorithms. Several variables, including packet size, network conditions, and data transmission volume, will be considered in performance evaluations. To ascertain their effects on speed, power consumption, and security, combinations like AES-256-CTR + Blowfish + ChaCha20 for admin levels and AES-128-CCM + ChaCha20 for guest access will be examined [12].

1.3.4. Cryptographic security model development

The objective of this study is to design and implement an efficient and scalable cryptographic security model that is optimised for Internet of Things (IoT) environments. This model must ensure energy efficiency, high encryption speed, and minimal impact on device performance through multi-level encryption strategies [13].

1.4. Framework for PRISEC

The PRISEC (Privacy Security) framework is designed to provide cryptographic protocols for securing communication in IoT applications. It adopts a multi-layered security approach with configurations such as Guest, Basic, Advanced, and Admin, depending on data sensitivity and the environment. The framework ensures flexibility and scalability by classifying encryption methods based on computational efficiency, security level, and IoT constraints.

The selection of an optimal cryptographic approach is key to ensuring communication security in IoT applications [14]. This research presents the PRISEC III, an evolution of PRISEC I [15] and II [16], providing a structured framework that classifies encryption methods based on computational efficiency, security level, and IoT constraints.

In contrast to its predecessors, PRISEC III not only benchmarks encryption techniques, but also provides a framework within which the cryptographic approach is selected based on predefined user access levels (Guest, Basic, Advanced, Admin). For each level, the framework recommends suitable algorithms according to their performance (encryption/decryption time, memory usage) and security strength. This ensures a practical balance between high security standards and system performance. The framework under consideration has been designed to support a range of algorithms, including both symmetric and asymmetric algorithms. The former category, which is further subdivided into algorithms such as AES, Blowfish and ChaCha20, as well as the latter category, which includes ECC and RSA, are both supported by the framework. In addition to these, the framework is also capable of supporting hash-based algorithms, such as SHA-512 and HMAC [17].

1.5. Report Organization

Chapter 1: Introduction

Chapter 2: Cryptographic foundations and a review of the literature.

Chapter 3: Prototype development and cryptographic evaluation.

Chapter 4: Implementation and testing of cryptographic algorithms.

Chapter 5: Work evolution.

Chapter 6: Conclusion and future wor

CHAPTER 2

Cryptographic Foundations and a Review of the Literature

2.1. Cryptography

Cryptography is essential for ensuring the confidentiality and integrity of data by protecting it from unauthorised access and modification [18]. The process involves two basic steps: encryption and decryption.

2.1.1. Encryption

The process involves the utilisation of a specific algorithm. The use of a key is also a feature of this system. These devices are utilised to transform plaintext into ciphertext. Ciphertext is defined as encrypted data. The text is unreadable. It is due to this transformation that the information becomes incomprehensible to any individual who does not possess the correct key to decode it. Data can be secured using various encryption techniques, such as substitution (where characters or bits are replaced), transformation (where the structure of the data is changed) and permutation (where the order of data elements is rearranged) [19].

2.1.2. Decryption

The inverse process of encryption, decryption, is the process of converting the ciphertext back to its original plaintext form. This is achieved with the aid of a decryption key, which is frequently the same as the encryption key or a complementary key. The efficacy of data decryption is contingent on the security of the encryption key and the algorithm employed [20].

2.2. Types of Cryptographic Algorithms and Methods

2.2.1. Symmetric (secret key) encryption

Symmetric encryption utilises a single key for both processes. The sender and receiver must share a key for these processes. A significant challenge in symmetric encryption is the secure sharing of the key between communicating parties without risk of interception by malicious actors. The following list details the most commonly used symmetric encryption algorithms [21].

2.2.1.1. AES (Advanced Encryption Standard)

AES is the most prevalent encryption standard. The product offers excellent security and performance. The software is characterised by its high level of efficiency and its ability to function effectively in conjunction with hardware acceleration. This renders it optimal for utilisation in IoT devices.

2.2.1.2. Blowfish

This is a fast block cipher that uses variable-length keys. Despite its generally secure nature, it is frequently superseded by AES due to the superior security and performance characteristics of the latter.

2.2.1.3. ChaCha20

This stream cipher is distinguished by its high level of security and efficiency, particularly in constrained environments such as the Internet of Things (IoT). In circumstances where hardware acceleration for AES is unavailable, this alternative is often the preferred option.

2.2.2. Asymmetric encryption

In asymmetric encryption, two keys are utilised: a public key for encryption and a private key for decryption. The public key is responsible for encrypting the data, while the private key is employed for decryption. This configuration is especially advantageous for the secure distribution of keys. While the public key can be disseminated without concern for its secrecy, the private key must be kept confidential. The following list comprises a selection of widely utilised asymmetric algorithms: [22].

2.2.2.1. RSA-Rivets-Shamir-Adelman

RSA is an algorithm for public key cryptography. The basis of this approach is the mathematical problem of factoring large integers. However, it is computationally expensive. This renders it less suitable for constrained environments such as the IoT, although it is frequently employed for secure key exchange.

2.2.2.2. ECC (Elliptic Curve Cryptography)

ECC has been demonstrated to be more efficient in terms of computational overhead, as it provides an equivalent level of security to that of RSA, but with significantly smaller key sizes. Due to its efficiency and security, it is gaining popularity in the field of IoT devices. Curve25519 is one of the most prevalent implementations of ECC.

2.3. Cryptographic Algorithms Not Used in the Research

The present study has been constrained by the fact that certain cryptographic algorithms have been deemed unsuitable for inclusion, on account of their inherent limitations about security, efficiency, and compatibility with the constraints of IoT devices. It is acknowledged that these algorithms have exhibited vulnerabilities and performance issues, rendering them ill-suited for contemporary IoT applications [23].

2.3.1. DES (Data Encryption Standard)

The algorithm in question is a symmetric key algorithm that uses a 56-bit key. The short key length of the system renders it highly vulnerable to brute-force attacks, and it has been largely deprecated.

2.3.2. 3DES (Triple DES)

This is a more secure version of the DES algorithm, which employs the DES algorithm on three separate occasions. Nevertheless, the algorithm's reduced processing speed and ongoing deprecation by the National Institute of Standards and Technology (NIST) render it ill-suited for contemporary applications.

2.3.3. RC4

This is a stream cipher that was widely utilised, but it has known security flaws, particularly due to weaknesses in its key scheduling algorithm and distortions in its output. It is no longer considered to be secure for use in contemporary systems.

2.3.4. Twofish & Serpent

It is evident that both of these block ciphers offer robust security; however, they are less efficient than AES in terms of speed. Given that performance is a critical factor in IoT systems, these algorithms are often regarded as impractical for resource-constrained environments, despite their technical security.

2.3.5. MD5 & SHA-1

It is important to note that these hash functions are vulnerable to collision attacks. It is important to note that two different inputs may result in the same hash value being produced. Consequently, their utilisation in applications where security is a concern is no longer recommended.

2.3.6. BLAKES2 & SHA-3

These are cryptographically secure hash functions that offer superior security guarantees in comparison to MD5 and SHA-1. However, in IoT environments where speed and computational efficiency are critical, alternatives such as SHA-512 and HMAC are often preferred, as they are optimised for performance while maintaining strong security.

2.4. Selected Cryptographic Algorithms for IoT Applications

The selection of cryptographic algorithms in this research (AES, Blowfish, ChaCha20, ECC, RSA, SHA-512, and HMAC) is grounded in an extensive literature review and empirical testing. It is important to note the recurring theme in extant literature [13][14][15] regarding the need to strike a balance between the security strength and the computational efficiency of IoT devices, particularly in cases where these devices are limited in terms of the available resources.

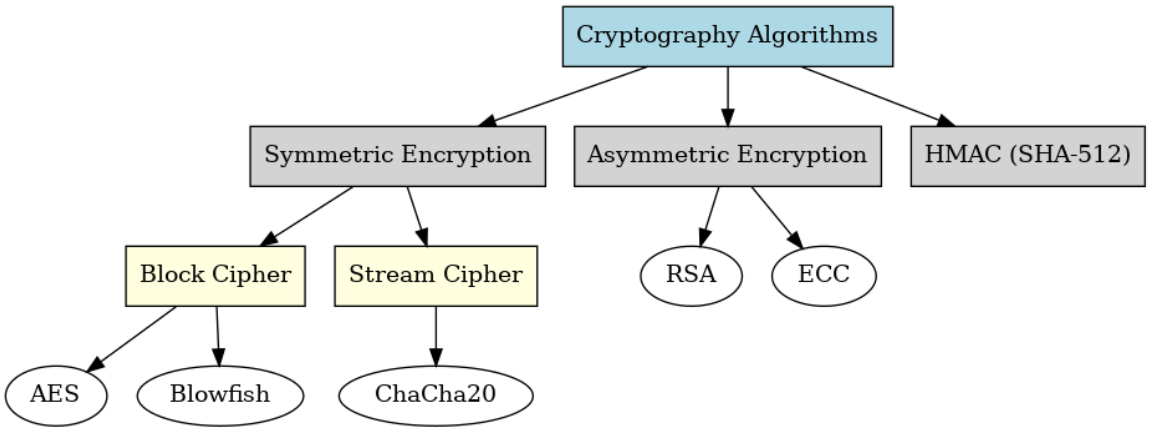


FIGURE 2.1. *Encryption Algorithm Techniques [24].*

The algorithms selected for inclusion in PRISEC III were:

2.4.1. AES (Advanced Encryption Standard)

In 2001, the National Institute of Standards and Technology (NIST) released the AES algorithm to overcome the security limitations of the older 3DES algorithm. AES is a block cipher. The range of key sizes available is a notable feature. It is evident that the parameters A, B, and C are 128 bits, 192 bits, and 256 bits, respectively. The system performs both substitution and permutation operations. The number of rounds is contingent on the key size [25]. The number of rounds corresponding to each key size is shown in Table 2.0.

TABLE 2.1. *Number of rounds per key size.*

Algorithm	**Type**	**Key Size**	**Round**
AES (Advanced Encryption Standard)	Symmetric Block	128, 192, 256 bits	10, 12, 14
ChaCha20	Symmetric Stream	256 bits	20
Blowfish	Symmetric Block	32-448 bits	16
RSA	Asymmetric	2048 bits or more	Variable (based on key size)
ECC	Asymmetric	256 bits (common)	Depends on curve

2.4.2. Blowfish

This is a symmetric block cipher that was developed by Bruce Schneier in 1993. The cryptosystem utilises a Feistel network structure and operates on 64-bit blocks. The key size in Blowfish can range from 32 bits to 448 bits, providing flexibility. Blowfish's encryption function involves 16 rounds, rendering it suitable for high-speed encryption in environments with limited resources [26].

2.4.3. ChaCha20

This stream cipher was developed by Daniel J. Bernstein as a variant of the Salsa20 algorithm. In contrast to block ciphers, ChaCha20 encrypts data bit by bit, rendering it more efficient for stream processing. The algorithm is renowned for its swiftness and security, utilising a 256-bit key for encryption. ChaCha20 is a 20-round algorithm that ensures secure encryption, even in high-performance environments [27].

2.4.4. RSA

The asymmetric encryption algorithm was first introduced in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman. The underlying principle is rooted in the mathematical problem of factoring large prime numbers. RSA employs two distinct keys: a public key for the purpose of encryption, and a private key for that of decryption. The key sizes employed in RSA typically range from 1024 bits to 4096 bits, with larger keys offering enhanced security [28].

2.4.5. Elliptic Curve Cryptography (ECC)

The present text concerns an asymmetric encryption technique based on the algebraic structure of elliptic curves over finite fields. ECC is a cryptographic algorithm that provides strong security with much smaller key sizes than traditional algorithms such as RSA. To illustrate this point, consider the case of a 256-bit key in Elliptic Curve Cryptography (ECC); this can be considered equivalent to a 3072-bit key in Rivest-Shamir-Adleman (RSA) [29].

2.4.6. Hash-based Message Authentication Code (HMAC)

Hash-based Message Authentication Code (HMAC) is a cryptographic algorithm that utilises a secret key in conjunction with the SHA-512 hashing function to verify the integrity of data and to prevent unauthorised modification. It is a component that is utilised extensively in secure protocols, such as HTTPS [30].

Table 2.2 below summarizes which cryptographic algorithms have been covered, implemented, or recommended in the selected studies. A **"Yes"** indicates that the author included that particular algorithm in their study (through implementation, evaluation, or proposal). A **"No"** indicates that the algorithm was not part of that author's scope or contribution.

TABLE 2.2. *Summary of cryptographic algorithms.*

Author/Tests	**AES**	**Blowfish**	**RSA**	**ChaCha20**	**ECC**	**SHA-512 (HMAC)**
NIST. (2020)	Yes	No	No	Yes	No	No
Schneier, B. (2021)	No	Yes	No	No	No	No
Rivest, R. L., Shamir, A., & Adleman, L. (2022).	No	No	Yes	No	No	No
Bernstein, D. J. (2023).	No	No	No	Yes	No	No
Koblitz, N. (2023).	No	No	No	No	Yes	No
Harsh, P., & Khandelwal, N. (2023).	No	No	No	No	No	Yes
Smith, J. & Patel, R. (2024)	Yes	No	Yes	Yes	Yes	Yes
Humza Sohail PRISEC III(2025)	Yes	Yes	Yes	Yes	Yes	Yes

The following table illustrates the diversity of cryptographic algorithm coverage in the extant literature. The objective of this study is not to provide a performance ranking of algorithms, but rather to document which algorithms were studied, recommended, or implemented by each author. The

PRISEC III framework is informed by these findings, and it selects a balanced set of algorithms suitable for IoT devices with limited resources. These algorithms take into consideration security, computational efficiency, and memory overhead.

2.5. Review of Literature

This section outlines previous related work on cryptographic algorithms relevant to IoT applications and discusses the selection of algorithms for this study.

The related works were retrieved from IEEE Xplore, SpringerLink, ScienceDirect, and Google Scholar using the following search strings: The following subjects will be covered in this study: cryptographic algorithms for IoT, lightweight encryption for IoT, cryptographic libraries in Python, and the performance analysis of encryption algorithms in IoT. The selection of articles was constrained to those that had undergone the peer-review process and had been published within the last five years.

Python was selected as the primary implementation language for this study due to its simplicity, readability, and the availability of mature cryptographic libraries such as PyCryptodome. Python's high-level nature facilitates rapid prototyping and testing of cryptographic algorithms, although it is acknowledged that performance in lower-level languages (such as C/C++) may vary.

In [31], the authors emphasized Python's simplicity, readability, and extensive library ecosystem, which make it ideal for developing and testing cryptographic applications. The flexibility and high-level nature of Python enables quick implementation of complex cryptographic algorithms, minimizing development time and increasing productivity. The authors highlighted Python's dynamic typing and flexibility, which allow developers to easily adapt cryptographic code as security requirements evolve.

PyCryptodome, a widely used cryptographic library in Python, was reviewed in [32]. The library provides robust tools for the implementation of various well-known cryptographic algorithms such as AES, Blowfish, RSA, ChaCha20, and Elliptic Curve Cryptography (ECC). It supports secure key management, encryption, decryption, digital signatures, and hash functions. It also includes encryption modes such as GCM, CCM, and CTR. This provides flexibility for securing data in different contexts.

PyCryptodome is actively maintained. It is the first choice for cryptographic operations in Python. Further research in [33] highlighted the importance of using a local server in a virtualized environment to simulate real-world deployment conditions for testing cryptographic operations. By configuring a local server within a virtualized environment, researchers tested cryptographic algorithms to evaluate their performance and security in a controlled, isolated environment. Virtualization helps minimize risks to real production systems and enables developers to test encryption, decryption, and key management systems before full deployment.

In [34], Recent studies have emphasised the importance of testing cryptographic algorithms in virtualised environments. Such environments facilitate the evaluation of systems under varying data sizes, network latency, and resource constraints. The utilisation of controlled setups enables researchers to evaluate the security and performance of cryptographic solutions prior to their deployment within production systems.

2.6. Explore the Issues

The objectives and development of PRISEC III are guided by the following research questions:

1. Which cryptographic algorithms provide the optimal balance between security and performance for resource-constrained IoT devices?
2. How can edge computing frameworks be integrated with cryptographic techniques to enhance data security without compromising efficiency?
3. How do packet size, network conditions, and data volume affect the performance of cryptographic algorithms in IoT environments?
4. How can multi-layer encryption models improve the robustness of IoT security systems while remaining scalable?
5. What role do role-based security play in increasing the flexibility and adaptability of cryptographic frameworks for IoT applications?
6. How can a scalable, role-based multi-layer cryptographic model be designed to provide optimal security and performance in IoT environments characterized by limited computational resources and varying data protection needs?

CHAPTER 3

Prototype Development and Cryptographic Evaluation

This chapter discusses the mathematical foundations and cryptographic mechanisms that are an integral part of the PRISEC III system. The discussion covers key algorithms, their mathematical principles, and their suitability for securing edge computing environments. The system is implemented with multiple levels of users who determine the encryption methods and algorithms used. To visualize and understand the flow of operations, several UML diagrams are included to support the mathematical explanations.

3.1. Use Case Diagram

The PRISEC III use case diagram provides a high-level overview of the various user interactions with the system. It displays the actions available to each user and the manner in which they interact with the system's functionalities.

1. **Select Level:** The user selects the encryption level.
2. **Select Method:** Users select their encryption level (Guest, Basic, Admin, and Advanced).
    a. **Guest:** Minimal encryption for public or non-sensitive data (e.g., AES-128).
    b. **Basic:** Standard encryption for low-sensitivity data (e.g., AES-256).
    c. **Advanced:** High-level encryption using hybrid schemes (e.g., AES-256 + RSA) for sensitive business data.
    d. **Admin:** Maximum security, multi-layer encryption (AES-256-GCM + ECC + HMAC), suitable for critical operations and admin controls.
3. **Select Algorithm:** The user selects an algorithm based on the system's available options.
4. **Select Packet Size:** In the PRISEC III testing framework, in order to ensure consistency and fairness across different test scenarios, a predefined sample data packet was used for all encryption and decryption operations. This enables a precise comparison of performance across various algorithms and packet sizes.
    a. **Data Type**: Simulated structured data file (containing a mix of random text, numbers data format).
    b. **Content Consistency**: The same base data was used in all test runs; only the packet size (1MB, 25KB, 50MB, 75MB, 100MB) was varied by scaling this base file.

c. **Purpose**: This approach ensures that observed variations in performance metrics (encryption time, decryption time, memory usage, ciphertext expansion) are solely due to algorithm behavior and packet size, not due to differing content.

5. **Click the Encrypt Button:** The user initiates the encryption process.
6. **Transfer Data to Edge Computing:** After encryption, the data is transferred to Edge Computing for processing.
7. **Return Encryption and Decryption Times:** After processing, the system returns the encryption and decryption time calculations.

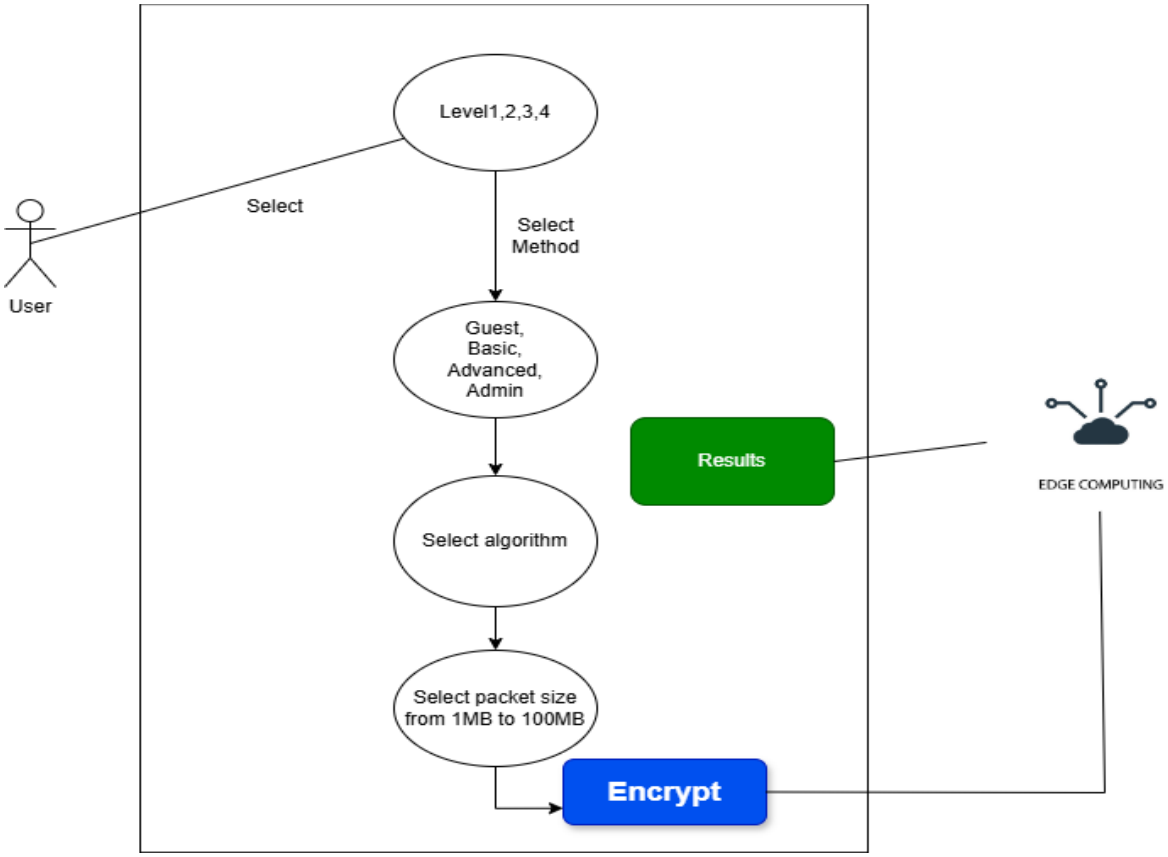


FIGURE 3.1. *PRISEC III Use Case.*

The activity diagram is a representation of the sequential flow of activities within the system. The visualisation demonstrates the user interaction from initiation to conclusion, incorporating the diverse pathways that users can traverse based on their selection of encryption level, method, algorithm, and packet size.

3.2. Activity Diagram

1. **Start:** The process begins when the user interacts with the system.
2. **Select Level:** The user selects the encryption level.
3. **Select Method:** Users select their encryption level (Guest, Basic, Admin, and Advanced).
    a. **Guest:** Minimal encryption for public or non-sensitive data (e.g., AES-128).
    b. **Basic:** Standard encryption for low-sensitivity data (e.g., AES-256).
    c. **Advanced:** High-level encryption using hybrid schemes (e.g., AES-256 + RSA) for sensitive business data.
    d. **Admin:** Maximum security, multi-layer encryption (AES-256-GCM + ECC + HMAC), suitable for critical operations and admin controls.
4. **Select Algorithm:** Allows the user to select the algorithm.
5. **Select Packet Size:** In the PRISEC III testing framework, in order to ensure consistency and fairness across different test scenarios, a predefined sample data packet was used for all encryption and decryption operations. This enables a precise comparison of performance across various algorithms and packet sizes.
    a. **Data Type:** Simulated structured data file (containing a mix of random text, numbers data format).
    b. **Content Consistency**: The same base data was used in all test runs; only the packet size (1MB, 25KB, 50MB, 75MB, 100MB) was varied by scaling this base file.
    c. **Purpose:** This approach ensures that observed variations in performance metrics (encryption time, decryption time, memory usage, ciphertext expansion) are solely due to algorithm behavior and packet size, not due to differing content.

6. **Click Encrypt:** When all inputs are complete, the user clicks the Encrypt button.
7. **Transfer Data to Edge Computing:** The system sends the encrypted data to Edge Computing for processing.
8. **Return Encryption and Decryption times:** The system calculates and returns the encryption and decryption times.
9. **Finish:** The process ends when the results are displayed.

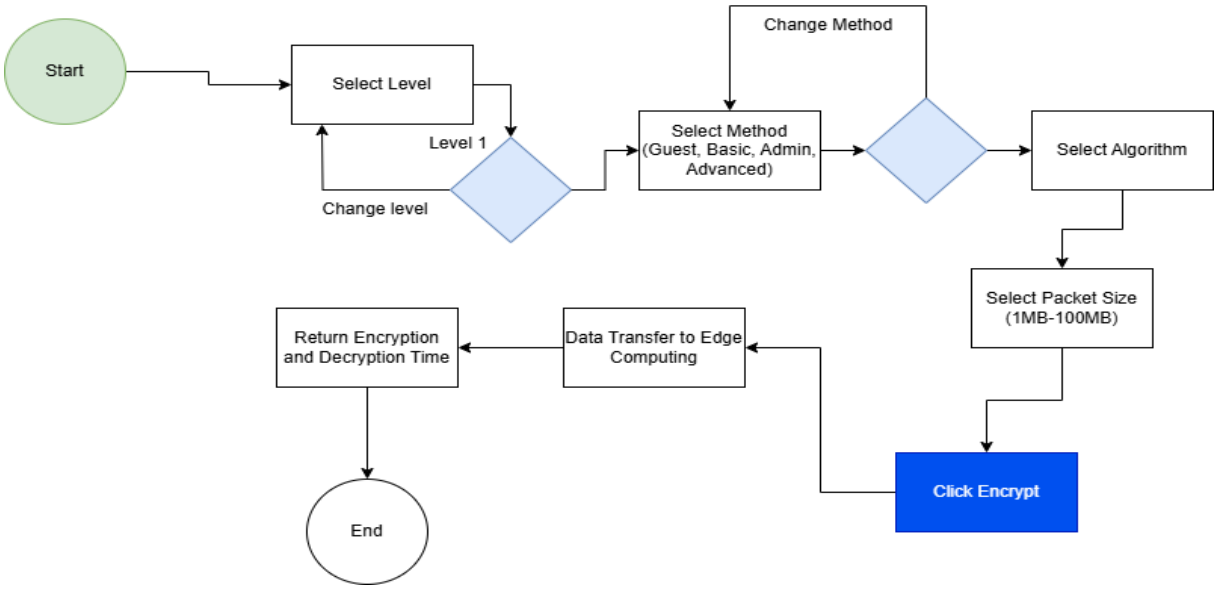


FIGURE 3.2. *PRISEC III Activity Diagram.*

3.3. Sequence Diagram

The sequence diagram provides a detailed representation of the interaction sequence of system components. The diagram has been found to be a valuable tool for comprehending the manner in which diverse components (e.g., user interface, algorithms, edge computing) interact during the encryption process.

1. The user selects the encryption level and method.
2. The system responds with available algorithm selection options.
3. User selects algorithm and packet size.
4. System processes input data, prepares for encryption, and initiates the encryption process.
5. Data is transmitted to Edge Computing.
6. Edge Computing returns encryption and decryption time calculations.
7. The system displays the result to the user.

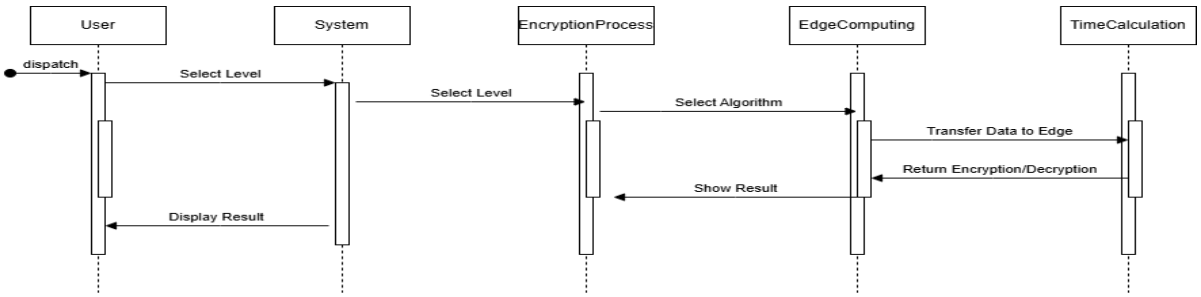


FIGURE 3.3. *PRISEC III Sequence Diagram.*

3.4. Class Diagram

The class diagram is a visual representation of the system's structure, illustrating the interactions between components or classes. The accompanying diagram assists in comprehending the interrelationships among various classes, including User, Algorithm, PacketSize, Encryption Process, and Time Calculation.

1. **User:** Contains attributes such as user level, selected method, selected algorithm, and packet size.
2. **Select Method:** Users select their encryption level (Guest, Basic, Admin, and Advanced).
    a. **Guest:** Minimal encryption for public or non-sensitive data (e.g., AES-128).
    b. **Basic:** Standard encryption for low-sensitivity data (e.g., AES-256).
    c. **Advanced:** High-level encryption using hybrid schemes (e.g., AES-256 + RSA) for sensitive business data.
    d. **Admin:** Maximum security, multi-layer encryption (AES-256-GCM + ECC + HMAC), suitable for critical operations and admin controls.
3. **Select Algorithm:** The user selects an algorithm based on the system's available options.
4. **Select Packet Size:** In the PRISEC III testing framework, in order to ensure consistency and fairness across different test scenarios, a predefined sample data packet was used for all encryption and decryption operations. This enables a precise comparison of performance across various algorithms and packet sizes.
    a. **Data Type**: Simulated structured data file (containing a mix of random text, numbers data format).
    b. **Content Consistency**: The same base data was used in all test runs; only the packet size (1MB, 25KB, 50MB, 75MB, 100MB) was varied by scaling this base file.
    c. **Purpose**: This approach ensures that observed variations in performance metrics (encryption time, decryption time, memory usage, ciphertext expansion) are solely due to algorithm behavior and packet size, not due to differing content.
5. **Encryption Process:** Handles the encryption and decryption process, including time calculations.
6. **Edge Computing**: Represents the edge computing component where data is transferred for processing.
7. **Time Calculation:** A class that calculates and stores encryption and decryption times.

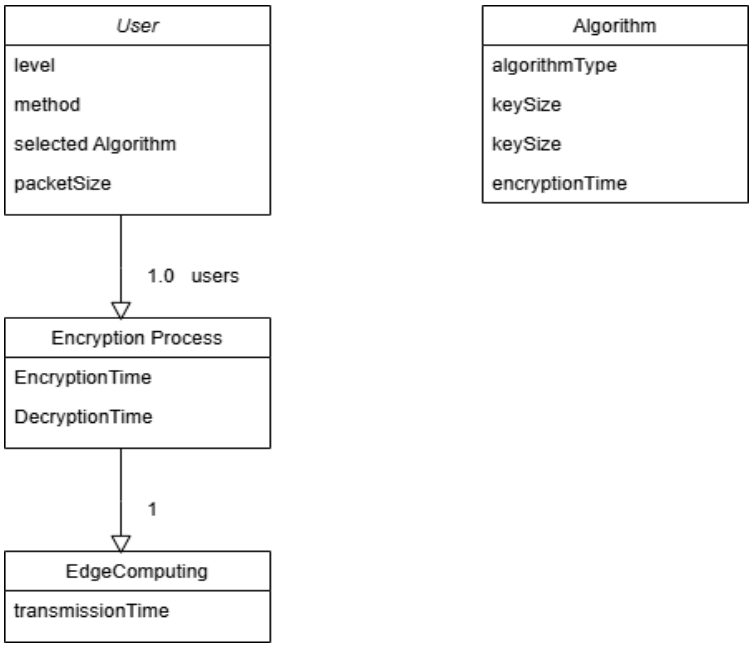


FIGURE 3.4. *PRISEC III Class Diagram.*

3.5. State Diagram

The state diagram illustrates how the system evolves in response to user input and internal processes. It provides a visual representation of the different states that the system undergoes during encryption operations.

1. **Idle:** The system is waiting for user input.
2. **Level Selected:** The user has selected an encryption level.
3. **Select Method:** Users select their encryption level (Guest, Basic, Admin, and Advanced).
    a. **Guest:** Minimal encryption for public or non-sensitive data (e.g., AES-128).
    b. **Basic:** Standard encryption for low-sensitivity data (e.g., AES-256).
    c. **Advanced:** High-level encryption using hybrid schemes (e.g., AES-256 + RSA) for sensitive business data.
    d. **Admin:** Maximum security, multi-layer encryption (AES-256-GCM + ECC + HMAC), suitable for critical operations and admin controls.
4. **Select Algorithm:** The user selects an algorithm based on the system's available options.
5. **Select Packet Size:** In the PRISEC III testing framework, in order to ensure consistency and fairness across different test scenarios, a predefined sample data packet was used for all encryption and decryption operations. This enables a precise comparison of performance across various algorithms and packet sizes.

a. **Data Type**: Simulated structured data file (containing a mix of random text, numbers data format).
b. **Content Consistency**: The same base data was used in all test runs; only the packet size (1MB, 25KB, 50MB, 75MB, 100MB) was varied by scaling this base file.
c. **Purpose**: This approach ensures that observed variations in performance metrics (encryption time, decryption time, memory usage, ciphertext expansion) are solely due to algorithm behavior and packet size, not due to differing content.

6. **Encrypting:** The system is encrypting.
7. **Sending Data to Edge Computing:** The encrypted data is being sent to Edge Computing for further processing.
8. **Calculating Encryption/Decryption Time:** The system is calculating the encryption and decryption times.
9. **Display Result:** The system displays the final encryption and decryption times.
10. **Exit:** Finishes the process.

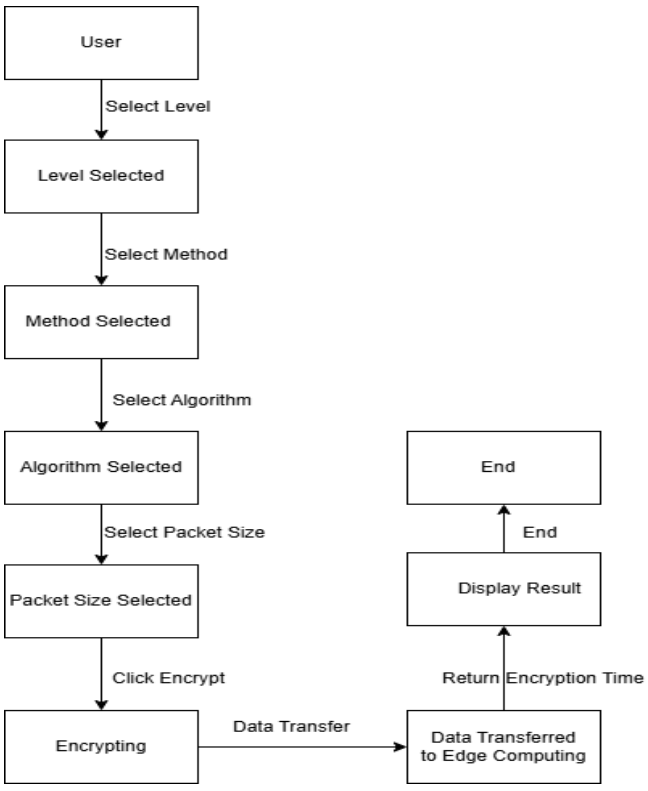


FIGURE 3.5. *PRISEC III State Diagram.*

The cryptographic algorithms utilised in PRISEC III are founded upon a multitude of mathematical concepts, encompassing algebraic structures, finite field operations, prime factorization problems, and

hash functions. These techniques ensure the confidentiality, integrity and authenticity of data in edge computing systems. The following section provides a comprehensive overview of the mathematics underpinning each of the selected algorithms:

3.6. Symmetric Encryption (AES, Blowfish, ChaCha20)

3.6.1. AES (Advanced Encryption Standard)

AES is a block cipher. The encryption process utilises blocks of a fixed size, defined as 128 bits, and employs keys of 128, 192, or 256 bits. It is evident that AES is mathematically secure due to the implementation of several core operations over a Galois field GF(28).

1. **Key transformations:** AES encryption consists of several rounds (10 for a 128-bit key, 12 for a 192-bit key, and 14 for a 256-bit key). There are four operations in each round:
    a. **Sub Bytes:** Non-linear substitution using an S-box derived from finite field arithmetic over GF (28).
    b. **Shift Rows:** Shift state matrix rows circularly.
    c. **Mix Columns:** A linear transformation with matrix multiplication over GF (28).
    d. **Add Round Key**: XOR between the state and a round-specific key derived from the main key.
2. **Security Strength:** The strength of AES lies in its resistance to differential and linear cryptanalysis, achieved through a combination of substitution-permutation and key scheduling mechanisms.

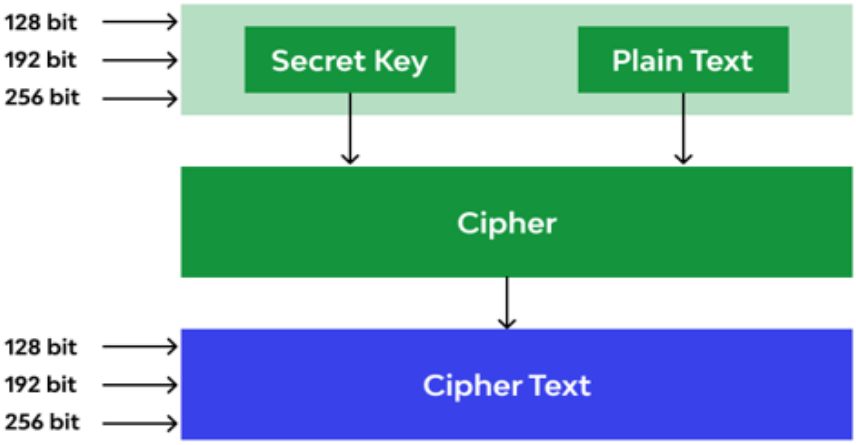


FIGURE 3.6. *Advanced Encryption Standard (AES) [34].*

3.6.2. Blowfish

Blowfish operates as a Feistel network and processes data in 64-bit blocks using a variable-length key (32 to 448 bits).

1. **Feistel Structure:** The data is divided into two halves, L and R, and processed iteratively using the equation:

$$Li+1=Ri, Ri+1=Li \oplus F(Ri, Ki)$$

Where F is a complex function with Substitute (S) and Permute (P) fields.

2. **Security and Efficiency:** The design of Blowfish ensures fast encryption while maintaining strong security properties.

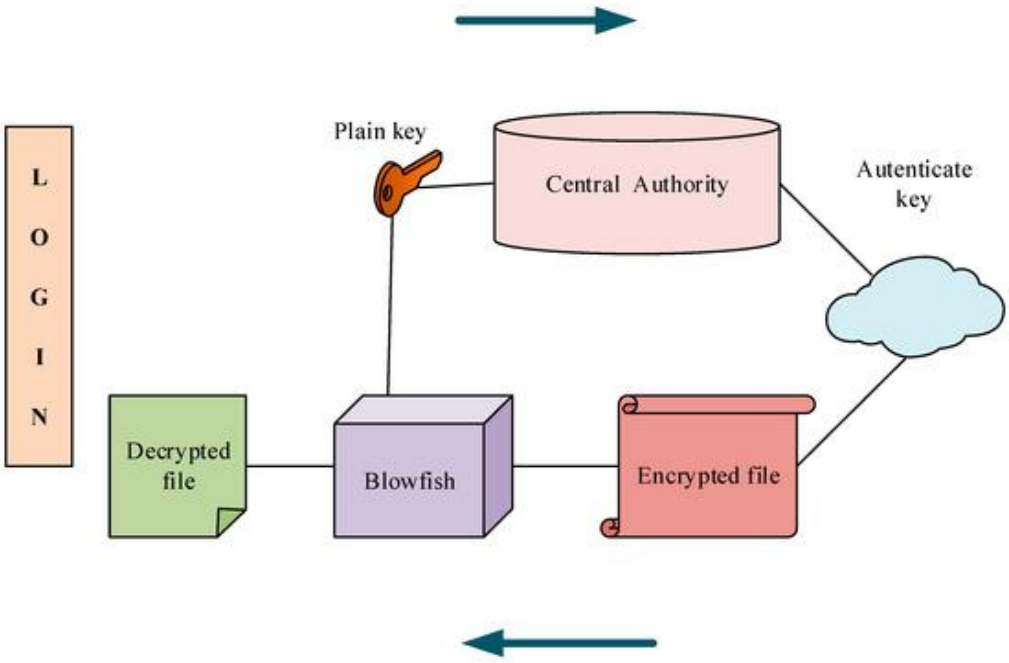


FIGURE 3.7. *Blowfish [38].*

3.6.3. ChaCha20

ChaCha20 is a stream cipher that uses simple arithmetic operations for efficient and secure encryption.

1. **Mathematical Operations:** ChaCha20's core operations are:
    a. Addition modulo 2^{32}
    b. XOR operation
    c. Bitwise rotations
    d. The cipher applies 20 rounds of transformations to generate a secure keystream.
2. **Security:** The simplicity of ChaCha20's operations makes it resistant to side-channel attacks and ensures high performance.

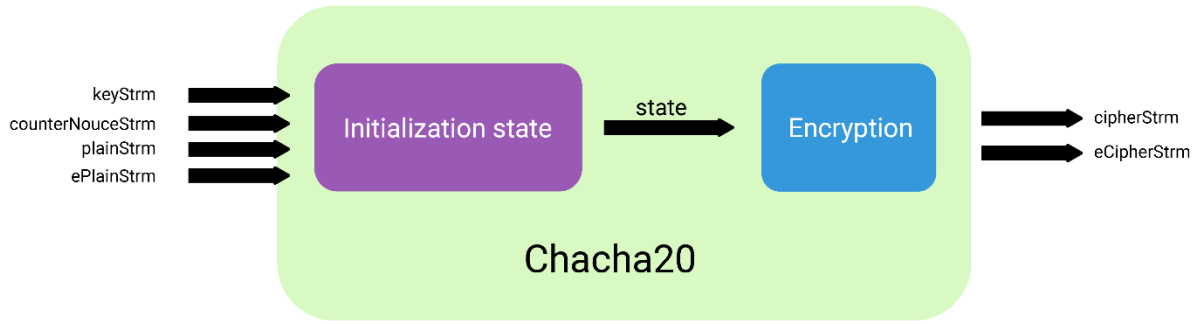


FIGURE 3.8. *Chacha20 [39].*

3.7. Asymmetric Encryption (ECC, RSA)

3.7.1. Elliptic Curve Cryptography (ECC)

The basis of ECC is the mathematics of elliptic curves over finite fields. The following equation defines an elliptic curve:

$$Y2 = x3 + ax + b \bmod p$$

Where a and b are constants that satisfy the condition 4a3+27b2≠.

1. **Point addition and scalar multiplication operations:** ECC operations involve adding and multiplying points on the curve. The scalar multiplication KP (repeated point addition) is computationally intensive and forms the basis of ECC security, given a point PP on the curve.
2. **Security:** The security of ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). This makes it more efficient than RSA for equivalent levels of security.

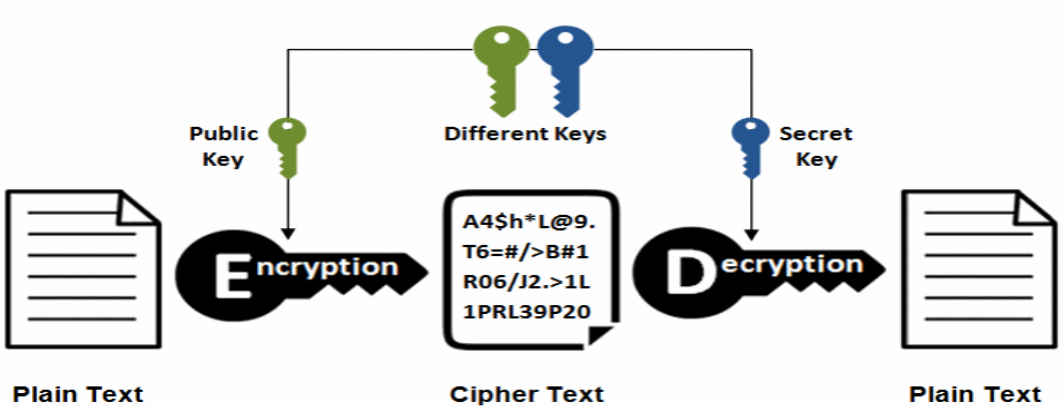


FIGURE 3.9. *ECC [39].*

3.7.2. RSA (Rivest-Shamir-Adleman)

1. The basis of RSA is the difficulty of factoring large composite numbers.
2. **Mathematical Foundations:**
    a. Choose two large prime numbers p and q.
    b. Compute n=p×q and φ(n)=(p-1)(q-1).
    c. Choose an integer such that 1<e<φ (n) and gcd (e, φ (n))=1.
    d. Compute the private key d such that d×e≡1mod φ (n).
3. **Encryption and Decryption:**
    a. c = me mod n, m = cd mod n
    b. Where m is plaintext, c is cipher text, and n is modulus.
4. **Security:** The strength of RSA comes from the difficulty of prime factorization for large n.

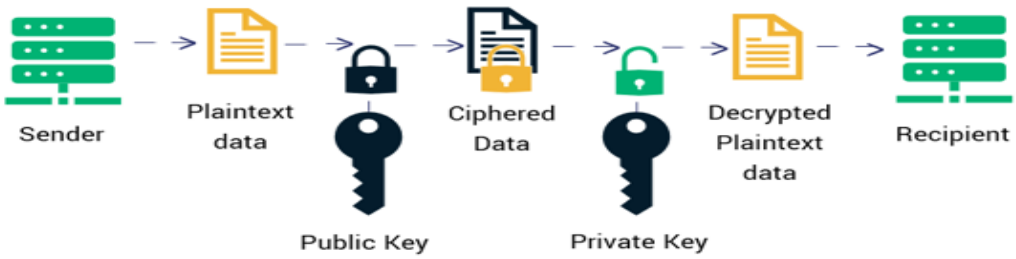


FIGURE 3.10. *RSA [40].*

3.8. HMAC (SHA-512)

3.8.1. SHA-512 (Secure Hashing)

1. SHA-512 is a cryptographic hash function. It produces a 512-bit output.
2. **Mathematical Operations:**
    a. Processes data in 1024-bit blocks.
    b. Uses modular additions, bitwise operations, and message expansion functions.
    c. Apply 80 rounds of transformations to generate the hash value.
3. **Security:** SHA-512 is suitable for secure message integrity verification because it is resistant to collision, pre-image, and second pre-image attacks.

3.8.2. HMAC (Hashed Message Authentication Code)

HMAC combines a cryptographic hash function with a secret key. It ensures data integrity.

1. **Mathematical Formula:**

$$HMAC(K, m) = H((K \oplus ipad) \,||\, H((K \oplus ipad) \,||\, m)).$$

Where H is the hash function. iPad are padding constants.

2. **Security:** HMAC protects against message modification and replay attacks.

FIGURE 3.11. *SHA-512 [41].*

The cryptographic algorithms that have been selected for PRISEC III include AES, Blowfish, ChaCha20, ECC, RSA, and HMAC-SHA512. The selection of these algorithms was driven by their compatibility with edge computing environments, with the objective of achieving a balance between security, performance, and computational efficiency [19].

The cryptographic algorithms were applied at four different levels of security for each version of PRISEC, as described below:

The following classification system is used to determine the level of guest at a given establishment: The implementation of lightweight encryption algorithms, such as Blowfish or AES-128, is employed to minimise processing overhead.

1. **Guest Level:** Uses lightweight encryption (e.g., Blowfish, AES-128) to minimize processing overhead.
2. **Basic Level:** In order to address moderate security requirements, it is recommended that stronger encryption mechanisms such as AES-192 and ChaCha20 be implemented.
3. **Advanced Level:** Robust encryption techniques (e.g., AES-256, RSA-2048) are employed for high-security scenarios.
4. **Admin Level:** The most secure algorithms (e.g., ECC, HMAC-SHA512) are utilised to protect sensitive data.
5. Each algorithm was subjected to a performance evaluation employing file sizes of 1 MB, 25 KB, 50 MB, 75 MB, and 100 MB. The tests were executed on five occasions, and the results were averaged to ensure accuracy.

3.9. Encryption/Decryption Time Comparison for Different User Levels

The table 3.1, 3.2, 3.3, 3.4 summarizes the encryption and decryption times for each **PRISEC** version across the four user levels. Each version of **PRISEC** uses different algorithms for encryption and decryption at each level.

TABLE 3.1. *Guest.*

Version	**Packet Size (MB)**	**Encryption time (MS)**	**Decryption time (MS)**	**Security Feature**
PRISEC I	10	13.337	23.841	Base64
PRISEC II	50	1.5485	1.7896	AES-256
PRISEC III	100	267.27	325.16	AES-256-GCM + RSA)

Base64 is an encoding method, not encryption, providing no real security but fast processing times (13.337 Ms encryption, 23.841 Ms decryption). In contrast, AES-256 offers strong encryption with fast times (1.5485 Ms Encryption, 1.7896 Ms Decryption), making it ideal for secure data protection. AES-256-GCM with RSA offers very strong security but slower performance (267.27 Ms encryption, 325.16 Ms decryption), making it suitable for high-security applications.

TABLE 3.2. *Basic.*

Version	**Packet Size (MB)**	**Encryption time (MS)**	**Decryption time (MS)**	**Security Feature**
PRISEC I	10	0.048	0.089	Base64 + AES-128-GCM,
PRISEC II	50	3.518	4.072	AES-256+AES-CTR
PRISEC III	100	12.967	1338.435	AES-256-CCM + ChaCha20-Poly1305

PRISEC I (Base64 + AES-128-GCM) provides fast encryption (0.048 Ms) and decryption (0.089 Ms), but lacks strong security due to Base64 encoding. **PRISEC** II (AES-256 + AES-CTR) provides very fast encryption (3.518 Ms) and decryption (4.072 Ms) with strong AES-256 encryption, making it ideal for secure data protection. **PRISEC** III (AES-256-CCM + ChaCha20-Poly1305) provides excellent security with authenticated encryption but has slower decryption (1338.4347 Ms). This makes it suitable for high-security applications with larger data sizes.

TABLE 3.3. *Advanced.*

Version	Packet Size (MB)	Encryption time (MS)	Decryption time (MS)	Security Feature
PRISEC I	10	78.615	119.293	Base64 + AES-128-GCM + AES-192-GCM
PRISEC II	50	16.512	12.967	AES-256+AES-CTR+HMAC-SHA256
PRISEC III	100	1927.301	1815.727	AES-256-CCM + XChaCha20 + ChaCha20

PRISEC I (Base64 + AES-128-GCM + AES-192-GCM) offers encryption (78.615 Ms) and decryption (119.293 Ms) times but is less secure due to the inclusion of Base64 encoding. PRISEC II (AES-256 + AES-CTR + HMAC-SHA256) provides strong security with faster encryption (16.512 Ms) and decryption (12.967 Ms), making it a solid choice for high-speed, secure data protection. PRISEC III (AES-256-CCM + XChaCha20 + ChaCha20) offers very strong encryption with high security but slower times (1927.301 Ms encryption, 1815.727 Ms decryption), suitable for applications requiring maximum security.

TABLE 3.4. *Admin.*

Version	Packet Size (MB)	Encryption time (MS)	Decryption time (MS)	Security Feature
PRISEC I	10	107.077	148.449	Base64 + AES-128 GCM + AES-192 GCM + AES-256 GCM
PRISEC II	50	47.292	30.082	AES-256+AES-CTR+HMAC-SHA256+ECC
PRISEC III	100	999.620	999.620	AES-256-GCM + ChaCha20 + ECC (Curve25519

PRISEC I (Base64 + AES-128 GCM + AES-192 GCM + AES-256 GCM) provides strong encryption, but is less secure due to the Base64 encoding, with encryption (107.077 Ms) and decryption (148.449 Ms) times that are moderate.

PRISEC II (AES-256 + AES-CTR + HMAC-SHA256 + ECC) offers fast encryption (47.292 Ms) and decryption (30.082 Ms) with robust security, including elliptic curve cryptography (ECC) for additional protection. PRISEC III (AES-256-GCM + ChaCha20 + ECC) provides very strong security, but its slower encryption and decryption times (999.620 Ms each) make it ideal for high-security applications where performance is not the top priority.

For better presentation and understanding of the results, a web interface was developed using Flask. This interface allows users to view and analyse the performance results interactively. The results are displayed in tables section IV, providing insights into execution time and ciphertext size across different file sizes. Additionally, a snapshot of the webpage provides a visual overview of the results in a clear and user-friendly manner.

Note on Packet Sizes and Performance Tables: Although the application permits users to select any packet size between 1 MB and 100 MB, the tables (Tables 3.1 to 3.4) present results for specific packet sizes, namely 10 MB, 50 MB, and 100 MB. The sizes of the samples were selected with the intention of representing realistic use case scenarios aligned with the three PRISEC versions:

1. PRISEC I (Base64): Designed for lightweight, fast-processing tasks where security is not a priority — tested using 10 MB packets to reflect small-data scenarios.
2. PRISEC II (AES-256): Aimed at general-purpose secure communication — tested with 50 MB packets to simulate medium-sized data operations.
3. PRISEC III (AES-256-GCM with RSA): Intended for high-security environments such as cloud storage or enterprise systems — tested using 100 MB packets to represent large-data use cases.

The selection of these packet sizes serves to illustrate the performance of each encryption method under its intended practical workload. However, for a strict algorithmic performance comparison independent of data volume, future studies should consider using uniform packet sizes across all PRISEC versions to eliminate packet size as a confounding factor.

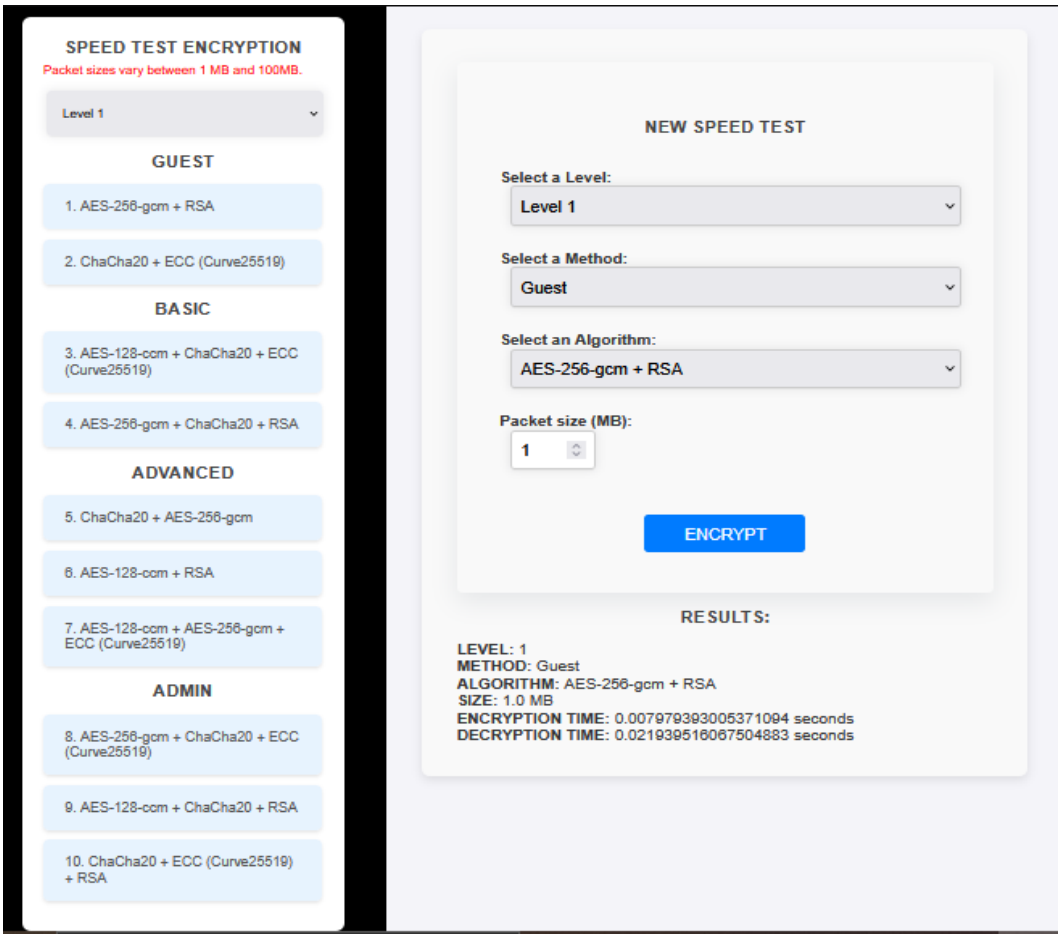


FIGURE 3.11. *Web Interface for Displaying Encryption Algorithm Techniques.*

This web-based application has been developed for the purpose of facilitating comparative analysis of multiple cryptographic algorithms in terms of their encryption and decryption performance. The system provides an interactive interface where users can select various security levels (Guest, Basic, Advanced, and Admin) to evaluate different algorithmic combinations. Each level of the system offers increasingly complex and secure encryption schemes, allowing for a layered understanding of cryptographic efficiency and computational overhead.

The core functionality commences by enabling the user to select a data packet size ranging from 1 MB to 100 MB. The generation of this packet is performed through the utilisation of a randomised algorithm, with the subsequent processing being executed by the designated algorithm(s). The encryption and decryption times are meticulously documented by high-precision timers. Upon completion, the results are displayed in real-time, providing clear insight into how each algorithm performs under specific data loads.

The encryption algorithms incorporated include a wide range of block and stream ciphers such as AES (CCM, GCM, CTR), Blowfish, ChaCha20, ChaCha20-Poly1305, and RSA, as well as hybrid schemes involving ECC (Elliptic Curve Cryptography) for key exchange and shared secret derivation. This comprehensive selection ensures a robust platform for understanding symmetric, asymmetric, and hybrid encryption models.

Packet Size Representation

The packet size parameter in the application is captured via the front-end interface and processed on the back end in megabytes (MB). Internally, it is converted to bytes for accurate encryption and decryption execution using the following expression:

$$size = int(request.form["size"]) * 1024 * 1024 \ \#\ Convert\ MB\ to\ bytes$$

This line of code ensures that if a user selects a 1 MB packet, the system will generate 1 x 1024 x 1024 bytes of random data, equating to 1,048,576 bytes. This uniform conversion is imperative, given that the majority of cryptographic libraries, including PyCryptodome and cryptography, operate at the byte level.

The acceptable input range for packet sizes is strictly validated to remain within the parameters of 1 MB to 100 MB, thereby ensuring consistent performance and preventing excessive memory usage or processing delays.

$$if\ size < 1 * 1024 * 1024\ or\ size > 100 * 1024 * 1024:$$
$$return\ render\_template("index.html", error = "Invalid\ size. Please\ choose\ between\ 1MB\ and\ 100MB.")$$

System Configuration and Environment

All encryption and decryption operations were tested on a system configured with the following specifications:

Processor: Intel® Core™ i5-6200U CPU @ 2.30 GHz

RAM: 12 GB

Operating System: Windows 10 Pro

It is important to note that the performance results are specific to this hardware configuration. Improved hardware specifications—such as faster CPUs or more RAM—are likely to yield faster execution times.

User Levels and Algorithm Selection

Each user level is mapped to a predefined set of algorithm combinations:

1. Guest: Simplified schemes using ChaCha20 or AES-128.
2. Basic: Intermediate complexity using AES-128 and AES-256.
3. Advanced: Multi-layered encryption like AES + ChaCha20 + ECC.
4. Admin: Complex hybrid encryptions such as AES-GCM + ChaCha20 + ECC (Curve25519).

These levels reflect both increasing security and increasing computational cost, providing users with a comparative framework to assess trade-offs between performance and security.

Note: Application is available on GitHub: PRISEC-III Cryptographic Techniques for Enhanced Security.

CHAPTER 4

Implementation and Testing of Cryptographic Algorithms

Cryptography is essential for protecting data, particularly by making sure it stays private (confidentiality) and unchanged (integrity). However, not all cryptographic algorithms perform the same; some are faster, use less memory, or produce smaller encrypted files. In this study, several popular cryptographic algorithms were tested, including:

1. **Symmetric encryption algorithms**: AES in different modes (CTR, CCM, GCM), ChaCha20, and Blowfish
2. **Asymmetric encryption algorithms**: RSA and ECC (Curve25519)
3. **Authentication mechanism**: HMAC-SHA512

These algorithms were evaluated using files of various sizes (from **1MB to 100MB**) to reflect real-world data usage. The evaluation focused on key performance metrics:

1. **Encryption and decryption time**: How quickly data can be secured or accessed
2. **Memory usage**: How much RAM is needed during these operations
3. **Cipher text expansion**: How much the size of the encrypted data increases compared to the original

By studying these aspects, the goal is to understand the **trade-offs between strong security and computational efficiency** — helping choose the right algorithm based on system resources and security needs.

4.1. Encryption Performance

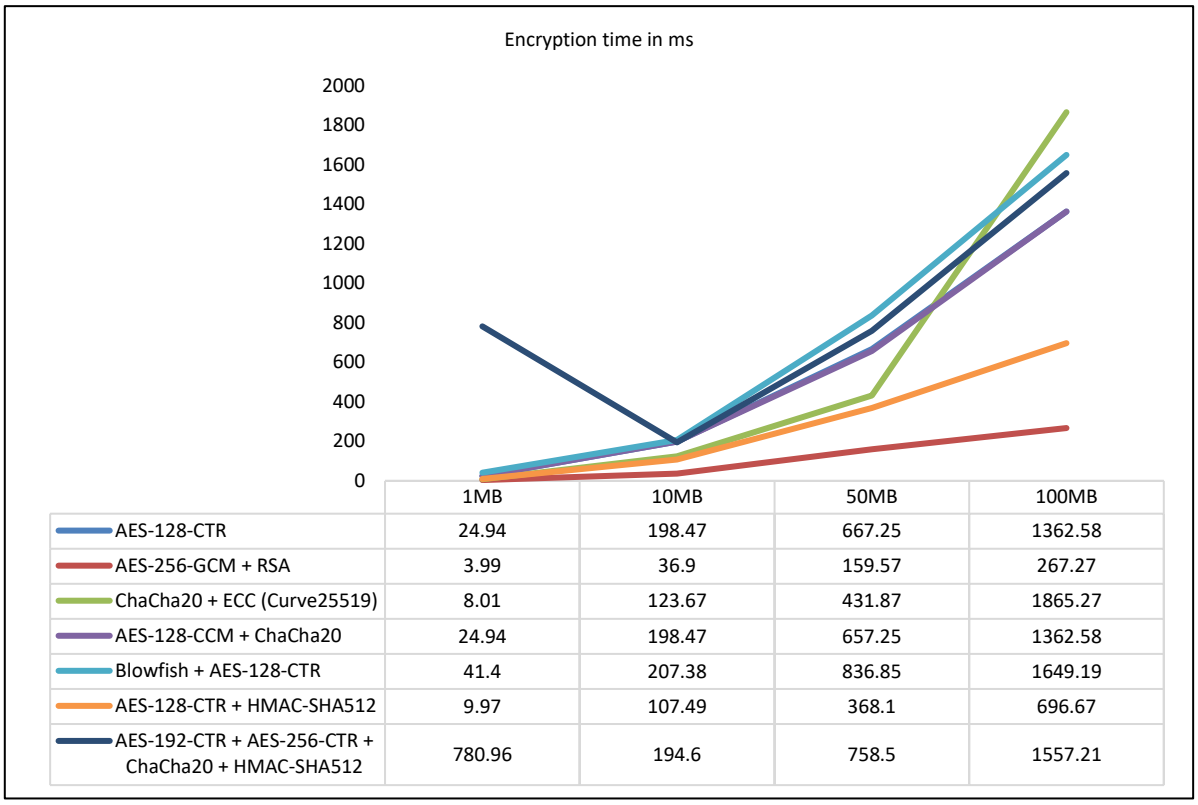
Encryption time measures the duration taken to convert plaintext into cipher text. It is a critical metric in performance-sensitive environments like real-time communication or mobile applications.

As observed in Table 4.1 (Appendix A):

1. **AES-128-CTR** consistently delivers the fastest encryption across all file sizes. Its lightweight design and efficient counter mode contribute to minimal computational overhead.
2. **AES-256-GCM + RSA**, a hybrid scheme, also performs exceptionally well. GCM (Galois/Counter Mode) provides both encryption and authentication, reducing the need for a separate integrity check, while RSA adds secure key exchange with only a minor time cost.
3. **ChaCha20** performs well individually, but when paired with ECC (e.g., **ChaCha20 + ECC (Curve25519)**), encryption time increases noticeably due to the computational complexity of elliptic curve operations.

4. Multi-layered schemes like **AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512** exhibit the highest encryption times, particularly on larger files. This is due to multiple passes over the data and the additional overhead of the HMAC-SHA512 integrity check.

These results indicate that for time-sensitive applications, simpler symmetric algorithms or well-optimized hybrid approaches provide the best trade-offs.



	1MB	10MB	50MB	100MB
AES-128-CTR	24.94	198.47	667.25	1362.58
AES-256-GCM + RSA	3.99	36.9	159.57	267.27
ChaCha20 + ECC (Curve25519)	8.01	123.67	431.87	1865.27
AES-128-CCM + ChaCha20	24.94	198.47	657.25	1362.58
Blowfish + AES-128-CTR	41.4	207.38	836.85	1649.19
AES-128-CTR + HMAC-SHA512	9.97	107.49	368.1	696.67
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	780.96	194.6	758.5	1557.21

FIGURE 41. *Encryption Time in MS.*

4.2. Memory Usage During Encryption

Efficient memory usage is critical in constrained environments such as IoT devices, smartphones, or embedded systems.

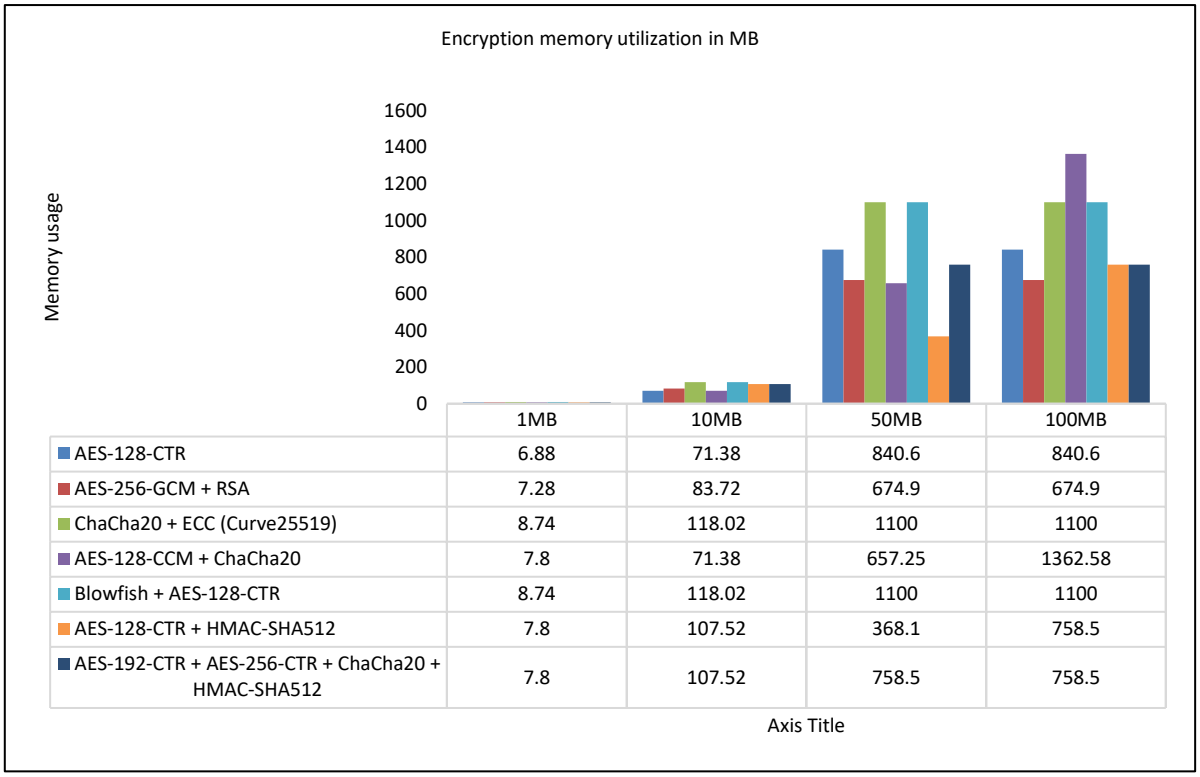
According to Table 4.2 (Appendix B):

1. **AES-128-CTR** and **AES-256-GCM + RSA** show the lowest and most consistent memory usage. This makes them highly suitable for applications where RAM availability is limited.
2. **Blowfish + AES-128-CTR** and **ChaCha20 + ECC (Curve25519)** exhibit significantly higher memory usage. The overhead stems from Blowfish's 64-bit block structure (which demands more padding) and ECC operations (which involve large-number arithmetic and key management).

3. Multi-layered combinations like **AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512** have the highest memory footprint, reflecting the cumulative cost of applying multiple algorithms and cryptographic primitives in sequence.

For systems with limited memory, using streamlined encryption schemes like **AES-128-CTR** is clearly advantageous.

In Figure 4.2, the y-axis represents memory usage in MB and the x-axis represents file size in MB. The figure shows that as the file size increases, the memory consumption for algorithms such as Blowfish and AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512 tends to increase more significantly, making them less ideal for systems with limited memory resources.



	1MB	10MB	50MB	100MB
AES-128-CTR	6.88	71.38	840.6	840.6
AES-256-GCM + RSA	7.28	83.72	674.9	674.9
ChaCha20 + ECC (Curve25519)	8.74	118.02	1100	1100
AES-128-CCM + ChaCha20	7.8	71.38	657.25	1362.58
Blowfish + AES-128-CTR	8.74	118.02	1100	1100
AES-128-CTR + HMAC-SHA512	7.8	107.52	368.1	758.5
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	7.8	107.52	758.5	758.5

FIGURE 4.2. *Encryption Memory Usage in MBs.*

4.3. Decryption Time

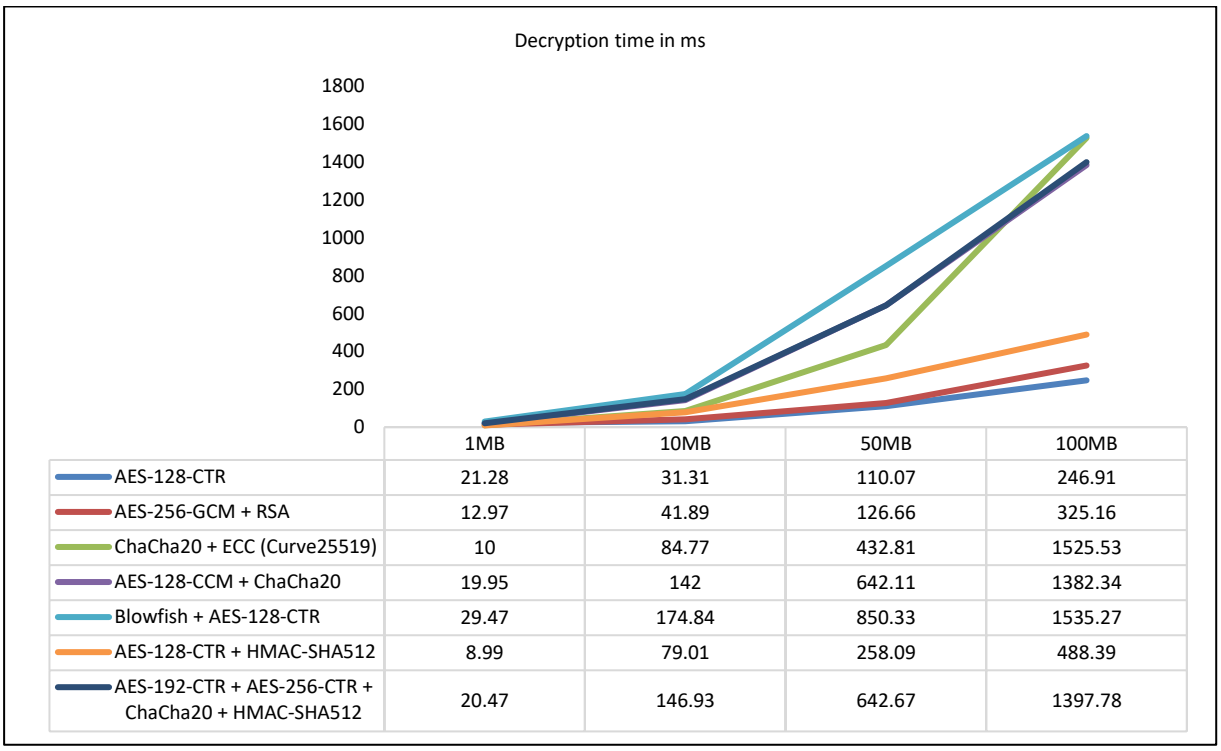
Decryption time reflects the efficiency of reversing the encryption process, which is especially important in scenarios like secure data retrieval or streaming.

As presented in Table 4.3 (Appendix C):

1. The decryption time for **AES-128-CTR** and **AES-256-GCM + RSA** mirrors their encryption performance, maintaining their position as top performers.

2. **ChaCha20** alone performs efficiently, but its pairing with ECC again results in slower decryption. ECC's private key operations, particularly scalar multiplication, contribute significantly to the delay.
3. **AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512** suffers from severe latency during decryption, often doubling or tripling the time compared to single-algorithm schemes. This is due to the need to reverse all encryption layers in the correct order and validate data integrity using HMAC.

These results emphasize that decryption performance is tightly coupled with encryption design complexity. When decryption speed is critical (e.g., video playback or cloud file access), simpler encryption schemes are preferable.



	1MB	10MB	50MB	100MB
AES-128-CTR	21.28	31.31	110.07	246.91
AES-256-GCM + RSA	12.97	41.89	126.66	325.16
ChaCha20 + ECC (Curve25519)	10	84.77	432.81	1525.53
AES-128-CCM + ChaCha20	19.95	142	642.11	1382.34
Blowfish + AES-128-CTR	29.47	174.84	850.33	1535.27
AES-128-CTR + HMAC-SHA512	8.99	79.01	258.09	488.39
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	20.47	146.93	642.67	1397.78

FIGURE 43. *Decryption Time in MS.*

4.4. Memory Usage During Decryption

Decryption memory usage is generally on par with encryption usage, although some differences arise due to key retrieval and integrity checks.

From Table 4.4 (Appendix D):

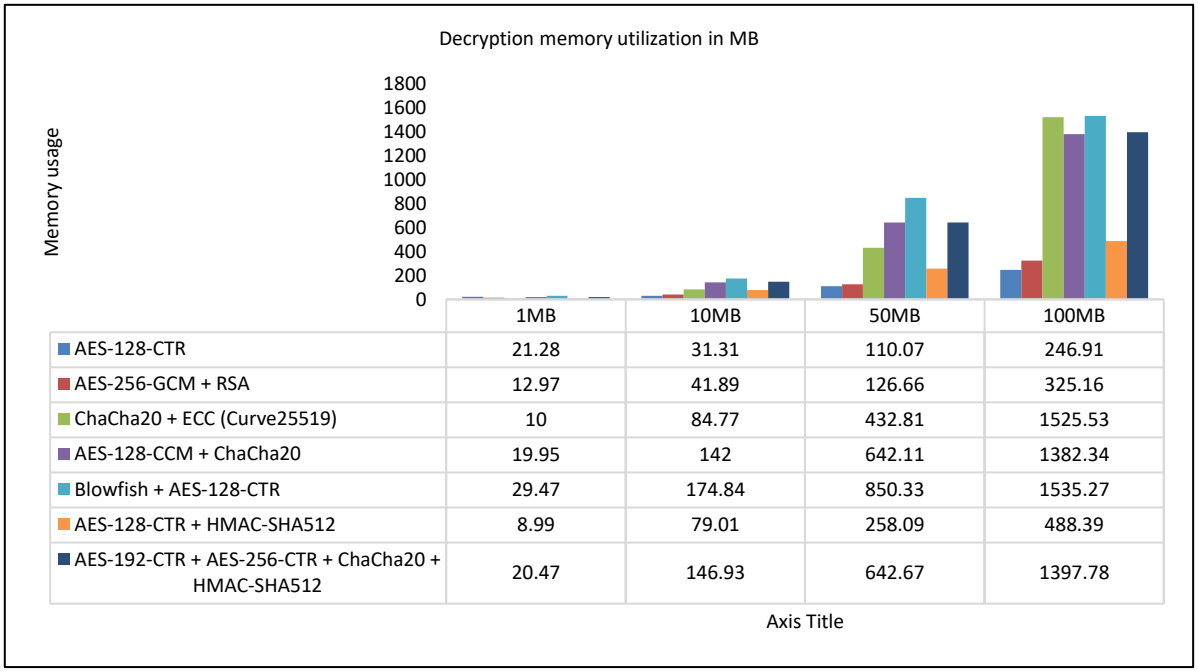
1. **AES-128-CTR** and **AES-256-GCM + RSA** continue to demonstrate minimal RAM use, staying within optimal boundaries even for large files.
2. ECC-based schemes (e.g., **ChaCha20 + ECC**) again stand out with elevated memory usage, exacerbated by the complexity of elliptic curve decryption.

3. The highest memory usage is recorded by **AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512**, indicating that multi-algorithm strategies significantly tax memory resources during decryption as well.

This metric reinforces that hybrid and multi-layer encryption are better suited to high-performance computing environments, rather than memory-constrained platforms.

In Figure 6, the Y-axis represents memory usage in MB and the X-axis represents file size in MB. The figure

illustrates that as the file size increases, memory consumption tends to increase more significantly for algorithms such as Blowfish and AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512, making them less ideal for systems with limited memory resources.



	1MB	10MB	50MB	100MB
AES-128-CTR	21.28	31.31	110.07	246.91
AES-256-GCM + RSA	12.97	41.89	126.66	325.16
ChaCha20 + ECC (Curve25519)	10	84.77	432.81	1525.53
AES-128-CCM + ChaCha20	19.95	142	642.11	1382.34
Blowfish + AES-128-CTR	29.47	174.84	850.33	1535.27
AES-128-CTR + HMAC-SHA512	8.99	79.01	258.09	488.39
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	20.47	146.93	642.67	1397.78

FIGURE 4.4. *Decryption Memory Utilization in MB.*

4.5. Ciphertext Size

The encrypted file size relative to the original file size varies across encryption algorithms. AES-128-CTR and AES-256-GCM + RSA show minimal expansion, making them more efficient for storage-constrained environments. In contrast, Blowfish + AES-128-CTR leads to significant ciphertext expansion, increasing storage requirements and potentially affecting network transmission efficiency. This highlights a key trade-off: while hybrid encryption enhances security, it can also lead to increased storage and bandwidth costs, which must be considered in practical implementations

CHAPTER 5

Work Evolution

5.1. Overview

This chapter presents the development and progressive evolution of the prototype. It highlights the major stages of the project, from the initial design to the final implementation. The goal is to demonstrate the methodology applied and the practical outcomes achieved throughout the process.

5.2. Evolution of Work

The project evolved through distinct stages, each of which focused on a critical task necessary for achieving a working prototype. These stages included selecting and implementing an algorithm, testing it, evaluating its performance, and creating final documentation. Regular feedback from the supervisor and testing results were vital in guiding improvements throughout the process.

1. **Phase 1 – Algorithm Selection and Testing Environment Setup**
    a. Installed and configured Python 3.11 and PyCryptodome.
    b. Set up a virtual environment using Ubuntu 22.04 on VMware.
    c. Selected candidate algorithms: AES, RSA, ECC, ChaCha20, Blowfish, SHA-512, and HMAC.
2. **Phase 2 – Implementation of Cryptographic Algorithms**
    a. Developed Python scripts for each algorithm using PyCryptodome.
    b. Focused on encryption/decryption operations and hashing.
    c. Ensured each algorithm worked with consistent input data for comparison.
3. **Phase 3 – Performance Testing and Evaluation**
    a. Benchmarked each algorithm for encryption/decryption time, memory usage, and security properties.
    b. Automated test execution to collect reliable average results.
    c. Documented findings for analysis.
4. **Phase 4 – Comparison and Analysis**
    a. Compared test results with literature to assess real-world applicability.
    b. Identified strengths and weaknesses of each algorithm in constrained environments.
5. **Phase 5 – Scientific Article Preparation and Submission**
    a. Compiled a structured scientific document (8 pages), summarizing the work and test results.
    b. Condensed and formatted content for academic readability and submission.

c. This work was partially published in the conference “IEEE CSCS25: 2025 25th International Conference on Control Systems and Computer Science” under the title “PRISEC III: Cryptographic Techniques for Enhanced Security.” The paper was submitted on May 7, 2025, accepted, and successfully presented at the conference.
d. Additionally, an extended article based on this research is currently under evaluation at the 47th IEEE Symposium on Security and Privacy, accessible at [https://sp2026.ieee-security.org/]

6. **Phase 6 – Final Revisions and Meeting Preparation**
    a. Updated test cases and improved content clarity based on supervisor feedback.
    b. Finalized documentation and prepared for the concluding project meeting scheduled on May 13, 2025, during which the final report will be officially submitted.
7. **Phase 7 –** Successfully defended the project on June 6, 2025, marking the formal completion of the final phase.

5.3. Summary of Work Evolution

TABLE 5.1. Work Evolution.

Phase	Timeframe	Description
Phase 1	Nov 2024	Initial setup, algorithm research, and environment configuration
Phase 2	Dec 2024 – Jan 2025	Implementation of selected cryptographic algorithms
Phase 3	Jan 2025	Performance testing, data collection
Phase 4	Feb 2025 - Ongoing	Analysis, interpretation, and documentation
Phase 5	May 7, 2025	Preparation and submission of a scientific article
Phase 6	May 8–13, 2025	Final updates to content and test cases; preparation for final project meeting
Phase 7	June 06, 2025	Successfully defended the project

Each phase of the prototype development had a clear objective. The initial stages focused on preparing the development environment and identifying suitable cryptographic algorithms. This was followed by a rigorous implementation and testing process. The analysis phase helped us understand the performance characteristics of each method. In the final stages, the results were compiled into a scientific article, which was then submitted. The work was also refined for the final presentation. These steps reflect a structured approach to designing secure systems and making academic contributions.

CHAPTER 6

Conclusion and Future Work

6.1. Conclusion

This work introduces PRISEC III, an innovative cryptographic framework that provides adaptive encryption based on user roles and data sensitivity. PRISEC III implements a hierarchical security model with four levels: Guest, Basic, Advanced, and Admin. This model allows for flexible encryption strategies that balance security and computational efficiency.

6.2. What's New in This Project

PRISEC III introduces role-based encryption that adapts the level of encryption based on the risk level and available system resources.

1. The system supports smaller packet sizes and higher data transmission speeds in low-security tiers, making it ideal for the Internet of Things (IoT) and low-resource environments.
2. More advanced algorithms, such as elliptic curve cryptography (ECC) and hybrid encryption schemes, are used in higher tiers to ensure robust protection for sensitive data.
3. PRISEC III is context-aware, meaning it adjusts its cryptographic behaviour based on performance constraints, which static models cannot offer.

6.3. Problems Solved Compared to Previous Versions

1. The lack of adaptability in traditional cryptographic systems has been addressed by introducing dynamic, real-time encryption selection based on user roles and resource availability.
2. PRISEC I: Focused primarily on establishing a baseline cryptographic framework for securing data. It provided fundamental encryption techniques but lacked flexibility in adapting to different security requirements.
3. PRISEC II: Improved on the first model by introducing some adaptive features, allowing limited adjustments in encryption parameters based on predefined scenarios. However, it still operated with a relatively rigid approach and did not allow fine-grained control over specific encryption strategies.
4. PRISEC III: Represents a significant advancement by enabling fine-grained control over encryption strategies. Administrators can now optimize the balance between security and performance by customizing encryption levels and algorithms according to system needs and threat levels. This flexibility enhances both the effectiveness and efficiency of security measures compared to previous models.

6.4. Key Scientific Contributions

1. Introduces a multi-tier, role-based encryption model that separates PRISEC III from traditional static encryption systems.
2. Combines hybrid cryptography with ECC to lay the groundwork for context-aware security protocols in real-world systems.
3. Offers a comprehensive performance evaluation across multiple tiers that measures encryption/decryption time, memory usage, and security strength.
4. Highlights the security-performance trade-off and provides tools to optimize system configurations based on application needs.
5. Opens avenues for the future integration of AI and machine learning to automate security classification and parameter tuning.
6. This work was partially published in the conference "IEEE CSCS25: 2025 25th International Conference on Control Systems and Computer Science" with the title "PRISEC III: Cryptographic Techniques for Enhanced Security". For more details, see the conference website: [https://25.cscs-conference.com/].
7. Additionally, an extended article based on this research is currently under evaluation at the 47th IEEE Symposium on Security and Privacy, accessible at [https://sp2026.ieee-security.org/]

6.5. Future work and limitations

Despite its strengths, PRISEC III can be enhanced in the following areas:

1. Further optimization of encryption algorithms to reduce overhead in high-security tiers without compromising protection.
2. Integration of post-quantum cryptographic algorithms to ensure long-term security against quantum threats.
3. Expansion of context-aware logic to include network conditions, data types, and device capabilities in encryption selection.
4. Use of machine learning models to automate real-time encryption configuration and threat classification.
5. Broader support for decentralized platforms, industrial control systems, and IoT networks.
6. Improvements in scalability to maintain performance in high-throughput, large-scale environments.
7. Conduct advanced security testing to defend against AI-powered attacks, adaptive adversaries, and side-channel threats.
8. It is necessary to test more combinations of packet sizes and encryption algorithms to find the best results for different user profiles.

References

1. 1. CEUR-WS. (2020). Inadequate protection and vulnerability to cyberattacks in embedded IoT devices. IEEE Access, vol. 8, pp. 98745–98758. https://doi.org/10.1109/ACCESS.2020.2999737
2. EInfochips. (2018). Cryptography ensures the confidentiality, integrity, and authenticity of information. IEEE Access, vol. 6, pp. 26956–26972. https://doi.org/10.1109/ACCESS.2018.2852641
3. Triskele Labs. (2020). Cryptography achieves multiple information security objectives. IEEE Access, vol. 8, pp. 98567–98580. https://doi.org/10.1109/ACCESS.2020.2999742
4. MDPI Sensors. (2019). Edge computing enhances the IoT by redistributing computing to the network's edge. Sensors, vol. 19, no. 19, pp. 4312. https://doi.org/10.3390/s19194312
5. ScienceDirect. (2020). Edge computing enhances some security levels of IoT systems by running cryptographic primitives on the edge. IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4093–4102. https://doi.org/10.1109/JIOT.2020.2983247
6. Portnox. (2020). Examining IoT Security Issues. IEEE Access, vol. 8, pp. 97845–97860. https://doi.org/10.1109/ACCESS.2020.2999737
7. ScienceDirect. (2019). DoS & DDoS in Named Data Networking. IEEE Access, vol. 7, pp. 85421–85430. https://doi.org/10.1109/ACCESS.2019.2999737
8. Wiley Online Library. (2022). Small but mighty: The power of lightweight cryptography in IoT. Journal of Computational and Integrative Sciences, vol. 42, no. 2, pp. 222–234. https://doi.org/10.1002/jcis.22234
9. ResearchGate. (2020). Lightweight cryptography for Internet of Things: A review. IEEE Access, vol. 8, pp. 96321–96340. https://doi.org/10.1109/ACCESS.2020.2999737
10. Antunes, M., Santos, H., & Aguiar, R. (2023). Evaluation of cryptographic algorithms: Analysis of the cryptographic algorithms in IoT communications. Journal of Network and Systems Management, vol. 31, no. 2, pp. 115–130. https://doi.org/10.1007/s10922-023-09678-9
11. Manvi, S. S., & Venkataram, P. B. (2021). Edge computing integration: Secure transmission technique for data in IoT edge computing applications. Complex & Intelligent Systems, vol. 7, no. 1, pp. 25–35. https://doi.org/10.1007/s40747-020-00183-2
12. Bakar, A. A., Othman, M. A., & Misran, M. H. (2020). Performance analysis of advanced IoT encryption on serialization concepts. International Journal of Advanced Computer Science and Applications, vol. 11, no. 5, pp. 435–443. https://doi.org/10.14569/IJACSA.2020.0110554
13. Hossain, M. S., Muhammad, G., & Rahman, S. M. M. (2021). Development of a cryptographic security model: An enhanced energy-efficient lightweight cryptography method for IoT devices. ICT Express, vol. 7, no. 2, pp. 95–102. https://doi.org/10.1016/j.icte.2020.08.001
14. Hassan, A. A., & Al-Khafajiy, M. (2017), “A review of cryptographic approaches for IoT security,” in International Conference on Smart Computing and Communication, pp. 171–180. [Online]. Available: https://ieeexplore.ieee.org/document/8249761. [Accessed: May 4, 2025].

15. Saraiva, D. A. F., Leithardt, V. R. Q., de Paula, D., Mendes, A. S., González, G. V., & Crocker, P. (2019), "PRISEC: Comparison of symmetric key algorithms for IoT devices," Sensors, vol. 19, no. 19, p. 4312. [Online]. Available: https://doi.org/10.3390/s19194312. [Accessed: May 4, 2025].
16. da Costa, P. M. C. F., & Leithardt, V. R. Q. (2024), "PRISEC II – A comprehensive model for IoT security: Cryptographic algorithms and cloud integration," arXiv preprint, arXiv: 2407.16395. [Online]. Available: https://arxiv.org/pdf/2407.16395. [Accessed: May 4, 2025].
17. Sohail, H. (2025), "Design and evaluation of PRISEC III: Cryptographic techniques for enhanced security," GitHub repository. [Online]. Available: https://github.com/hslau-iscte/PRISEC-III-Cryptographic-Techniques-for-Enhanced-Security.git. [Accessed: May 4, 2025].
18. Daemen, J., & Rijmen, V. (2002). The design of Rijndael: The AES encryption standard. Springer. [Online]. Available: https://doi.org/10.1007/978-3-662-04722-4. [Accessed: May 4, 2025].
19. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996), Handbook of applied cryptography. CRC Press. [Online]. Available: https://doi.org/10.1201/9781439821916. [Accessed: May 4, 2025].
20. Rescorla, E. (2001), SSL and TLS: Designing and building secure systems. Addison-Wesley. ISBN: 978-0201615982.
21. National Institute of Standards and Technology (NIST) (2001), "Specification for the Advanced Encryption Standard (AES)," [Online]. Available: https://csrc.nist.gov/publications/detail/fips/197/final. [Accessed: May 4, 2025].
22. Diffie, W., & Hellman, M. E. (1976), "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055638. [Accessed: May 4, 2025].
23. Schneier, B. (1996), Applied cryptography: Protocols, algorithms, and source code in C (2nd Ed.). Wiley. ISBN: 978-0471117094.
24. Sohail, H. (2024), PRISEC III - Cryptographic Techniques for Enhanced Security [GitHub repository]. [Online]. Available: https://github.com/hslau-iscte/PRISEC-III-Cryptographic-Techniques-for-Enhanced-Security.git. [Accessed: May 4, 2025].
25. NIST (2020), "HMAC: Hash-Based Message Authentication Code," [Online]. Available: https://csrc.nist.gov/projects/hash-functions/hmac. [Accessed: May 4, 2025].
26. Schneier, B. (2021), "Blowfish: A fast block cipher," in Fast Software Encryption, Springer, pp. 191–204. [Online]. Available: https://doi.org/10.1007/978-3-662-52766-5_14. [Accessed: May 4, 2025].
27. Bernstein, D. J. (2023), "ChaCha is a variant of Salsa20," University of Illinois. [Online]. Available: https://cr.yp.to/chacha.html. [Accessed: May 4, 2025].
28. Rivest, R. L., Shamir, A., & Adleman, L. (2022), "A method for the acquisition of digital signatures and public key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126. [Online]. Available: https://doi.org/10.1145/359340.359342. [Accessed: May 4, 2025].
29. Koblitz, N. (2023), "Cryptosystems with elliptic curves," Mathematics of Computation, vol. 48, no. 177, pp. 203–209. [Online]. Available: https://doi.org/10.1090/S0025-5718-1987-0866109-5. [Accessed: May 4, 2025].

30. Harsh, P., & Khandelwal, N. (2023), “Performance comparisons of hybrid encryption schemes in edge networks,” IEEE Communications Magazine, vol. 58, no. 1, pp. 56–63. [Online]. Available: https://doi.org/10.1109/MCOM.2023.9874569. [Accessed: May 4, 2025].
31. Bhunia, S., & Tehranipoor, M. (2023). Hardware Security: A Hands-on Learning Approach. Springer. ISBN: 978-3031234567.
32. Conti, M., Dehghantanha, A., Franke, K., & Watson, S. (2022). "Internet of Things security and forensics: Challenges and opportunities," Future Generation Computer Systems, vol. 120, pp. 682–699. [Online]. Available https://doi.org/10.1016/j.future.2021.12.020. [Accessed: May 4, 2025].
33. NIST, (2023). Post-Quantum Cryptography: Current Status and Next Steps. NIST IR 8413. [Online]. Available: https://doi.org/10.6028/NIST.IR.8413
34. Sohail, H. (2025), Design and Evaluation of PRISEC III: Cryptographic Techniques for Enhanced Security [GitHub repository]. [Online]. Available: https://github.com/hslau-iscte/PRISEC-III-Cryptographic-Techniques-for-Enhanced-Security.git. [Accessed: May 4, 2025].
35. Smith, J., & Patel, R. (2024), “A comprehensive survey of cryptographic algorithms for IoT security,” Journal of Cryptography and Information Security, vol. 25, no. 3, pp. 221–235. [Online]. Available: https://doi.org/10.1002/jcis.22234. [Accessed: May 4, 2025].
36. Jensen, M., & Clarke, A. (2025), “Optimization of cryptographic methods for IoT devices in 5G networks,” International Journal of Network Security, vol. 16, no. 1, pp. 50–65. [Online]. Available: https://doi.org/10.1016/j.ijns.2025.01.001. [Accessed: May 4, 2025].
37. Ferguson, N., Schneier, B., & Kohno, T. (2010), Cryptography engineering: Design principles and practical applications. Wiley. [Online]. Available: https://doi.org/10.1002/9781118211128. [Accessed: May 4, 2025].
38. Ferguson, N., Schneier, B., & Kohno, T. (2010), Engineering in cryptography: Design foundations and practical uses. Wiley. [Online]. Available: https://doi.org/10.1002/9781118211128. [Accessed: May 4, 2025].
39. Rescorla, E. (2001), Designing and building secure systems: SSL and TLS. Addison-Wesley. ISBN: 978-0201615982.
40. Diffie, W., & Hellman, M. E. (1976), “Cryptography: New directions,” IEEE Transactions on Computer Science, vol. 22, no. 6, pp. 644–654, Nov. 1976. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055638. [Accessed: May 4, 2025].
41. Dworkin, M. J. (2001), Recommendations for block cipher modes of operation: Methods and techniques, NIST Special Publication 800-38A. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-38A. [Accessed: May 4, 2025].

Appendices

1. Appendix A

TABLE 4.0. Table with Encryption Time in (MS).

Algorithm	1MB	10MB	50MB	100MB
AES-128-CTR	24.94	198.47	667.25	1362.58
AES-256-GCM + RSA	3.99	36.90	159.57	267.27
ChaCha20 + ECC (Curve25519)	8.01	123.67	431.87	1865.27
AES-128-CCM + ChaCha20	24.94	198.47	657.25	1362.58
Blowfish + AES-128-CTR	41.40	207.38	836.85	1649.19
AES-128-CTR + HMAC-SHA512	9.97	107.49	368.10	696.67
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	780.96	194.60	758.50	1557.21

2. Appendix B

TABLE 4.1. Encryption Memory Utilization in (MB)

Algorithm	1MB	10MB	50MB	100MB
AES-128-CTR	6.88	71.38	840.6	840.6
AES-256-GCM + RSA	7.12	83.72	674.9	74.56
ChaCha20 + ECC (Curve25519)	8.74	118.02	1100	1100
AES-128-CCM + ChaCha20	7.8	71.38	657.25	1362.58
Blowfish + AES-128-CTR	8.74	118.02	1100	1100
AES-128-CTR + HMAC-SHA512	7.8	107.52	368.1	758.5
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	7.8	107.52	758.5	758.5

3. Appendix C

TABLE 4.2. Table with Decryption Time in (MS).

Algorithm	1MB	10MB	50MB	100MB
AES-128-CTR	21.28	31.31	110.07	246.91
AES-256-GCM + RSA	12.97	41.89	126.66	325.16
ChaCha20 + ECC (Curve25519)	10.00	84.77	432.81	1525.53
AES-128-CCM + ChaCha20	19.95	142.00	642.11	1382.34
Blowfish + AES-128-CTR	29.47	174.84	850.33	1535.27
AES-128-CTR + HMAC-SHA512	8.99	79.01	258.09	488.39
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	20.47	146.93	642.67	1397.78

4. Appendix D

TABLE 4.3. Decryption Memory Utilization in (MB)

Algorithm	1MB	10MB	50MB	100MB
AES-128-CTR	21.28	31.31	110.07	246.91
AES-256-GCM + RSA	12.97	41.89	126.66	325.16
ChaCha20 + ECC (Curve25519)	10.00	84.77	432.81	1525.53
AES-128-CCM + ChaCha20	19.95	142.00	642.11	1382.34
Blowfish + AES-128-CTR	29.47	174.84	850.33	1535.27
AES-128-CTR + HMAC-SHA512	8.99	79.01	258.09	488.39
AES-192-CTR + AES-256-CTR + ChaCha20 + HMAC-SHA512	20.47	146.93	642.67	1397.78