

**Final Year Project**  
**Master Degree in Computer Engineering**

  
Use and application of Randomized Cryptographic Techniques for Enhanced Security in Edge Computing

Supervisor: Prof. Valderi Reis Quietinho Leithardt

Co-Supervisor: Prof. António Rui Trigo Ribeiro

Student: Humza Sohail (Nº109999)

**2025**

**Dedication**

I would like to extend my deepest gratitude to my Supervisor, **Professor Valderi Reis Quietinho Leithardt**, whose exceptional guidance, insight, and encouragement have been invaluable throughout this journey. His dedication to fostering learning and his ability to inspire have greatly contributed to the successful completion of this project.

My heartfelt thanks also go to my Co-Supervisor, **Professor António Rui Trigo Ribeiro**, for his thoughtful feedback, constructive suggestions, and unwavering support. His expertise and mentorship have been instrumental in refining the quality and direction of this work.

I am profoundly grateful to my family and friends, whose encouragement, patience, and belief in my abilities provided the strength I needed to overcome challenges and stay focused.

Lastly, I would like to thank my colleagues and everyone who contributed to this project in any way. Your valuable insights, collaboration, and support have been critical in shaping the outcomes of this endeavor.

### **Abstract**

This project explores the implementation of **randomized cryptographic techniques** to enhance data security and efficiency in interconnected systems, particularly in environments like the Internet of Things (IoT). The primary objective is to develop and implement a model that applies diverse cryptographic algorithms to secure data transmissions while maintaining efficiency. Given the computational constraints of IoT devices, this work focuses on selecting and employing distributed cryptographic techniques to achieve optimal performance. The proposed model integrates four distinct security levels, each leveraging varying cryptographic methods to provide enhanced data protection and scalability. Additionally, the use of edge computing ensures improved resource allocation and processing efficiency, creating a robust framework for secure and efficient data transmission.

**Keywords**  
Cryptographic Algorithms, Security, Distributed Algorithms, Data Protection, Algorithm Performance, Cryptographic Security, Cybersecurity, AI-Enhanced Encryption ,and Edge computing.

**Table of Content**

[Dedication ii](#_bookmark0)

[Abstract i](#_bookmark1)ii

[List of figures vi](#_bookmark3)

[List of Tables viii](#_bookmark4)

[List of Acronyms ix](#_bookmark5)

[List of software used](#_bookmark6) x

1. [Introduction 1](#_bookmark7)
   1. [Overview 2](#_bookmark8)
   2. [Motivation and Contents 2](#_bookmark9)
   3. [Objectives of the project 3](#_bookmark10)
   4. [Report Organization 3](#_bookmark11)
2. [State of the art 5](#_bookmark12)
   1. [Related works 6](#_2.1_Related_Works)
   2. [Comparison of Algorithms in Relation to Previous Versions of PRISEC](#_2.2_Comparison_of)……..7
   3. [Conclusion 10](#_2.3_Conclusion)
3. [Project description 13](#_Project_Description)
   1. [Introduction](#_3.1_Introduction) 14
   2. [Framework](#_3.2_Framework) 14
   3. [Discussion of problems and approaches](#_3.3_Discussion_of) 15
4. [Selection of cryptographic Algorithms and tools used](#_Selection_of_Cryptographic) 17
   1. [Selection of cryptographic algorithms](#_4.1_Selection_of) 18
      1. [AES](#_4.1.1_AES_(Advanced) 18
      2. [Blowfish](#_4.1.2_Blowfish) 18
      3. [Chacha20 and XChacha20 and Chacha20-Poly1305](#_4.1.3_ChaCha20_,XChaCha20) 18
      4. [ECC](#_4.1.4_ECC_(Elliptic) 19
      5. [HMAC-SHA-512](#_4.1.5_HMAC-SHA-512) 19
      6. [RSA](#_4.1.6_RSA(Rivest-Shamir-Adleman))………………………………………………………………………………………….19
   2. [Tools used for implementation of the client-server](#_4.2_Tools_Used) 19
      1. [Python Programming Language 19](#_4.2.1_Python_Programming)
      2. [Edge Computing 20](#_4.2.2_Local_Server)
5. [Implementation and testing of cryptographic algorithms 21](#_Implementation_and_testing)
   1. [Introduction 22](#_Implementation)
   2. [Implementation 22](#_Implementation)
      1. [Initial testing 22](#_Initial_testing)
      2. [Packet quantity 32](#_bookmark56)
      3. [Edge implementation 36](#_bookmark64)
   3. [Discussion of the results obtained 43](#_bookmark77)
6. [Conclusions and future work 47](#_bookmark78)
7. [References 49](#_bookmark79)

### **List of figures**

### **List of tables**

### List of Acronyms

|  |  |
| --- | --- |
| **AES CTR**  **CCM**  **ECC**  **HMAC HTML HTTP**  **IoT JSON** | Advanced Encryption Standard Counter  Elliptic-Curve Cryptography  Hash Message Authentication Code Hypertext Markup Language Hypertext Transfer Protocol  Internet of Things JavaScript Object Notation |

### List of software used

**Git**

**Microsoft Excel Microsoft Word Python**

**Visual Code Studio Windows remote Desktop**

# Chapter 1

## Introduction

This chapter explains the importance of cryptography in securing IoT communications, emphasizing its role in protecting sensitive data and ensuring reliable connections. It also describes the goals of this project and the motivations that inspired its development.

###### Overview

With the growing use of **edge computing** and big data in daily life and business operations, ensuring the security of data processed by IoT devices has become a top priority. Fields such as healthcare, smart grids, home automation, precision agriculture, and urban mobility are just a few examples where these technologies are widely applied. However, alongside the benefits of connectivity come serious challenges, particularly in protecting sensitive data, which is the main focus of this project.

**Cryptography** plays a critical role in safeguarding sensitive information. It involves creating and using coded algorithms to secure data, ensuring that only authorized users with the proper decryption tools can access it. In short, cryptography is essential to ensuring that data remains protected as it moves through interconnected systems.

**Edge computing**, in contrast to cloud computing, processes data closer to the source (such as on local servers or devices). This reduces latency and improves real-time performance by eliminating the need for long-distance data travel. In this project, **edge computing** will be used to enhance data processing efficiency, ensuring quick data analysis and encryption at the point of generation, thereby reducing delays and optimizing security for IoT devices.

This project focuses on integrating cryptographic algorithms directly within the edge computing framework to secure data. By processing and securing data on local servers, it reduces the risk of data breaches and ensures faster, more efficient encryption and decryption. Further details on the integration of edge computing with cryptographic algorithms will be discussed later in the report.

###### Motivation and Contents

IoT applications today are generally divided into four layers: the sensing layer, network layer, middleware layer, and application layer. Each of these layers relies on various technologies, which introduce a range of issues and security threats. The application layer, in particular, directly interacts with end users and provides services to them. This layer faces unique security challenges, such as data theft and DDoS attacks, which are less common in the other layers. Since IoT applications handle a significant amount of critical and private data, and there is constant data transfer, the data in motion becomes more vulnerable to attacks than data stored at rest. One effective method to secure IoT applications from such threats is data encryption [22].

This project aims to address these data security challenges by implementing cryptographic techniques. To achieve this, the project involves researching commonly used cryptographic algorithms, especially those applicable to scenarios and applications utilizing wireless internet. These algorithms will be tested based on factors such as packet quantity and size. The ultimate goal of the project is to develop and implement a robust cryptographic solution that enhances the security of IoT devices and protects sensitive data from unauthorized access or tampering. Various cryptographic algorithms will be explored throughout the project to determine the most effective solution. To provide further clarity, the specific objectives of this project will also be outlined.

###### Objectives of the project

The main goal of this project is to choose and study different cryptographic algorithms to find the ones that perform best in terms of how quickly they can encrypt and decrypt data, while also considering the size of the data packets. Next, these algorithms will be applied to a structured model, which will be explained later in the report, to measure the time it takes to encrypt and decrypt packets of different sizes. Once the most efficient model is identified, its performance will be tested by analyzing how it handles different numbers of packets being sent.

After this, the improved model will be used with edge computing services to evaluate how well it performs in these environments. Finally, the model will be adapted for real-world applications, using edge computing to ensure data remains secure and is processed efficiently.

###### Report Organization

This report is organized into six chapters, each covering key aspects of the project:

* **Introduction**: Introduces the project by outlining its motivation, objectives, and the structure of the report.
* **State of the Art**: Reviews current cryptographic techniques, focusing on their applications in edge computing, as well as their use in securing IoT devices and data transmissions.
* **Project Description**: Explains the framework of the project, the challenges related to data security in IoT and edge environments, and the approaches taken to address these challenges.
* **Selection of Cryptographic Algorithms and Tools**: Details the cryptographic algorithms chosen for this project and the tools used for their implementation and testing.
* **Implementation and Testing**: Describes the process of implementing the selected algorithms, conducting initial tests, and analyzing the results from the encryption and decryption tests.
* **Conclusions and Future Work**: Summarizes the findings of the project and outlines possible directions for future research to further enhance security and efficiency in IoT and edge computing.

# Chapter 2

## State of the art

This chapter reviews the current state of the art in cryptographic solutions, with a focus on their implementation in **real-time systems, edge computing environments, and IoT networks**. It also explores the challenges and impact of these algorithms on devices with limited computational power, particularly in the context of processing data locally on edge servers. This review provides a foundation for the project's proposed security model, which aims to enhance data protection for IoT applications using edge computing.

#### 2.1 Related Works

Security is an ever-growing concern in systems that handle sensitive data, especially with the increased reliance on distributed computing models like edge computing. Researchers have conducted extensive studies to understand attacks and defenses in such systems. For example, the study in [17] analyzed the effectiveness of security research in real-time systems by categorizing and evaluating various defense mechanisms. The research also focused on scheduler-based security techniques, which are among the most widely used. To measure the effectiveness of these mechanisms, a common metric called "attacker’s burden" was introduced, helping to quantify how difficult it would be for an attacker to breach the system.

Another area of interest is decentralized systems, where computational tasks are spread across multiple nodes without relying on a central authority. These systems face unique challenges because any node could potentially be compromised. The study in [9] explored existing security solutions for such environments and assessed the performance of cryptographic algorithms on devices with different resource capacities. It provided benchmarks for widely used cryptographic techniques and demonstrated the resource requirements needed to perform encryption and decryption effectively in constrained systems.

While decentralized systems pose their own challenges, security concerns span across many computing environments. For example, the research in [26] highlighted the increasing risks of data breaches and information leakage across industries. The study proposed a cryptographic framework tailored for Industry 5.0, focusing on securing data while maintaining compliance with regulatory standards. The framework emphasized real-time data protection, access control, and future-ready architectures to address emerging security needs.

With the increasing use of networked systems and distributed computing, the need for robust cryptographic models has become critical. The study in [1] proposed a cryptographic framework that integrates encryption, hash functions, and digital signatures to protect sensitive data, ensure integrity, and establish secure communication channels.

For environments like edge computing, where data processing occurs close to the source, cryptographic frameworks must be both secure and efficient. The work in [5] introduced a hybrid cryptographic framework combining Elliptic Curve Cryptography (ECC) and the Advanced Encryption Standard (AES). This framework was designed to meet the challenges of resource-constrained systems by optimizing memory usage and energy efficiency while maintaining robust security.

In addition, the study in [20] evaluated the performance of symmetric key algorithms, such as AES, Blowfish, ChaCha20, XChaCha20, and ChaCha20-Poly1305. Researchers analyzed encryption and decryption times, throughput, and energy consumption, identifying the most efficient algorithms for systems that require secure and fast data processing. These findings are particularly relevant to this project, as the algorithms under evaluation include:

* **AES**
* **Blowfish**
* **ChaCha20, XChaCha20, and ChaCha20-Poly1305**
* **Elliptic Curve Cryptography (ECC)**
* **HMAC-SHA-512**
* **RSA**

Building on these studies, this project focuses on evaluating and testing the performance of these cryptographic algorithms in an edge computing environment, implemented on a local server. The goal is to measure encryption and decryption times for various data packet sizes and assess the overall efficiency of the algorithms when applied in real-world scenarios involving secure data processing.

### 2.2 Comparison of Algorithms in Relation to Previous Versions of PRISEC

The cryptographic algorithms and their combinations used in this project were designed to enhance the security and performance of PRISEC, a framework that previously relied on simpler and less optimized cryptographic methods. Compared to earlier versions of PRISEC, the current implementation is:

#### 1. **More Robust**

The combinations now incorporate advanced cryptographic methods such as **AES-256-GCM**, **ChaCha20**, and **ECC (Curve25519)**. These algorithms are recognized for their resilience against brute-force attacks and other modern cryptographic threats. Additionally, by integrating multiple algorithms (e.g., AES with ChaCha20 and ECC), the framework ensures enhanced protection, even if one algorithm's security is compromised.

#### 2. **Faster Performance**

The adoption of lightweight algorithms such as **ChaCha20** significantly improves encryption and decryption speeds, particularly in resource-constrained environments like edge computing. Compared to previous versions, which relied on computationally intensive methods, the current framework minimizes latency and improves real-time processing capabilities. Performance tests revealed that:

* **ChaCha20** outperformed AES in terms of encryption speed for smaller data packets.
* **AES-256-GCM**, when combined with ECC, showed balanced performance in both speed and security for larger data sizes.

#### 3. **More Secure**

Earlier versions of PRISEC primarily utilized single-algorithm encryption methods, making them vulnerable to targeted attacks. The current version leverages multi-layer encryption, integrating **HMAC-SHA-512** for data integrity and **ECC** for secure key exchange. This layered approach not only strengthens security but also ensures compliance with modern cryptographic standards.

#### 4. **Supports More Data**

The current implementation handles larger datasets more efficiently due to the inclusion of **AES-128-CTR**, **AES-256-CTR**, and **Blowfish**, which are optimized for scalability. The use of block and stream ciphers in tandem ensures that the framework remains efficient across varying data sizes without compromising security.

#### 5. **Scalable and Adaptable**

The algorithm combinations are categorized into four security levels (**Guest**, **Basic**, **Advanced**, and **Admin**) to address varying security and performance requirements. Unlike earlier versions, which followed a one-size-fits-all approach, this modular framework allows for dynamic adaptability based on user roles and system needs.

### Key Improvements Over Previous Versions

| **Feature** | **Previous PRISEC** | **Current Implementation** |
| --- | --- | --- |
| **Algorithm Types** | Single-layer encryption (AES, RSA) | Multi-layer encryption (AES, ChaCha20, ECC, HMAC) |
| **Performance** | Slower encryption/decryption times | Faster speeds with optimized lightweight algorithms |
| **Security** | Vulnerable to certain attack types | Stronger resistance to modern cryptographic threats |
| **Data Size Support** | Limited scalability | Handles larger datasets efficiently |
| **User Roles** | Uniform security approach | Role-based modular security levels |

### **2.3 Conclusion**

In conclusion, the current implementation of PRISEC represents a significant leap forward compared to previous versions. By integrating advanced cryptographic algorithms like **AES-256-GCM**, **ChaCha20**, **ECC**, and **HMAC-SHA-512**, the framework offers **stronger security**, **improved performance**, and **greater scalability**. These improvements make the system more robust against modern cryptographic threats, faster in processing encryption and decryption operations, and more adaptable to various use cases, particularly in edge computing environments.

The enhancements in security, speed, and data handling make the current version of PRISEC a more efficient and future-proof solution for real-time data protection, especially in distributed edge environments. Furthermore, the **modular approach** with role-based security levels ensures that the system can cater to different user requirements, offering flexibility and adaptability for a range of applications.

In comparison to previous versions, the current PRISEC framework is not only more secure but also far more efficient, making it a suitable choice for modern computational environments and large-scale data processing. The result is a **robust, high-performance, and secure framework** that addresses the challenges of both security and efficiency in modern systems.

# Chapter 3

### **Project Description**

This chapter provides a detailed overview of the project, including its description, framework, and approaches to solving potential challenges related to the integration and implementation of cryptographic algorithms for data security in edge computing environments.

### **3.1 Introduction**

In today's interconnected world, ensuring the security and efficiency of data transmission has become a paramount concern. The primary objective of this project is to design and implement a structural model that utilizes advanced cryptographic algorithms to secure data within an edge computing environment.These algorithms have been selected based on academic research to identify the most effective options for real-world scenarios.

The project focuses on integrating these cryptographic algorithms into a local server setup, leveraging the computational resources available at the edge to enhance data security.This approach aims to provide robust protection against unauthorized access and data breaches, ensuring the integrity and confidentiality of sensitive information.

### **3.2 Framework**

The project framework is centered on safeguarding data integrity within the edge computing paradigm. Edge computing involves processing data closer to the data source, reducing latency and bandwidth usage.However, this distributed nature introduces unique security challenges, as edge devices are often exposed to various threats.

To address these challenges, the project aims to analyze and select the most suitable cryptographic algorithms that can perform efficiently in edge computing environments. Once selected, these algorithms will be integrated into a local server setup, ensuring scalability and resource optimization.By leveraging the computational power of the local server, the encryption processes can be offloaded from edge devices, reducing their computational burden and improving overall efficiency.

### **3.3 Discussion of Problems and Approaches**

Cryptography involves various computations and key management systems that transform plaintext into ciphertext through encryption.The time taken to encrypt and decrypt data can vary significantly based on several factors, including the size of the data packet and the complexity of the algorithms being used.Larger data packets tend to result in longer encryption and decryption times, which can be problematic in real-time applications.

In this project, one of the key challenges is to identify a set of cryptographic algorithms that provide optimal security without compromising speed, especially when handling large data packets.The project aims to select algorithms that enable the fastest transmission of data while maintaining the highest levels of security.

The proposed solution includes the development of structural models for data encryption.These models will use encapsulation of cryptographic algorithms and will be designed around four distinct security levels:

* **Guest**: This level offers basic encryption for less sensitive data.
* **Basic**: This level enhances security with moderate encryption protocols.
* **Advanced**: The encryption here is multi-layered, providing robust protection for sensitive data.
* **Admin**: This level employs the strongest encryption available, offering the highest security for critical data.

Each level has a different approach to the number of encryption layers used.For example, at the **Advanced** level, the data will undergo multiple rounds of encryption, which will improve security but also result in increased encryption and decryption times, demanding more computational power. Therefore, higher levels of encryption also come with trade-offs in performance.

In addition to algorithm encapsulation, this project will leverage the computational resources of a local server within the edge computing environment to enhance the processing efficiency of these cryptographic algorithms.By integrating the local server, the project will be able to offload some of the encryption and decryption tasks from edge devices, optimizing resource usage and improving the overall speed of the encryption process.This approach ensures that the edge devices can operate efficiently without being overwhelmed by computationally intensive tasks.

The final encryption models will be tested across different scenarios, ranging from smaller data packets to large datasets, ensuring that the system remains efficient under varying conditions while providing the necessary security.This comprehensive testing will help identify the optimal configurations for different use cases, balancing security and performance effectively.

# Chapter 4

## **Selection of Cryptographic Algorithms and Tools Used**

This chapter discusses the cryptographic algorithms selected for this project and the tools used in the implementation and testing phases. The algorithms were chosen based on their security, efficiency, and compatibility with edge computing environments. The tools used, including the Python programming language and local server configurations, are integral to the project’s success.

### **4.1 Selection of Cryptographic Algorithms**

This section provides an overview of the cryptographic algorithms selected for this project, explaining their roles in securing data transmitted within the edge computing framework.

### **4.1.1 AES (Advanced Encryption Standard)**

AES is one of the most widely used symmetric block ciphers globally, known for its security and efficiency. AES works by encrypting data in fixed-size blocks (128 bits) using a secret key, and it supports key sizes of 128, 192, or 256 bits. The AES algorithm operates by applying multiple rounds of processing, with the number of rounds varying depending on the key length (e.g., 10 rounds for 128-bit, 12 for 192-bit, and 14 for 256-bit).

* **AES-256** is the most secure variation, which is why it has been chosen for this project. It ensures robust protection against various cryptographic attacks, including brute force, making it suitable for securing sensitive data in edge computing environments.

In the context of this project, AES-256 is implemented using Python with the **PyCryptodome** library, which allows us to efficiently encrypt and decrypt data within the local server framework. The performance of AES-256 in terms of speed and security will be thoroughly tested during the project.

### **4.1.2 Blowfish**

Blowfish is a symmetric key block cipher that was designed as an alternative to the older DES encryption algorithm. It offers an excellent combination of speed and security and is a versatile option for use in applications where performance is a priority. Blowfish uses a block size of 64 bits and supports key sizes from 32 bits to 448 bits.

* In this project, Blowfish will be used for scenarios where lower encryption/decryption latency is necessary, as it tends to perform faster than AES in certain situations. Its compact design also makes it well-suited for the constrained environments found in edge computing.

Blowfish will be implemented as part of the encryption scheme, offering an alternative to AES where speed is crucial.

### **4.1.3 ChaCha20 ,XChaCha20 and** ChaCha20-Poly1305

ChaCha20 is a stream cipher widely adopted for its speed and security. It is designed to avoid the weaknesses found in older ciphers, such as RC4. ChaCha20 works by applying a series of transformations to the data, ensuring robust encryption with efficient computational demands. It has gained popularity in modern cryptographic applications due to its resistance to various attacks, such as those targeting traditional block ciphers like AES.

• XChaCha20 is an extended version of ChaCha20 that uses a longer nonce, providing additional security and preventing certain vulnerabilities. This algorithm is ideal for environments where fast, reliable encryption is needed without compromising security.

**ChaCha20-Poly1305** combines the ChaCha20 stream cipher with the Poly1305 message authentication code (MAC). ChaCha20 provides the encryption, while Poly1305 ensures that the data has not been altered during transmission, offering both confidentiality and data integrity.

ChaCha20, XChaCha20, and ChaCha20-Poly1305 will be employed in this project for their speed, especially in scenarios requiring fast, real-time encryption and decryption on edge devices. These algorithms are well-suited to local server environments where performance is critical.

### **4.1.4 ECC (Elliptic Curve Cryptography)**

ECC is a form of public-key cryptography based on the mathematics of elliptic curves. It is known for providing high security with relatively smaller key sizes compared to traditional public-key algorithms such as RSA. ECC is computationally efficient and provides stronger security per bit, making it an excellent choice for edge devices that require low-power encryption.

* In this project, **ECC (Curve25519)** will be utilized to provide secure key exchanges between devices in the edge computing network. The smaller key sizes make it particularly suitable for scenarios where resources are limited, such as on devices with low computational power.

ECC's ability to maintain high security with lower computational overhead makes it an essential tool for the encryption infrastructure in this project.

### **4.1.5 HMAC-SHA-512**

HMAC (Hashed Message Authentication Code) is a mechanism for verifying the integrity and authenticity of a message. **HMAC-SHA512** uses the **SHA-512** hash function in combination with a secret key to produce a unique code that can be used to check the integrity of data and confirm its origin.

* **SHA-512**: Part of the **SHA-2** family of cryptographic hash functions. It outputs a 512-bit hash.
* **HMAC**: Adds a layer of security to the hash function by combining the message with a secret key, ensuring that even if an attacker knows the hash function, they cannot generate the correct HMAC without the key.

### **4.1.6 RSA**(Rivest-Shamir-Adleman)

RSA is an asymmetric encryption algorithm that uses two keys: a public key and a private key. The public key is used for encryption, and the private key is used for decryption. RSA relies on the difficulty of factoring large prime numbers to ensure security. It’s often used in scenarios where secure key exchange is required, such as in SSL/TLS protocols.

* **Key Sizes**: RSA key sizes usually range from **1024 bits** to **4096 bits**, with larger keys providing stronger security but requiring more computational resources.
* **Operations**: In RSA, data is encrypted using the recipient's public key, and only the corresponding private key can decrypt it. It can also be used for digital signatures.

### **4.2 Tools Used for Implementation of the Client-Server**

This section outlines the tools and platforms used for implementing the cryptographic algorithms and deploying the server-side components of the project.

### **4.2.1 Python Programming Language**

Python was chosen for this project due to its simplicity, readability, and extensive libraries that support cryptographic operations. It is a versatile programming language widely used for security-focused applications and enables rapid development of the cryptographic algorithms.

* **PyCryptodome**: This Python library provides the tools necessary for implementing cryptographic algorithms, including AES, ChaCha20, and Blowfish. It supports both encryption and decryption operations, as well as secure random number generation for cryptographic purposes.
* The project will focus on **Python** for the server-side cryptographic operations, with direct implementation of the encryption and decryption functionalities via local server scripts. This approach is more lightweight and efficient, particularly in resource-constrained environments like edge computing.

The combination of Python, PyCryptodome allows for efficient cryptographic operations, easy integration with the local server, and the ability to create a user-friendly interface for encryption tasks.

### **4.2.2 Local Server (Virtualized Environment)**

A local server is configured within a virtualized environment to provide the necessary computational resources for handling the cryptographic tasks. This server will be used for testing and processing encrypted data as part of the project.

The local server is essential for running and testing the encryption algorithms in a controlled environment before deployment in larger-scale systems.

# Chapter 5

## Implementation and testing of cryptographic algorithms

This chapter, we will discuss the implementation and testing of the cryptographic algorithms used in this project. The main objective of the tests was to evaluate the performance of different encryption and decryption combinations in terms of time required for processing data packets, specifically focusing on the time taken for both encryption and decryption operations at different packet sizes

###### Implementation

###### Initial testing

In the initial phase of this project, various structured models were created and tested based on the algorithms previously explained. The objective was to identify the optimal structured model that achieved the best encryption and decryption times while also considering packet quantity and size. All tests during this phase were conducted on a single machine, ensuring that both the encryption and decryption processes occurred simultaneously.

For this phase, all tests were performed on an HP ProBook 640 G2 with the following specifications:

* **Processor:** Intel® Core™ i5-6200U CPU @ 2.30 GHz (2.40 GHz)
* **Installed RAM:** 12.0 GB (11.9 GB usable)
* **System Type:** 64-bit operating system, x64-based processor
* **Operating System:** Windows 10 Pro, Version 22H2, Build 19045.5371

The program, developed in Python, accepts two arguments from the user: access level and package size (in megabytes). After providing these inputs, the following process is executed:

1. A random package of the specified size is generated.
2. The package is encrypted using the specified access level's algorithm.
3. The encryption time is measured.
4. The package is decrypted using the same access level's algorithm.
5. The decryption time is measured.

### **Encryption Levels and Algorithm Structure**

In the initial phase of the project, four distinct encryption levels were designed to organize and test various cryptographic combinations. Each level—Guest, Basic, Advanced, and Admin—features a progression of algorithms in terms of complexity, security, and performance. These levels allow a systematic evaluation of how different cryptographic techniques perform across varying scenarios.

**Level 1** introduces foundational encryption combinations, starting with lightweight yet secure methods such as AES-256-GCM with RSA or ChaCha20 with ECC (Curve25519) for Guest access. As we move up to Basic and Advanced levels, the combinations grow more sophisticated, incorporating multiple algorithms such as AES-128-CCM, ChaCha20, and RSA. Admin access employs the most advanced setups, combining algorithms like AES-256-GCM with ChaCha20 and ECC to ensure the highest security and efficiency.

**Level 2** emphasizes CCM-based encryption modes, focusing on secure and authenticated encryption. Guest access starts with lightweight algorithms like AES-128-CCM combined with ChaCha20, while Basic and Advanced levels integrate AES-256-CCM and XChaCha20. Admin access employs combinations that feature the highest levels of computational strength, like AES-256-CCM and ChaCha20-Poly1305.

**Level 3** transitions to CTR-based encryption modes, offering high performance and flexibility. Guest access employs AES-128-CTR for simplicity, while Basic and Advanced levels introduce combinations of AES-CTR with ChaCha20, Blowfish, and ECC (Curve25519). Admin-level encryption integrates multi-layered approaches, including AES-256-CTR with Blowfish and ChaCha20, to provide comprehensive security for critical data.

**Level 4** focuses on hybrid encryption schemes, incorporating Blowfish, AES-CTR, and HMAC-SHA512. Guest access includes simple combinations like Blowfish with AES-128-CTR, while Basic and Advanced levels explore advanced integrations, such as AES-CTR with ChaCha20 and HMAC-SHA512. Admin-level encryption features highly robust configurations, combining multiple algorithms like AES-CTR, ChaCha20, HMAC-SHA512, and ECC for maximum security.

This structured progression allows for targeted testing and performance evaluation across use cases, from lightweight guest-level access to the highest security for administrative tasks.

### **Level 1 (Guest)**

### **1-AES-256-GCM + RSA**

AES-256-GCM combined with RSA is a hybrid encryption approach designed to ensure both high-speed data encryption and secure key exchange. AES-256-GCM is a symmetric encryption algorithm that uses a 256-bit key, offering robust security and integrating authentication with the Galois Counter Mode. This ensures data integrity alongside encryption. RSA, an asymmetric encryption algorithm, is primarily used for secure key exchange, relying on two keys: a public key for encryption and a private key for decryption. While RSA adds strong security guarantees, its computational complexity increases processing time, especially as data sizes grow.

In this experiment, the encryption and decryption times of AES-256-GCM with RSA were measured for different packet sizes ranging from 1 MB to 100 MB. The results show that encryption and decryption times increase linearly with packet size, highlighting the impact of RSA's computational overhead on performance. AES-256-GCM + RSA demonstrates strong security characteristics but is less efficient for large-scale or high-performance applications.

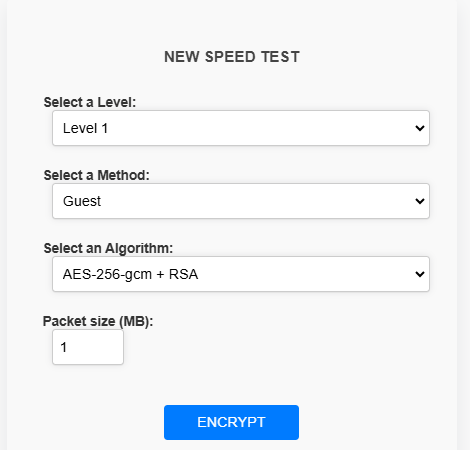


Figure 5.1 – Initial structured model in the Guest level

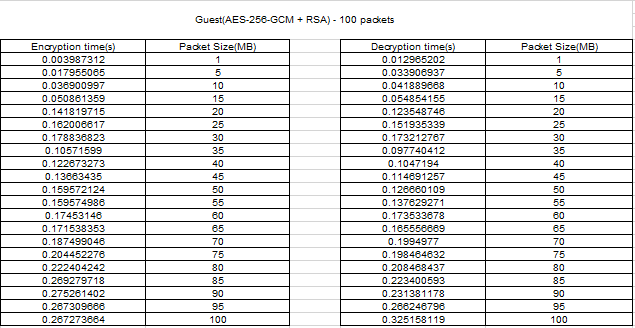


Table 5.1 – Values obtained for encryption and decryption in Guest level for the initial model.

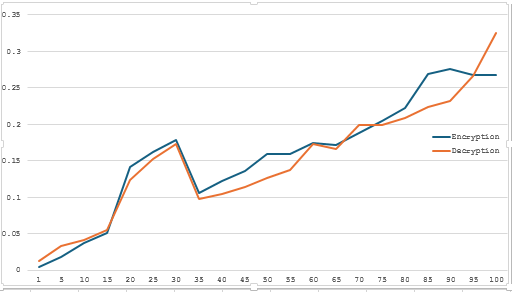


Figure 5.2 – Results of the encryption time in the Guest level.

### **2-ChaCha20 + ECC (Curve25519)**

ChaCha20 combined with ECC (Curve25519) is an efficient and lightweight encryption method tailored for performance-critical environments. ChaCha20 is a stream cipher that operates with high speed and low computational complexity, using a 256-bit key for strong encryption. Paired with ECC, specifically the Curve25519 curve, this combination ensures secure and efficient key exchange through elliptic curve cryptography. ECC reduces computational overhead by using smaller key sizes compared to RSA while maintaining equivalent security levels, making it an ideal choice for modern applications.

In this experiment, encryption and decryption times were evaluated for ChaCha20 + ECC across packet sizes ranging from 1 MB to 100 MB. The results demonstrate that this combination provides consistently faster encryption and decryption times compared to AES-256-GCM + RSA, particularly for smaller packet sizes. Even for larger packets, ChaCha20 remains efficient, highlighting its suitability for systems with limited computational resources.

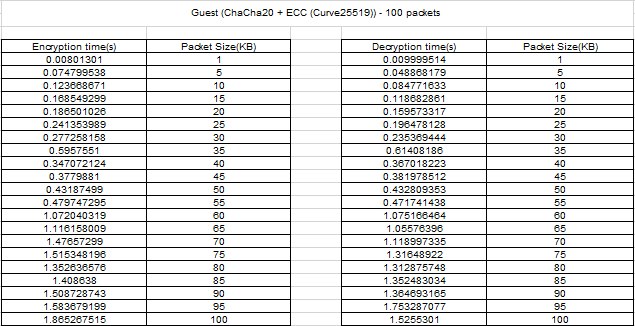


Table 5.2 – Values obtained for encryption and decryption in Guest level for the initial model

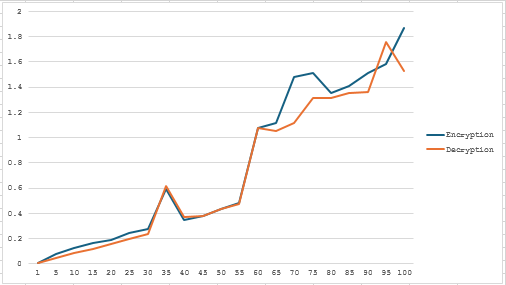


Figure 5.3 – Results of the encryption time in the Basic level.

### **Comparison and Conclusion**

When comparing the two algorithms, it is evident that ChaCha20 + ECC (Curve25519) outperforms AES-256-GCM + RSA in terms of speed and efficiency. The lightweight nature of ChaCha20 and the streamlined key exchange of ECC make it a better choice for scenarios where performance is critical, especially for smaller packet sizes or resource-constrained environments. Conversely, AES-256-GCM + RSA, while slower, provides higher cryptographic strength and robust security features, making it more suitable for applications requiring stringent security measures.

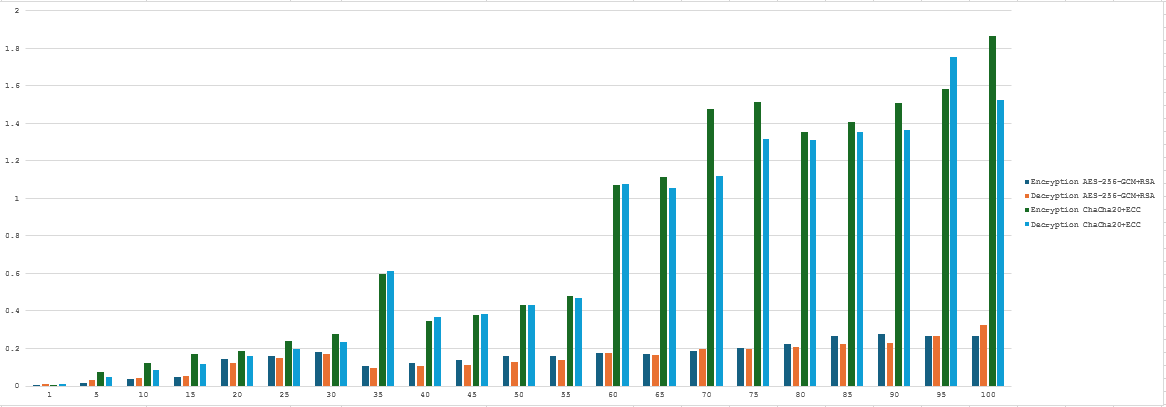


Figure 5.4 – Results of the comparison between in Guest and Basic level

ChaCha20 + ECC (Curve25519) is the better algorithm for environments where speed and computational efficiency are paramount. However, AES-256-GCM + RSA remains a strong contender for scenarios demanding heightened security, despite its performance trade-offs.

### **Level 1 Basic: Comparison of Algorithms**

#### **1-AES-128-CCM + ChaCha20 + ECC (Curve25519)**

This combination integrates AES-128-CCM, ChaCha20, and ECC (Curve25519) to create an efficient and secure hybrid encryption scheme. AES-128-CCM is a block cipher encryption mode that provides both confidentiality and authentication. By pairing it with ChaCha20, a high-speed stream cipher optimized for modern processors, the scheme ensures fast and secure data encryption. Additionally, ECC (Curve25519) facilitates a lightweight and highly secure key exchange mechanism, reducing computational overhead compared to RSA. This combination excels in performance-critical environments, offering low encryption and decryption times while maintaining strong cryptographic security.

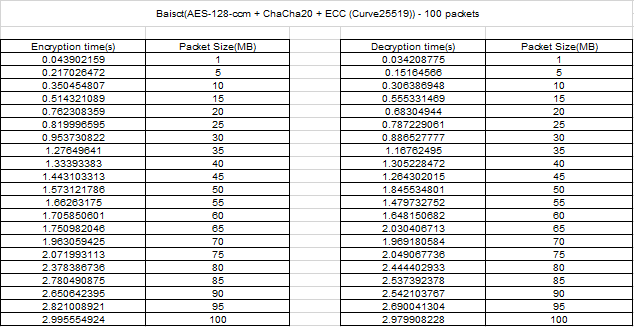


Table 5.3 – Values obtained for encryption and decryption in Basic level.

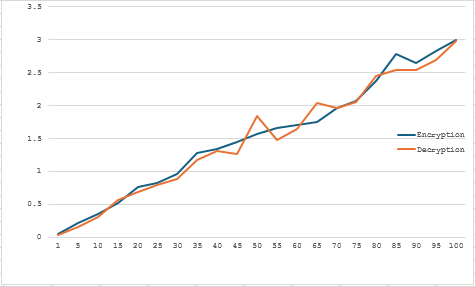


Figure 5.5 – Results of the encryption time in the Basic level.

#### **AES-256-GCM + ChaCha20 + RSA**

This scheme employs AES-256-GCM, ChaCha20, and RSA to achieve a balance between security and functionality. AES-256-GCM is a widely trusted symmetric encryption algorithm known for its high-level security and efficient authentication using Galois Counter Mode (GCM). Coupled with ChaCha20, the stream cipher boosts encryption speed, while RSA ensures secure asymmetric key exchange. However, RSA is computationally intensive, which impacts the overall performance of this combination, especially for large data packets. While it provides exceptional security, the processing time is higher compared to combinations that use ECC for key exchange.

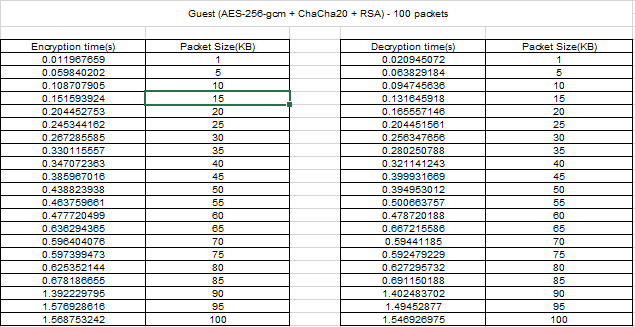


Table 5.4 – Values obtained for encryption and decryption in Guest level.

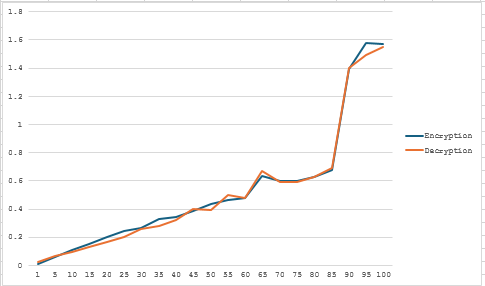


Figure 5.6 – Results of the encryption time in the Basic level.

### **Comparison and Conclusion**

In terms of encryption and decryption speeds, **AES-128-CCM + ChaCha20 + ECC (Curve25519)** outperforms **AES-256-GCM + ChaCha20 + RSA**. The streamlined design of ECC reduces key exchange overhead significantly, making it more efficient than RSA, which requires larger key sizes and greater computational resources. Additionally, the combination of AES-128-CCM and ChaCha20 offers a faster, lightweight alternative compared to AES-256-GCM, which, while secure, is more resource-intensive due to its 256-bit key size.

Security-wise, AES-256-GCM offers a higher encryption key length, providing a stronger security guarantee compared to AES-128-CCM. However, for scenarios where speed and efficiency are critical—such as real-time applications or systems with limited computational resources—**AES-128-CCM + ChaCha20 + ECC (Curve25519)** proves to be the better choice.

While both combinations offer strong encryption and secure key exchange mechanisms, **AES-128-CCM + ChaCha20 + ECC (Curve25519)** is the better choice for environments prioritizing performance and efficiency. On the other hand, **AES-256-GCM + ChaCha20 + RSA** is more suitable for applications that require the highest levels of cryptographic strength and can tolerate higher processing times.

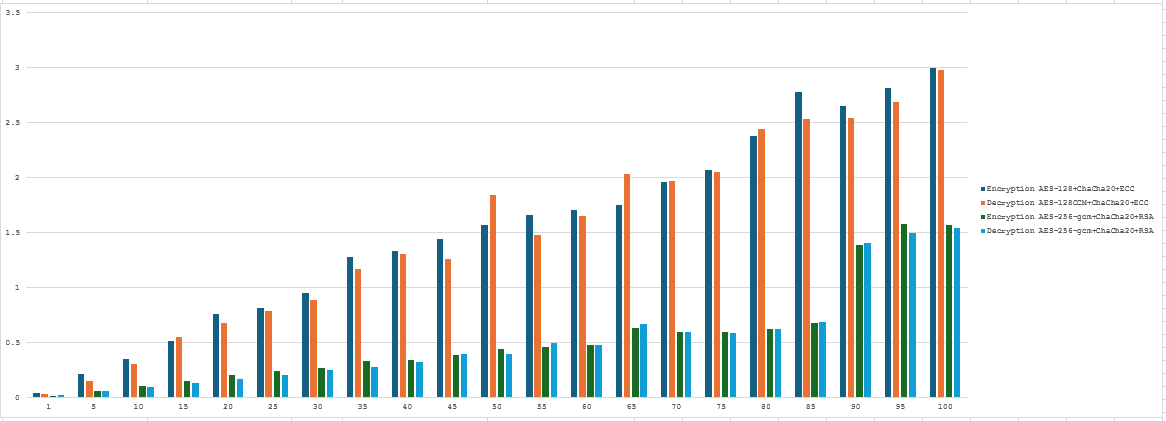


Figure 5.7 – Results of the encryption time in the Basic level.

### **Level 1 Advanced: Comparison of Encryption Algorithms**

#### **1. ChaCha20 + AES-256-GCM**

This combination uses **ChaCha20** as the stream cipher for encryption and **AES-256-GCM** (Galois/Counter Mode) for authenticated encryption. ChaCha20 is known for its efficiency and speed, especially in environments where hardware acceleration for AES is not available. **AES-256-GCM** provides strong encryption with a 256-bit key length and an integrated authentication tag.

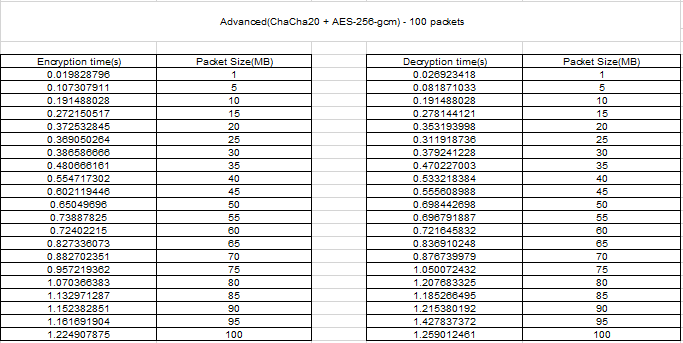


Table 5.5 – Values obtained for encryption and decryption in Advanced level.

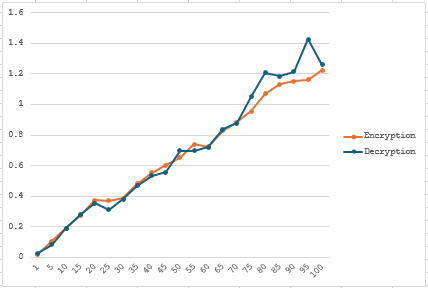


Figure 5.7 – Results of the encryption time in the Advanced level.

#### **2. AES-128-CCM + RSA**

This combination uses **AES-128-CCM** for encryption and **RSA** for key exchange. **AES-128-CCM** offers authenticated encryption with a smaller key size, which is computationally more efficient than AES-256-GCM. **RSA**, although reliable and widely used for public-key encryption, tends to be slower than ECC due to its larger key sizes.

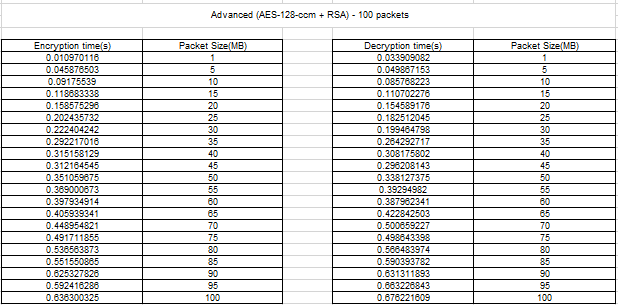


Table 5.6 – Values obtained for encryption and decryption in Advanced level.

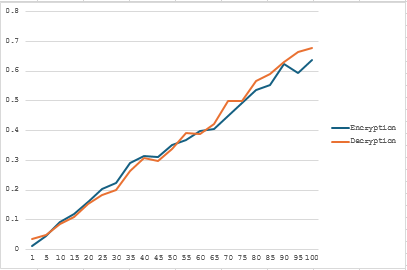


Figure 5.8 – Results of the encryption time in the Advanced level.

#### **3. AES-128-CCM + AES-256-GCM + ECC (Curve25519)**

This combination uses **AES-128-CCM** for lightweight encryption, **AES-256-GCM** for stronger encryption, and **ECC (Curve25519)** for efficient key exchange. By combining these three methods, it provides both high security and high performance, making it suitable for systems that need both robustness and efficiency.

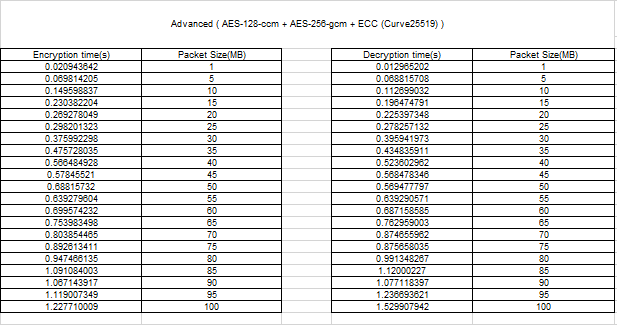


Table 5.7 – Values obtained for encryption and decryption in Advanced level.

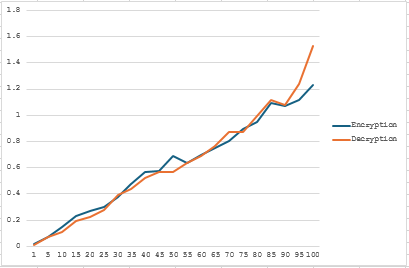


Figure 5.9 – Results of the encryption time in the Advanced level.

### **Performance and Comparison**

ChaCha20 combined with AES-256-GCM offers an excellent balance between speed and security, making it particularly effective in environments where hardware acceleration for AES is unavailable. AES-256-GCM provides robust encryption and authentication, while ChaCha20 ensures fast stream cipher encryption, enhancing overall performance. In contrast, the combination of AES-128-CCM and RSA delivers strong security through RSA's public-key encryption system, paired with AES-128-CCM's efficient encryption and authentication capabilities. However, RSA's computational intensity can result in slower performance, making it more suitable for applications where security outweighs speed concerns. On the other hand, AES-128-CCM, AES-256-GCM, and ECC (Curve25519) together optimize both performance and security. AES-128-CCM and AES-256-GCM provide layered encryption with varying strength levels, while ECC ensures fast and secure key exchange. This combination is ideal for high-security applications where both cryptographic robustness and operational efficiency are critical. Overall, ChaCha20 with AES-256-GCM stands out for performance in less optimized environments, while AES-128-CCM with RSA serves security-focused needs, and AES-128-CCM with AES-256-GCM and ECC is the most versatile and secure option.

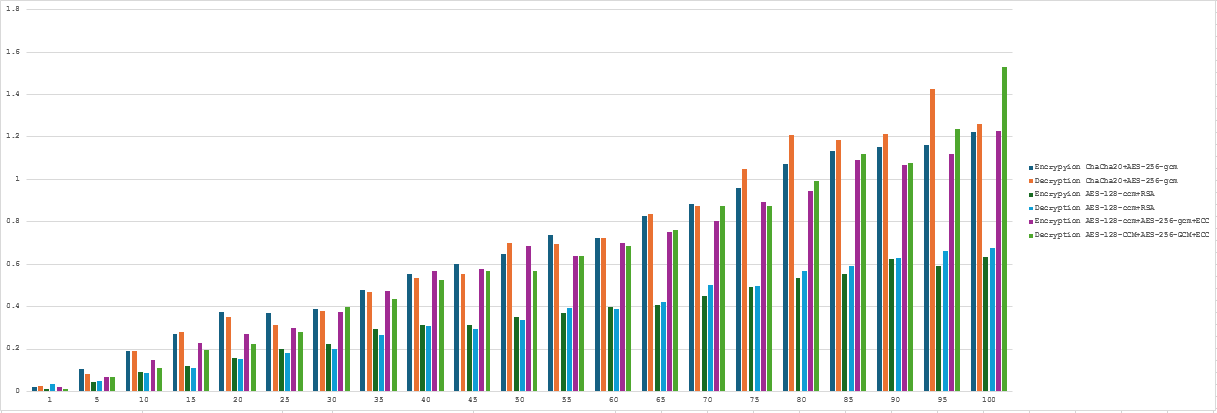


Figure 5.10 – Results of the encryption time in the Advanced level.

### **Level 1 Admin: Encryption Algorithms Comparison for Admin Use Cases**

#### **1. AES-256-GCM + ChaCha20 + ECC (Curve25519)**

This combination leverages **AES-256-GCM** for authenticated encryption, **ChaCha20** for stream cipher encryption, and **ECC (Curve25519)** for efficient key exchange. This is a powerful combination that provides high levels of security, speed, and resistance to various types of cryptographic attacks, making it suitable for both high-performance and highly secure applications.

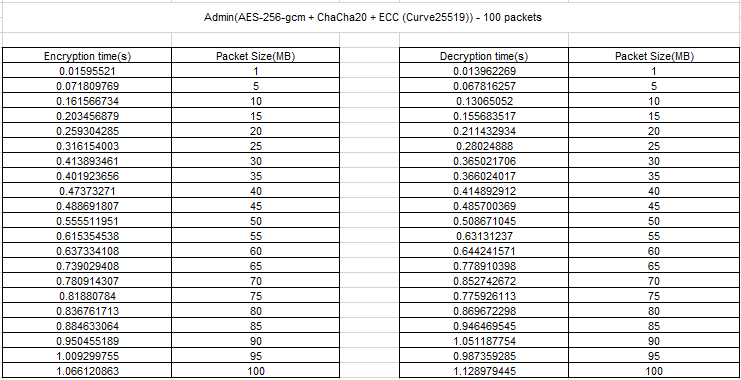


Table 5.8 – Values obtained for encryption and decryption in Admin level.

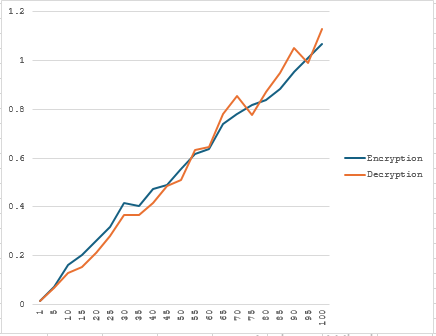


Figure 5.11 – Results of the encryption time in the Admin level.

#### **2. AES-128-CCM + ChaCha20 + RSA**

This combination uses **AES-128-CCM** for authenticated encryption, **ChaCha20** for encryption, and **RSA** for secure key exchange. **RSA** is well-known for providing a high level of security for key exchange, although it can be computationally more expensive compared to **ECC**. This scheme is good for administrative environments requiring strong encryption and the use of public-key infrastructure (PKI) for key management.

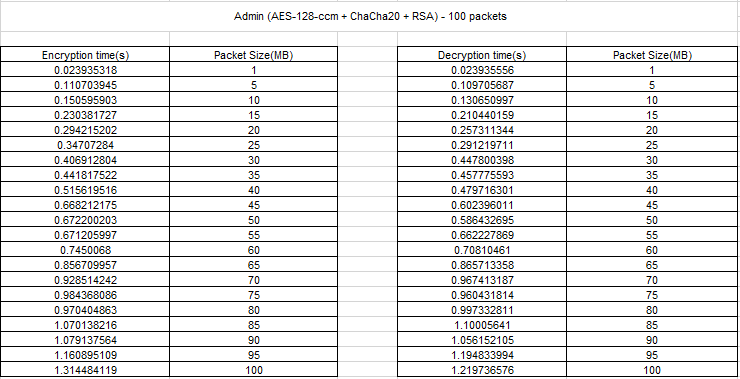


Table 5.9 – Values obtained for encryption and decryption in Admin level.

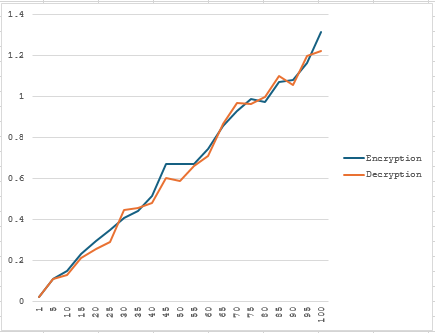


Figure 5.12 – Results of the encryption time in the Admin level.

#### **3. ChaCha20 + ECC (Curve25519) + RSA**

This combination uses **ChaCha20** for fast stream cipher encryption, **ECC (Curve25519)** for efficient key exchange, and **RSA** for public-key encryption. While **RSA** is effective for securing keys, its larger key sizes make it slower compared to ECC. However, **ChaCha20** ensures fast encryption, while **ECC** provides a lightweight, secure method for key exchange, making this combination ideal for environments where both speed and security are important.

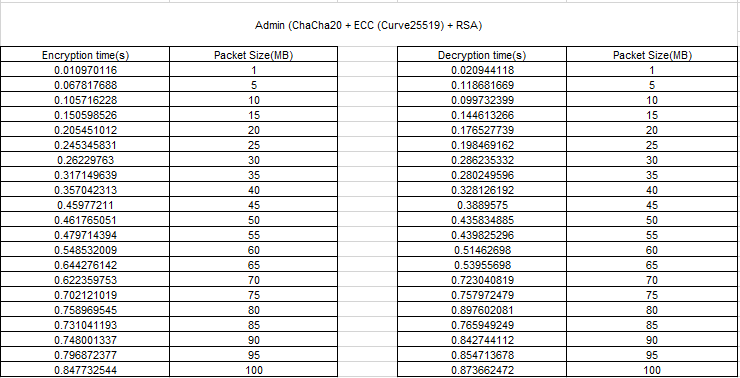


Table 5.10 – Values obtained for encryption and decryption in Admin level.

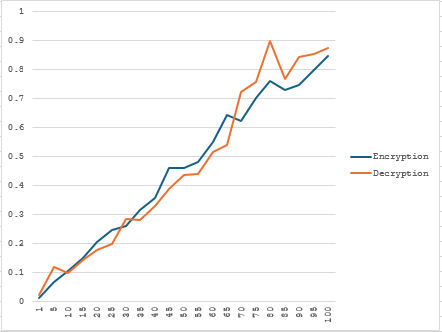


Figure 5.13 – Results of the encryption time in the Admin level.

### **Performance and Comparison**

The combination of AES-256-GCM, ChaCha20, and ECC (Curve25519) offers a high-performance and secure encryption scheme, integrating AES for strong encryption, ChaCha20 for speed, and ECC for efficient key exchange, making it a versatile choice for both performance and security-critical applications. Alternatively, AES-128-CCM with ChaCha20 and RSA provides effective authenticated encryption and secure public-key exchange, though RSA's computational demands can lead to slower performance compared to ECC. For scenarios prioritizing speed, the combination of ChaCha20, ECC (Curve25519), and RSA strikes a balance by delivering fast encryption through ChaCha20, efficient key exchange with ECC, and robust security with RSA. Overall, AES-256-GCM with ChaCha20 and ECC (Curve25519) stands out as the most robust and efficient solution, while the other configurations cater to specific needs such as authenticated encryption or environments with stricter speed requirements.

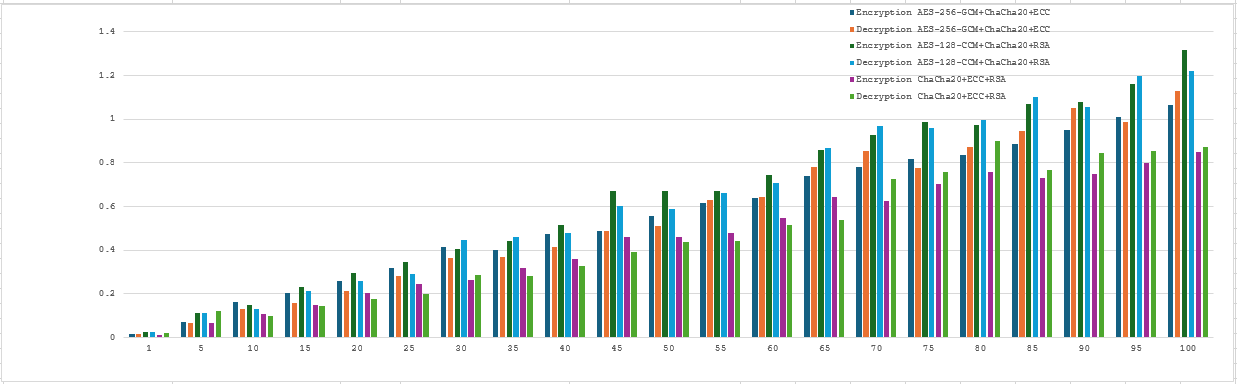


Figure 5.14 – Results of the encryption time in the Admin level.

### **Level 2: Guest**

#### **AES-128-CCM + ChaCha20**

The AES-128-CCM combined with ChaCha20 offers a hybrid encryption mechanism that balances security and performance. AES-128-CCM provides authenticated encryption with associated data (AEAD), ensuring both the confidentiality and integrity of the transmitted data. ChaCha20, on the other hand, is a fast and efficient stream cipher designed for high-speed encryption. Together, they create a secure environment suitable for applications requiring lightweight and low-latency data protection, such as real-time communications or IoT devices.

Performance analysis shows that encryption times for small packets (1 MB) are as low as 0.0249 seconds, with decryption times slightly faster at 0.0199 seconds. As packet sizes increase, processing times also grow, with encryption times for 100 MB packets reaching 1.36 seconds and decryption times slightly higher at 1.38 seconds. The dataset below highlights the consistent performance for encryption and decryption across various packet sizes:

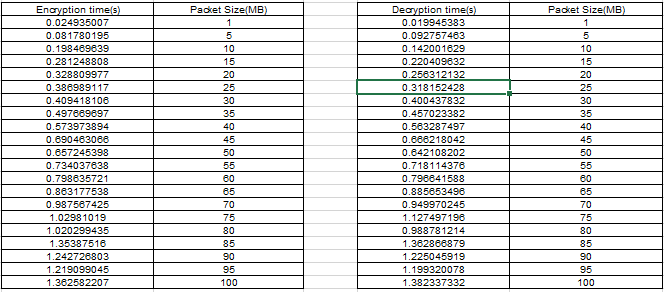


Table 5.11 – Values obtained for encryption and decryption in Guest level.

This method is especially suitable for secure real-time data transfer, where both high speed and robust encryption are crucial. Below is a graphical representation of encryption and decryption times for AES-128-CCM + ChaCha20.

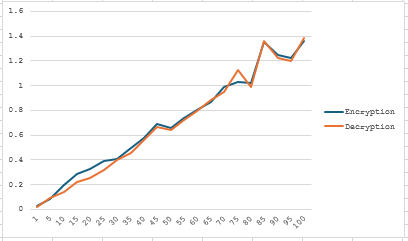


Figure 5.15 – Results of the encryption time in the Guest level.

#### **AES-128-CCM + AES-192-CCM**

The combination of AES-128-CCM and AES-192-CCM layers two authenticated encryption mechanisms, enhancing the overall security of data transmission. AES-128-CCM, with its 128-bit encryption, is optimized for high-speed processing, while AES-192-CCM adds an extra layer of protection with a longer 192-bit key. This combination is particularly suited for scenarios demanding higher encryption standards without significantly impacting performance.

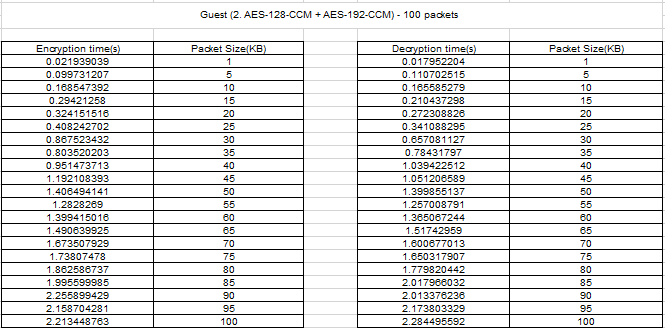


Table 5.12 – Values obtained for encryption and decryption in Guest level.

The performance analysis indicates that encryption and decryption times remain efficient for smaller packet sizes, with 1 KB packets being encrypted in 0.0219 seconds and decrypted in 0.0179 seconds. For larger packet sizes of 100 KB, encryption times reach 2.21 seconds, while decryption times are slightly higher at 2.28 seconds. Detailed performance metrics are provided in the table below:

This method is ideal for applications requiring enhanced encryption levels, such as secure financial transactions or sensitive data exchanges. Below is a graphical comparison of encryption and decryption times for AES-128-CCM + AES-192-CCM.

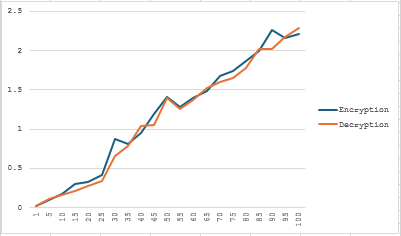


Figure 5.16 – Results of the encryption time in the Guest level

### **Performance and Comparison**

A side-by-side comparison of the encryption methods highlights the trade-offs in terms of processing speed and security.

1. **AES-128-CCM + ChaCha20**:
   * Demonstrates faster encryption and decryption times for smaller packets, making it ideal for real-time applications.
   * Maintains consistent performance across a range of packet sizes, with only marginal slowdowns for larger data.
2. **AES-128-CCM + AES-192-CCM**:
   * Provides an added layer of security through the combination of 128-bit and 192-bit encryption.
   * Encryption and decryption times are higher, especially for larger packet sizes, due to the computational complexity introduced by the longer key length.

In summary, AES-128-CCM + ChaCha20 is better suited for environments requiring low latency and moderate security, while AES-128-CCM + AES-192-CCM is preferable for scenarios prioritizing enhanced encryption at the cost of speed.

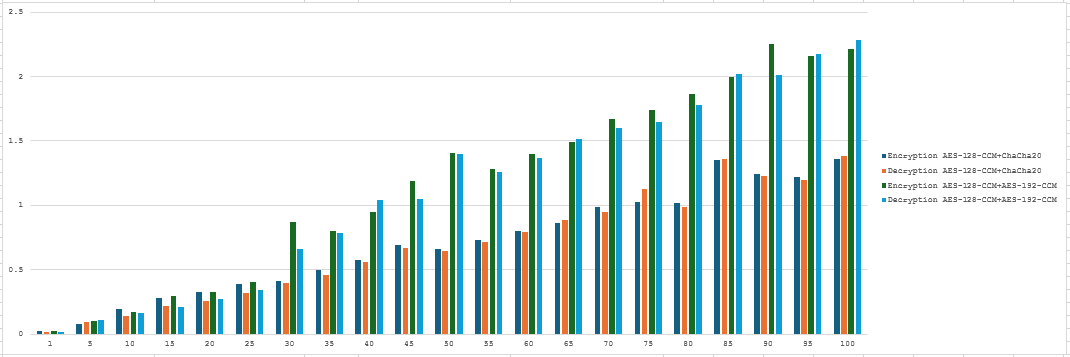


Figure 5.17 – Results of the encryption time in the Guest level

The analysis of the two encryption methods reveals that each has unique strengths tailored to specific use cases. **AES-128-CCM + ChaCha20** strikes a balance between speed and security, making it ideal for applications like IoT, real-time streaming, or lightweight communications. On the other hand, **AES-128-CCM + AES-192-CCM** offers an elevated level of encryption strength, suitable for sensitive data exchanges such as financial transactions, at the expense of increased processing times.

Both methods provide robust security features, ensuring that data remains confidential and tamper-proof. The choice between them depends on the priority of the application—speed or enhanced security. Future optimizations could explore combining these methods further or introducing dynamic encryption techniques for improved adaptability.

### **Level 2: BASIC**

#### **AES-256-CCM + ChaCha20-Poly1305**

This scheme combines AES-256-CCM for authenticated encryption and ChaCha20-Poly1305 for high-speed stream cipher encryption. The encryption times show steady growth with increasing packet sizes, reaching 1.428 seconds for 100 MB, while the decryption time is 1.338 seconds for the same size. This combination balances strong encryption and speed, making it ideal for scenarios where robust security and quick performance are required. The efficiency of ChaCha20-Poly1305 complements the authenticated encryption provided by AES-256-CCM, ensuring fast and reliable data processing.

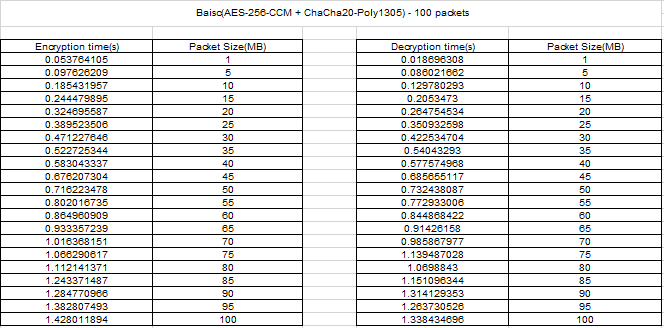


Table 5.13 – Values obtained for encryption and decryption in Basic level.

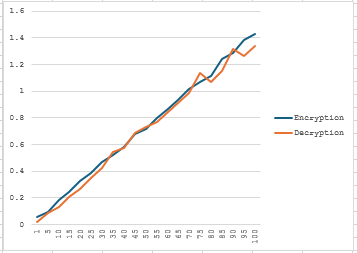


Figure 5.18 – Results of the encryption time in the basic level

#### **AES-128-CCM + AES-192-CCM + XChaCha20**

This scheme integrates three encryption methods: AES-128-CCM for efficiency, AES-192-CCM for added robustness, and XChaCha20 for modern stream cipher performance. Encryption times are slightly longer compared to the first scheme, reaching 1.966 seconds for 100 MB packets, with decryption times of 1.865 seconds for the same packet size. This multi-layered approach increases computational overhead, making it slightly slower, but it enhances compatibility and redundancy, catering to use cases where diverse encryption schemes are needed.

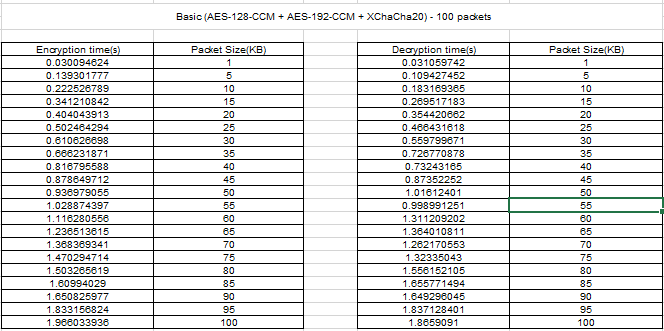


Table 5.14 – Values obtained for encryption and decryption in Basic level.

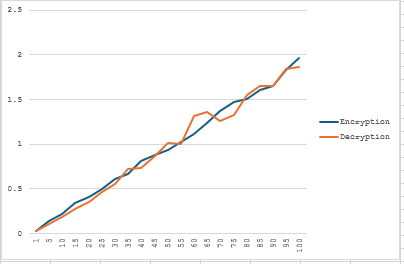


Figure 5.19 – Results of the encryption time in the basic level

### **Performance and Comparison**

* **Encryption Time:** AES-256-CCM + ChaCha20-Poly1305 demonstrates faster encryption times, with significant differences in smaller packet sizes. For 100 MB, it is approximately 0.54 seconds faster than AES-128-CCM + AES-192-CCM + XChaCha20.
* **Decryption Time:** Both schemes perform efficiently, but AES-256-CCM + ChaCha20-Poly1305 is consistently quicker. Its decryption time for 100 MB packets is 1.338 seconds compared to 1.865 seconds for the alternative scheme.
* **Scalability:** While both schemes scale well with increasing packet sizes, AES-256-CCM + ChaCha20-Poly1305 outperforms the other in both encryption and decryption time, especially as data volume grows.

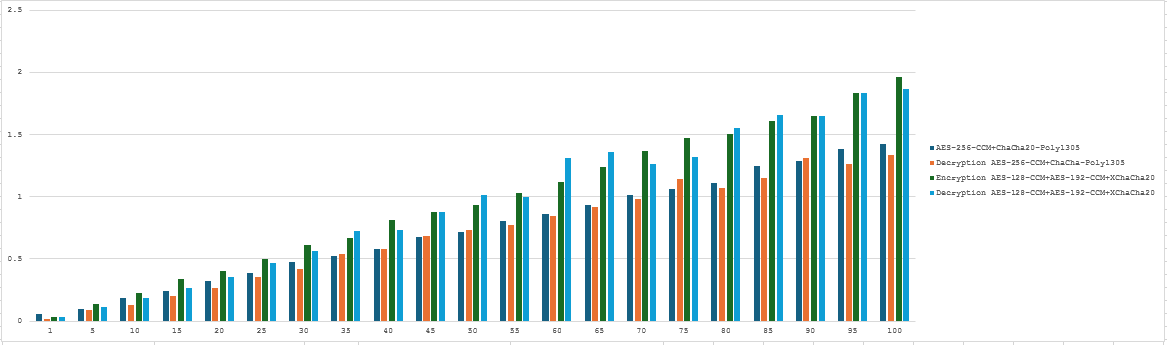


Figure 5.21 – Results of the encryption time in the basic level

**AES-256-CCM + ChaCha20-Poly1305** is the better algorithm for applications requiring high performance, speed, and strong encryption. It is ideal for environments where encryption speed and secure data handling are critical. Meanwhile, **AES-128-CCM + AES-192-CCM + XChaCha20** offers a more layered encryption process, making it suitable for scenarios demanding added security through redundancy and diverse algorithms, though at the expense of performance.

### **Level 2 Advanced**

#### **AES-128-CCM + AES-256-CCM + ChaCha20**

The encryption combination of AES-128-CCM, AES-256-CCM, and ChaCha20 represents a highly secure and efficient framework. AES-128-CCM provides baseline encryption for smaller key sizes with robust performance. Meanwhile, AES-256-CCM enhances this with an extended key length, significantly increasing cryptographic security. ChaCha20, a high-speed stream cipher, complements these methods by improving overall speed and flexibility, especially for environments with limited hardware acceleration for AES.

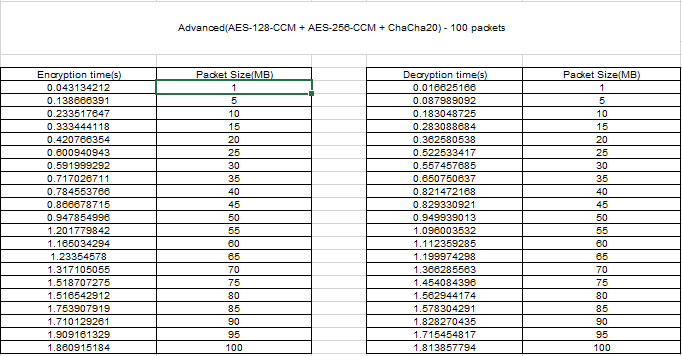


Table 5.15 – Values obtained for encryption and decryption in Advanced level.

Performance metrics demonstrate consistent encryption and decryption times across packet sizes. For instance, encrypting 1 MB packets takes approximately 0.0431 seconds, while decryption is even faster at 0.0166 seconds. As packet sizes increase, the process remains efficient, with 100 MB packets requiring around 1.86 seconds for encryption and 1.81 seconds for decryption. The data below highlights this trend:

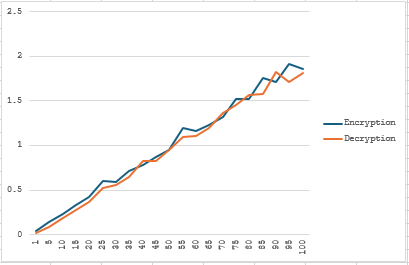


Figure 5.22– Results of the encryption time in the Advanced level

#### **AES-256-CCM + XChaCha20 + ChaCha20**

AES-256-CCM, XChaCha20, and ChaCha20 together provide a triple-layer encryption system designed for environments where both security and speed are critical. AES-256-CCM forms the foundation with its extended 256-bit encryption, while XChaCha20, an extended variant of ChaCha20, offers nonce misuse resistance and broader security guarantees. ChaCha20 further enhances speed and adaptability, particularly in low-resource environments.

For smaller packets, the encryption time is extremely low, with 1 MB packets processed in 0.0259 seconds and decrypted in 0.0207 seconds. The method scales efficiently as packet sizes grow, with encryption and decryption times for 100 MB packets being 1.92 seconds and 1.81 seconds, respectively. The performance data below highlights the stability across varying packet sizes:

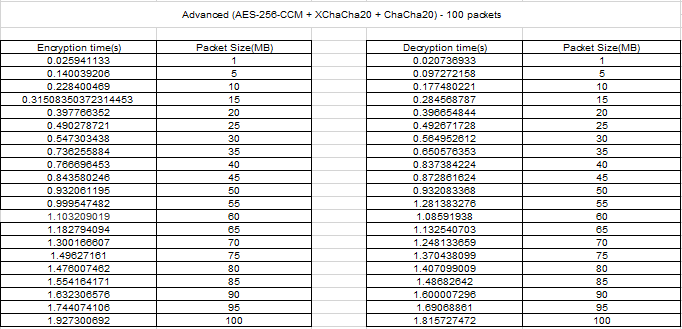


Table 5.16 – Values obtained for encryption and decryption in Advanced level.

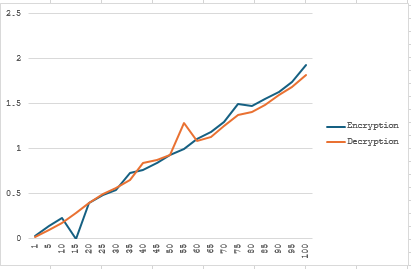


Figure 5.23– Results of the encryption time in the Advanced level

#### **AES-192-CCM + XChaCha20**

AES-192-CCM combined with XChaCha20 offers a streamlined encryption solution with intermediate key length and nonce-based security. AES-192-CCM provides a balanced approach between the speed of AES-128 and the security strength of AES-256, while XChaCha20 improves encryption efficiency, ensuring robust protection against nonce reuse attacks.

Encryption times for this method are particularly efficient for smaller packet sizes, with 1 MB packets encrypted in 0.0139 seconds and decrypted in 0.0054 seconds. For larger packet sizes, such as 100 MB, encryption and decryption times are 1.30 seconds and 1.21 seconds, respectively, demonstrating its suitability for applications involving moderate-to-large data transmissions.

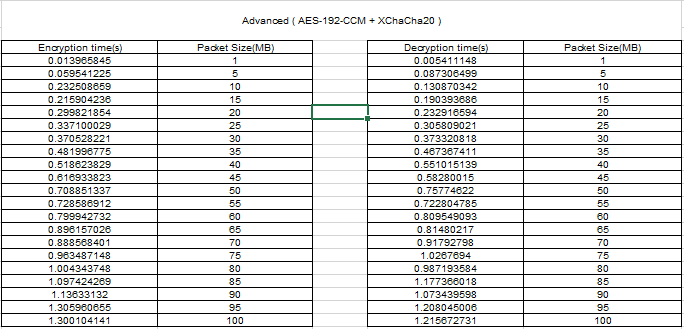


Table 5.17 – Values obtained for encryption and decryption in Advanced level.

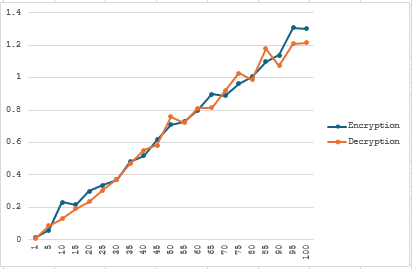


Figure 5.24– Results of the encryption time in the Advanced level

### **Performance and Comparison**

1. **AES-128-CCM + AES-256-CCM + ChaCha20**:
   * Provides the strongest cryptographic layering but shows slightly higher encryption times for larger packets compared to other methods.
   * Excels in environments demanding robust security for large-scale file sharing.
2. **AES-256-CCM + XChaCha20 + ChaCha20**:
   * Combines high security with excellent speed, making it the fastest option for packet sizes above 50 MB.
   * Well-suited for low-latency applications, such as live streaming or secure messaging.
3. **AES-192-CCM + XChaCha20**:
   * Offers a middle-ground solution with efficient encryption times for both small and large packets.
   * Ideal for moderately sensitive applications like web communications or financial data exchanges

.

The advanced encryption methods analyzed demonstrate significant enhancements in security, efficiency, and scalability. **AES-128-CCM + AES-256-CCM + ChaCha20** is the most robust but slightly slower for large packet sizes. **AES-256-CCM + XChaCha20 + ChaCha20** balances high-level security with the fastest processing times, making it suitable for real-time applications. Lastly, **AES-192-CCM + XChaCha20** offers a versatile and efficient encryption solution for moderate-security scenarios.

Each method presents unique strengths tailored to specific needs, whether it's high-speed encryption for real-time data transfer or maximum security for sensitive information. Future advancements may combine these approaches into adaptive encryption frameworks for broader usability.

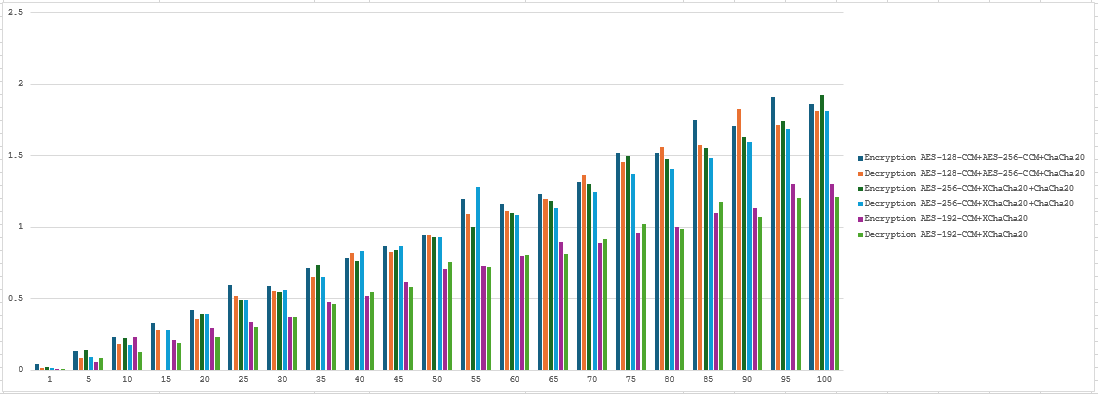


Figure 5.26– Results of the encryption time in the Advanced level

### **Admin**

1. **AES-256-CCM + AES-128-CCM + ChaCha20:**

This configuration combines two block cipher modes, AES-256-CCM and AES-128-CCM, with the lightweight ChaCha20 stream cipher, aiming for a balance between computational overhead and encryption speed.

This combination exhibits a steady increase in encryption and decryption times as packet sizes grow. For smaller packets (1 MB to 10 MB), the encryption times range between 0.026 seconds and 0.218 seconds, while for larger packets (95 MB to 100 MB), the encryption time grows to approximately 1.857 seconds. The decryption process shows a similar pattern, indicating this configuration performs reliably for consistent security.

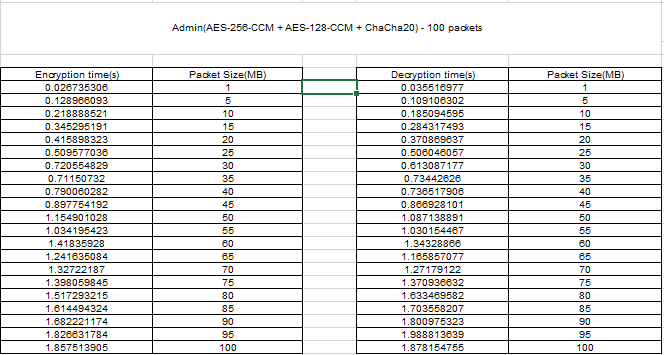


Table 5.18 – Values obtained for encryption and decryption in Admin level.

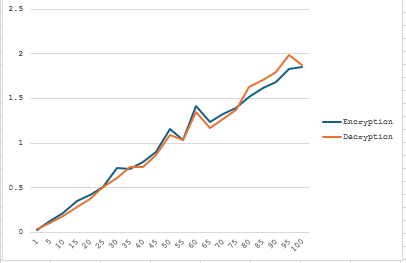


Figure 5.27– Results of the encryption time in the Admin level

1. **AES-192-CCM + AES-256-CCM + XChaCha20:**

By using a higher key length AES-192-CCM and AES-256-CCM along with XChaCha20, this approach targets maximum security while maintaining efficiency.

This combination showcases improved performance over the previous algorithm, especially for small and medium packet sizes. The encryption time for a 1 MB packet is only 0.021 seconds, and for a 100 MB packet, it is 1.936 seconds. The decryption times are similarly efficient, showing this configuration is well-suited for applications requiring secure communication with optimized speed.

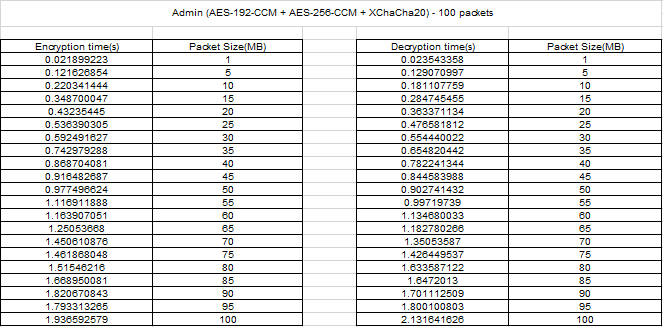


Table 5.19 – Values obtained for encryption and decryption in Admin level.

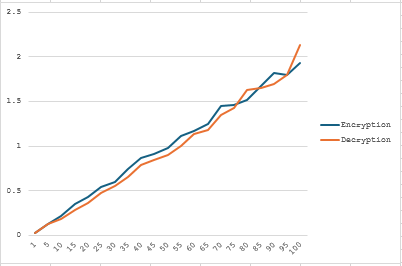


Figure 5.28– Results of the encryption time in the Admin level

1. **AES-128-CCM + ChaCha20-Poly1305 + XChaCha20:**

This variant incorporates AES-128-CCM with two highly optimized stream cipher algorithms, ChaCha20-Poly1305 and XChaCha20, aiming for faster operations and reduced latency.

This configuration leverages the strengths of lightweight stream ciphers, resulting in impressive encryption times. For 1 MB packets, encryption takes 0.028 seconds, while for 100 MB packets, it increases to 2.052 seconds. While its encryption time is slightly higher for smaller packets compared to AES-192-CCM + AES-256-CCM + XChaCha20, it maintains competitive decryption performance, making it ideal for scenarios prioritizing lightweight operations.

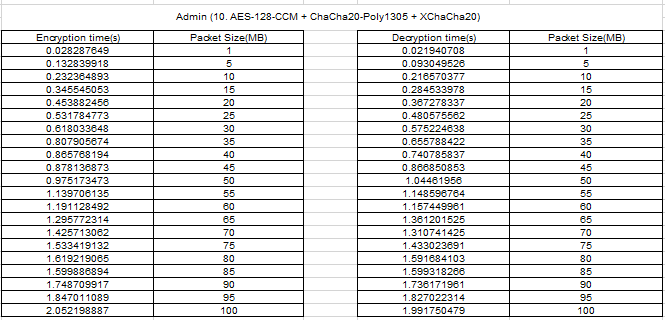


Table 5.20 – Values obtained for encryption and decryption in Admin level.

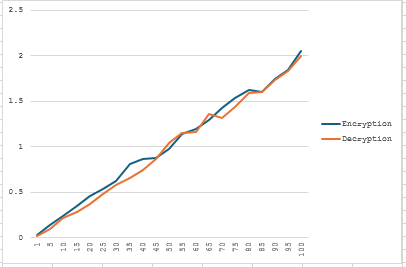


Figure 5.29– Results of the encryption time in the Admin level

#### **Best Performing Algorithm**

Based on the results, **AES-192-CCM + AES-256-CCM + XChaCha20** stands out as the most balanced and efficient algorithm. It achieves superior encryption and decryption times, especially for smaller and medium-sized packets, while maintaining high levels of security. Its performance edge is particularly noticeable in applications where rapid data transmission is crucial.

In conclusion, the Admin algorithms provide a range of choices catering to different security and performance requirements. Among them, AES-192-CCM + AES-256-CCM + XChaCha20 emerges as the best choice for applications needing both high security and efficient processing times.

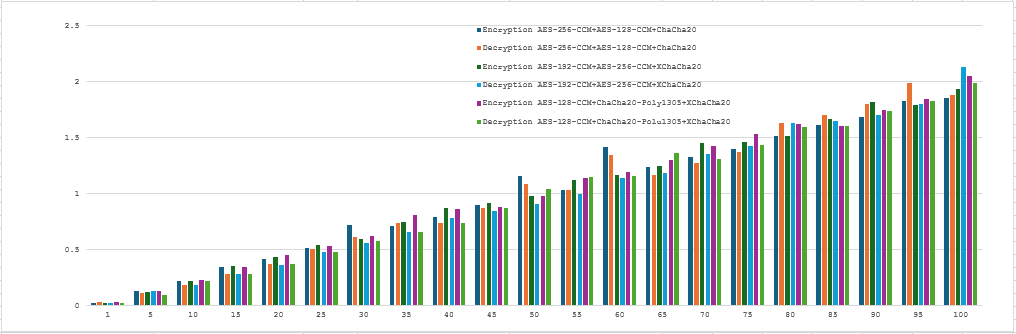


Figure 5.30– Results of the encryption time in the Admin level