Engineering Degree Project

Anonymization of Sensitive Data through Cryptography

Author: Isac Holm, Johan Dahl
Supervisor: Tobias Andersson Gidlund
Lnu Supervisor: Tobias Andersson Gidlund
Semester: Spring 2023
Subject: Computer Science

Abstract

In today's interconnected digital landscape, the protection of sensitive information is of great importance. As a result, the field of cryptography plays a vital role in ensuring individuals' anonymity and data integrity. In this context, this thesis presents a comprehensive analysis of symmetric encryption algorithms, specifically focusing on the Advanced Encryption Standard (AES) and Camellia. By investigating the performance aspects of these algorithms, including encryption time, decryption time, and ciphertext size, the goal is to provide valuable insights for selecting suitable cryptographic solutions. The findings indicate that while there is a difference in performance between the algorithms, the disparity is not substantial in practical terms. Both AES and Camellia, as well as their larger key-size alternatives, demonstrated comparable performance, with AES128 showing marginally faster encryption time. The study's implementation also involves encrypting a data set with sensitive information on students. It encrypts the school classes with separate keys and assigns roles to users, enabling access control based on user roles. The implemented solution successfully addressed the problem of role-based access control and encryption of unique identifiers, as verified through the verification and validation method. The implications of this study extend to industries and society, where cryptography plays a vital role in protecting individuals' anonymity and data integrity. The results presented in this paper can serve as a valuable reference for selecting suitable cryptographic algorithms for various systems and applications, particularly for anonymization of usernames or short, unique identifiers. However, it is important to note that the experiment primarily focused on small data sets, and further investigations may yield different results for larger data sets.
Keywords: Cryptography, Encryption, Decryption, AES, Camellia

Contents

1 Introduction

This paper is done in regard to a 15 HEC bachelor's degree project. It investigates the field of computer security and touches on subjects concerning cryptography and privacy preservation. The goal is to create a software solution that encrypts data to protect the privacy of the data subjects. The paper puts a high emphasis on the performance of the encryption algorithm used. Controlled experiments are carried out to evaluate the algorithms in regard to speed and cipher size metrics. The complete implementation is also verified on functional requirements using automatic testing.

Chapter 1 serves as the introductory chapter of this thesis, setting the stage for the research undertaken. It provides a comprehensive overview of the problem under investigation. The chapter is structured into several subchapters, each dedicated to a distinct topic or theory. It begins with a detailed background of the problem, followed by a review of related work in the field. The problem formulation is then presented, clearly defining the research question and objectives. The chapter also outlines the motivation behind the study, providing a rationale for the research and its significance. It presents the milestones that guided the progression of the thesis and discusses the scope and limitations of the study.

1.1 Background

Computer security as a field started to grow in the 1960s and has since then led to a deeper understanding of threats in the world of computers and how to manage them. Its most fundamental form addresses four basic premises: vulnerabilities, threats, incidents, and controls of a system [1]. However, as technology rapidly evolves, new threats and vulnerabilities within the field arise. With the growing popularity of using cloud-based capabilities for data storage among enterprises [2], and the great amount of data that is collected today through rising technologies such as IoT or social media. Stakeholders' concerns about how to maintain the confidentiality of sensitive data have risen. This thesis addresses the field of computer security and more specifically the problem of how to maintain the privacy of individuals in a data set containing sensitive information.

The problem originates from a request of the researcher group EdTechLnu. EdTechLnu conducts research within the field of educational technology, with three main areas: Technical research, Pedagogical research, and Organizational research. Some goals of EdTechLnu's research include developing new technology that supports teaching and learning, methods to study the effects of technology-supported learning, as well as analyzing and integrating the use of data related to students' online learning behaviors to increase the quality of education [3].

1.2 Related Work

A comprehensive performance study of four symmetric encryption algorithms - Triple Data Encryption Standard (3DES), Advanced Encryption Standard (AES), Blowfish, and Twofish is provided in [4]. The study conducts experiments on the different algorithms by measuring execution time, memory utilization, and cipher text size against different file sizes. The experiments are conducted by implementing the algorithms in a .NET environment using C#. The results show that AES has the lowest execution time in terms of encryption and decryption. AES and 3DES consume a similar amount of memory in the encryption process. Blowfish and Twofish perform worst in the majority of the

experiments. However, as the file size increases, 3DES is outperformed by the Blowfish algorithm.

In [5] a performance evaluation on AES is carried out. The author investigates how the algorithm performs using different input configurations, such as key size and the chaining modes Cipher Block Chaining (CBC) and Electronic Codebook (ECB), but also different hardware configurations. The results show that the average increase in encryption time over the different hardware configurations and file sizes is approximately 16.4% when using a greater key size. An additional 4.6% increase is added when using the CBC mode. In terms of decryption time, the numbers result in a 15.4% increase without CBC, and an additional 7.5% with CBC.

The report in [6] provides another evaluation of various symmetric encryption algorithms, such as AES, ARC2, Blowfish, CAST, and 3DES. The evaluation is implemented in Python [7]. Some conclusions from the experiments conducted in the report are that the Blowfish algorithm performs the fastest. AES provides a more consistent performance overall in comparison to the other algorithms, with the exception of Blowfish. However, the authors also emphasize the importance of the block sizes when comparing AES and Blowfish. Blowfish is a 64-bit block cipher, making it more vulnerable to birthday attacks than AES and its 128-bit blocks.

In [8], the author investigates the encryption and decryption process of the AES algorithm. A brief introduction to cryptography, in general, is given, along with the history of AES. The paper proceeds to in detail describe the encryption-decryption and key generation process of AES.

1.3 Problem Formulation

The task, as formulated by EdTechLnu is to find a method that anonymizes a data set of information on school students. The data holds unique identifiers that need to be encrypted to ensure the confidentiality of the students. Decryption should only be possible by school staff with certain roles. For example, the head principal should be able to decrypt and access the information on all students in their school. In contrast, class principals should be able to decrypt and view the information of the students in their class only. The implementation should also support the need of revoking access to the information, in case of changes to the staff in the school. Hence, the problem of possibly having to re-encrypt the data also needs to be addressed. In regard to this, the following research questions will be examined throughout this thesis:

RQ1 What are the most suitable encryption algorithms for protecting the unique identifiers in a student data set, and why are they considered suitable?

RQ2 How do the identified encryption algorithms perform in terms of speed and resource usage when applied to the student data set?

RQ3 How can a role or identity-based decryption system be designed and implemented to ensure secure access to the student data set, and how can this system accommodate changes in access privileges?

The expected results of this thesis are to identify several encryption algorithms that achieve a suitable cryptographic storage solution for the student data set. Implement one or several of these encryption algorithms and evaluate the performance of these algorithms through a series of controlled experiments. Additionally, an access control mechanism that satisfies the functional requirements made by the stakeholders will be implemented and tested through automatic unit testing.

1.4 Motivation

Emerging technologies such as the Internet of Things (IoT) gather and distribute big data on sensitive user information across the internet [9]. In the annual report of the cybersecurity threat landscape by the European Union Agency for Cybersecurity (ENISA) [10] it is stated that the global production and consumption of data amounted to 79 zettabytes in 2021. A number that is projected to increase to more than 180 zettabytes by 2025. Furthermore, this new role of data used by companies has enabled a data-driven economy, consequently making data a significant target for cybercriminals. The report also states that data compromises increased by 68% over the year 2020 and that this increase in leaked personal and sensitive data has led to a significant increase in identity theft. Moreover, privacy laws such as the Health Insurance Portability and Accountability Act (HIPAA) require safeguards that protect and limit the use and disclosure of a user's data within the health industry [11].

This recent increase in data usage and cybersecurity threats, coupled with laws requiring protection against these threats, highlights the substantial need for data anonymization methods to preserve users' privacy

1.5 Milestones

In the course of formulating this thesis, a series of milestones were established to provide a structured framework for the progression of the research. These milestones are presented in Table 1.1

Table 1.1: Established milestones.

M1	First project plan handed in for peer review.
M2	Finished project plan delivered to supervisor.
M3	Find suitable encryption algorithms for implementation.
M4	Finishing chapter 1-3 of the report.
M5	Finish implementation of encryption methods.
M6	Conduct performance experiments on the encryption algorithms.
M7	Finish implementation of access control and role-based decryption.
M8	Conduct testing of implementation done in M7.
M9	Finishing chapter 4-6 of the report.
M10	Finishing chapter 7-8 of the report.

1.6 Target group

The target group for this work is primarily the researchers at EdTechLnu. However, it could also be interesting to other developers in the field of computer security. Especially people that work with cryptography and data storage. Additionally, it can be of further interest to schools with similar problems. That is, to ensure student confidentiality.

1.7 Outline

This report consists of a total of 8 chapters, including the introduction. The following Chapter will introduce and describe the theory of important topics that are relevant to the work done in this thesis. It will in detail explains topics needed to understand the

problem of investigation. As well as methods and theories used to solve the problem. The chapter is divided into subchapters where each subchapter investigates a different topic or theory. Chapter 3 describes the methodology used to investigate the problem of this thesis. It motivates the choices of methodologies made to solve the problems of this thesis. It discusses issues concerning reliability and validity that can occur with the chosen methodologies when applied to the problem. In addition, the ethical aspects of the project are considered. Chapter 4 presents the software implementation of the solution to the problem of this work. It provides an overview of the implementation using class diagrams and flowcharts, but also detailed explanations of the source code. Chapter 5 describes the experimental setup and presents the results of the experiment. This includes the environment in which the experiment is carried out, such as hardware specifications, source code programming language, operating system, and experimental metrics. The results are presented using a combination of tables and graphs. Followed by Chapter 5 is an analysis of the results, Chapter 6. This chapter analyses the results of the experiments carried out. Conclusions are made and motivated by the data presented in the previous chapter along with statistical tests performed on the data. Chapter 7 discusses the presented results in the context of the problem formulation and if the problem, as described in Chapter 1.3 has been answered. Chapter 8 draws conclusions about the findings of this thesis and what the thesis has accomplished. It discusses different possible approaches that could have made for a better result as well as how future work can further improve the project.

2 Theory

This chapter lays the groundwork of this thesis by introducing and explaining the key theoretical concepts that are relevant to the research. It serves as a foundation for understanding the work carried out. The chapter begins with a discussion on data anonymization, explaining its importance in protecting the privacy of individuals disclosed by the data. It mentions various methods of achieving data anonymization, both cryptographic and non-cryptographic. Additionally, this chapter provides a comprehensive examination of important cryptographic concepts, including symmetric and asymmetric encryption, along with a description of various encryption algorithms.

2.1 Data Anonymization

Data anonymization is a means to protect the privacy of individuals disclosed by the data. There are many methods of achieving data anonymization, both cryptographic and non-cryptographic. Depending on the purpose of the data, it is important that the right methods are deployed. Examples of cryptographic anonymization techniques are *Storage privacy, Privacy-preserving computations* and *Secure private communications* [12]. Storage privacy can be achieved through encryption techniques and access control. Privacy-preserving computations allow computations to be carried out on cipher text and can be achieved by homomorphic encryption. Secure private communications are done by encrypting the communications between users, thus preventing any third parties to intervene and read the data.

Another way to protect privacy is through k-anonymization, originally proposed by [13]. k-anonymization mainly focuses on the privacy preservation of publicly disclosed data to avoid linking attacks. The process involves applying operations such as suppression or generalization on cell values of the data in such a way that each record is indistinguishable from at least k - 1 others [14]. This thesis will further explore cryptographic techniques, namely storage privacy.

2.2 Cryptography

Cryptography is the practice of information security and has historically been used for political, religious, and military purposes to prevent opposing factions from intercepting valuable or sensitive information. It can be traced back to 400 BC, when it was used by Spartans [15]. However, in recent years cryptography is widely used to mask and protect any type of data stored on our devices and also to protect the confidentiality of digital messages sent via networks.

The topic of cryptography is built on several areas such as mathematics, theoretical computer science, number theory, finite field algebra, computational complexity, and logic. Combining these areas, we get modern cryptography [1].

The concept cryptography is based on the transformation of plaintext to ciphertext. Plaintext is the text that has not yet been encrypted. Once the plaintext has been processed by an encryption algorithm, it is then called ciphertext, which is the encrypted version of the plaintext [1]. For this purpose, there exist multiple conventional encryption algorithms and methods, all of which use cryptographic keys in the process. The keys used for this purpose are either symmetric or asymmetric. To return the ciphertext to plaintext, decryption algorithms are used and are the exact opposite of encryption, using either the symmetric or asymmetric key to perform the operation [16].

Cryptographic keys are bitstrings consisting of random letters, numbers, and signs. They often range in size from 56 to 2048 or more bits per key, depending on the cryptographic algorithm. See Figure 2.1 [1] for a simplified example of the encryption, and decryption process.

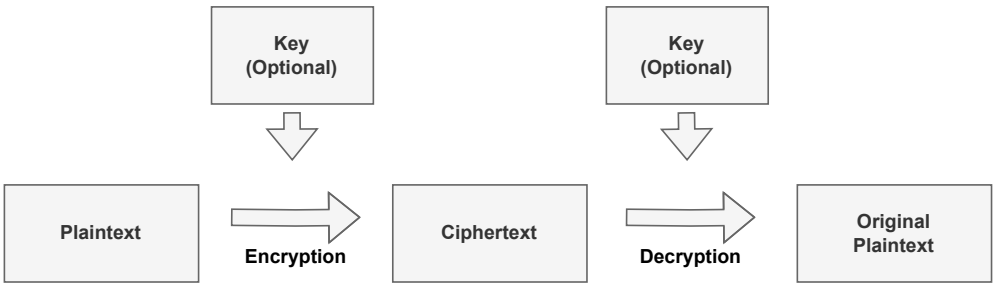

Figure 2.1: Cryptographic process [1].

In addition, some block algorithms apply padding, such as AES and Camellia. Padding is the process of adding extra characters to a block of plaintext when it is smaller or larger than the algorithm's block size. This additional padding ensures that the plaintext reaches an even block size.

2.3 Cryptanalysis

The practice of cryptanalysis involves deciphering ciphertext when the corresponding key is unknown. This is achieved by analyzing the encrypted data or transmission for patterns and clues, or by attempting various known attacks on the encrypted data to employ a brute-force approach to uncover the secret key [1].

Analysts look for any kind of pattern and often prefer to have a large amount of data to analyze. The most preferable case is to have both plaintext and ciphertext so that they can be compared and see what kind of transformation that has been used. An instance of this could involve intercepted messages featuring a known template, such as signatures found in official statements, which can subsequently be employed in cryptanalysis [1].

Cryptanalysis finds applications in various domains, including military operations where it is used to decipher transmissions from enemy forces. Additionally, hackers may leverage cryptanalysis to acquire confidential information, thereby exploiting it for malicious ends.

2.4 Symmetric Encryption

Symmetric encryption, also called single-key or secret key encryption, is when the key used to encrypt the plaintext into ciphertext is also used to decrypt the ciphertext back to plaintext. This key is known as a "secret key" and is shared amongst the parties involved. Symmetric encryption is usually the fastest way of encrypting and decrypting, but it is also less secure because the same key is used for both encryption and decryption. If the symmetric key is intercepted during communication the encryption can easily be cracked by third-party users, such as hackers. See Figure 2.2 [1] for a simplified example of symmetric cryptography where the same key is used for encryption and decryption.

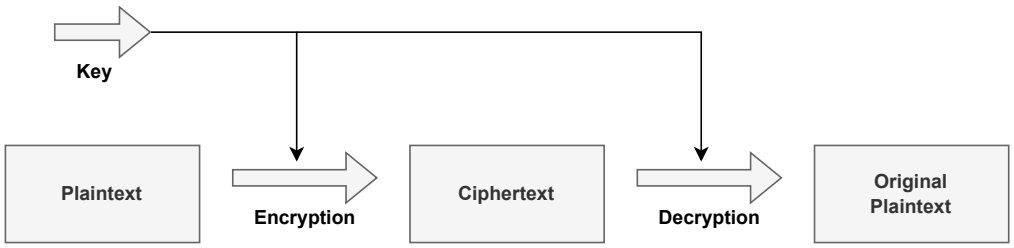

Figure 2.2: Process of symmetric encryption [1].

In [4], the symmetric encryption algorithms AES [16], 3DES, Blowfish and Twofish are compared in terms of encryption-decryption time and memory utilization. The author concludes that AES performed best in all categories. The implementation is done using C#. However, this conclusion is not shared by the author of [17] where another performance evaluation on four common symmetric encryption algorithms is conducted. These are DES, 3DES, AES, and Blowfish. The implementation is done using Java, which is motivated by its platform Independence, allowing for testing and comparison on a variety of platforms. The results show that Blowfish performed best in terms of speed, followed by DES, AES, and 3DES in that order. However, the authors highlight the importance of performance vs security trade-offs and that security should be considered first.

2.4.1 Advanced Encryption Standard

In 1997, the National Institute of Standards and Technology announced its intent to find a new encryption standard [18]. The goal was to develop a Federal Information Processing Standard (FIPS) capable of protecting sensitive information and used by the U.S Government as well as the private sector. Five finalist candidates were chosen: MARS, RC6, Rijndael, Serpent, and Twofish. After an extensive evaluation, the Rijndael algorithm, by Joan Daemen and Vincent Rijmen won the competition and became the new encryption algorithm of choice. Since then, the algorithm has become more known as the Advanced Encryption Standard (AES).

AES is a secure symmetric block cipher that processes 16 bytes (128 bits) at a time. It uses symmetric keys with a length of 128, 192, or 256 bits to encrypt and decrypt messages [19]. AES is widely used in many fields, as well as in cryptography protocols such as Socket Security Layer (SSL) [8].

In the encryption process of AES, the plaintext is divided into rows and columns, which are then manipulated in four steps: byte substitution, shift rows, mix columns, and add round key. These steps form a standard round and are repeated 10, 12, or 14 times, depending on the key size [19]. The byte substitution performs S-box transformation on the 16-byte matrix to mix the bytes from one to another. In the row shift transformation, the bytes are shifted cyclically left depending on the row number. In the third step, mix columns multiply each byte in a transformation matrix with each column of the state matrix. Each multiplication is then used in conjunction with an XOR to produce the new bytes. Add round key uses a subkey derived from the main key to combine each byte from the previous state with the byte of the subkey using XOR. This produces the final round

state that is used as input in the next standard round [8].

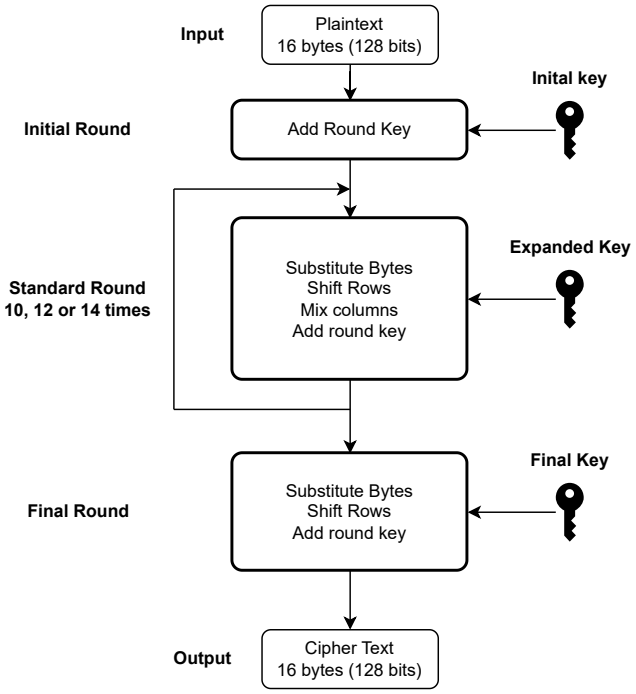

Figure 2.3: Encryption structure of AES.

The encryption process is described in Figure 2.3. First, the plaintext is manipulated by an add round key transformation in the initial round. Next follows the standard rounds containing all four transformations. Lastly, a final round is performed by doing the substitute byte, shift rows and add round key transformation which outputs the final cipher text.

2.4.2 Camellia

Camellia, developed by Nippon Telegraph and Telephone and Mitsubishi Electric Corporation in the year 2000, is a symmetric cryptographic block cipher method that applies permutation, diffusion, and avalanche effects [20]. Similar to AES, Camellia can accept key sizes of 128, 192, or 256 bits. For 128-bit encryption, it applies 18 Feistel rounds, and 24 Feistel rounds for 192 and 256-bit keys. Each iteration in Camellia is safe and effective in both software and hardware implementations and widely used over the internet and in applications [21]. Camellia provides strong protection against both differential and linear cryptoanalysis. Furthermore, it has proven to be resilient against attacks such as interpolation, related-key, truncated differential, boomerang, and slide attacks. Camellia substitution tables are 8-by-8-bit, making the algorithm optimal for multiple platforms with a variety of processors with varying performance, in addition to having low RAM requirements.

When performing encryption using 128-bit keys, the plaintext undergoes an XOR operation and is then split into two 64-bit blocks. Feistel, Substitution, and Permutation functions are then applied for all rounds except rounds 6 and 12, which instead inserts an FL function to prevent regularity. The FL function is a logical transformation that performs input and output whitenings on the text. The process for 192 and 256-bit keys

is the same but with one added FL round at round 18. Finally, the result is XORed down to a 128-bit ciphertext [22].

2.5 Asymmetric Encryption

In asymmetric encryption, a key pair is used, commonly referred to as the public and private keys. In this method, one of the keys is assigned to be the public key. The public key could be any of the keys in the pair and it is only important to keep the private key private. The public key is used for encryption while the private key is used for decryption. This technique is most often used when you do not want to risk sending the symmetric encryption over a network and risking interception by a third party. Instead, one of the parties generates a public-private key pair and sends the public key over the network. The receiver then uses this key to encrypt the plaintext and returns it to the sender. This ciphertext can then only be decrypted by the private key, which is held by the party that generated the key pair and is not shared. This is called a cryptographic key exchange [16]. See Figure 2.4 [1], for a simplified example of asymmetric cryptography where different keys are used for encryption and decryption.

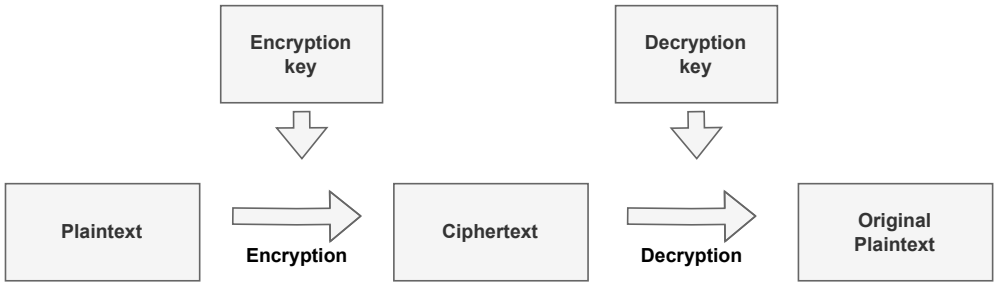

Figure 2.4: Process of asymmetrical cryptography [1].

Rivest-Shamir-Adelman (RSA) is an asymmetric public-key cryptography method used for secure transactions over the internet [23]. The encryption methods using the RSA algorithm are performed by dividing the plaintext into blocks, then performing a mathematical exponential computation on each of them by raising them to a power. The power depends on the key. Since exponentiation can scale up to larger numbers, RSA is considered heavily time-consuming, up to 10 000 times slower than symmetrical methods such as DES or AES. There is no known cryptoanalytic attack that is significant to RSA [1].

When sharing asymmetrical encryption keys over a network, there is a risk of interception. Therefore, it is necessary to employ cryptographic key exchange methods that significantly prevent attackers from intercepting the keys. These methods are typically, if not always, asymmetrical, where the secret key is transmitted in an encrypted form once a public-private key relationship has been established. Examples of such methods include Rivest-Shamir-Adelman (RSA), Hypertext Transfer Protocol Secure (HTTPS), Secure Socket Layer (SSL), and Transport Layer Security (TLS). [24].

2.5.1 Secure Transfer

HTTPS is a technology that combines either SSL or TLS with additional certificate mechanisms to provide secure and confidential communication over the web. In the first steps of HTTPS, secure communication is created by using a public-private key. Once the communication is secured, the data is then sent, using AES encryption. HTTPS is considered secure by most, but websites with outdated certificates are vulnerable to attacks, such as Man-In-The-Middle (MITM). In most cases, the user is warned by the web browser and hence, most incidents are caused by the user ignoring this warning [25].

SSL is an encryption protocol that helps to secure data exchanged between the user's web browser and server. It uses a mix of four protocols which strengthens upper layer protocols [26]. By doing that, SSL 3.0 can provide confidentiality, authenticity, and replay protection, if sent via TCP. In addition, SSL is also responsible for the initialization of the key-exchange protocol used for sending sensitive data [27].

TLS is the successor of SSL, with additional features and changes in security parameters. But both are still commonly used [24] [26].

2.6 Role-based Encryption

Role-Based Encryption (RBE) is a hierarchical encryption approach that classifies users into a hierarchy of roles and layers. Each role corresponds to a specific layer, forming a top-down pyramid structure. Roles positioned at the top of the pyramid possess access to all resources and can grant access to roles situated beneath them. Roles at lower levels can only access data available within their layer or any lower layers. Users are assigned specific roles with varying levels of access rights. Each role is associated with a key(s) that allows access to particular encrypted data based on the user's role. Users are authorized to both encrypt and decrypt data accessible to their designated role(s). RBE builds upon the concept of Role-Based Access Control (RBAC) while incorporating encryption. Consequently, only users possessing the appropriate role can encrypt and access data associated with that permission level [28].

2.7 Client-server

Client-server refers to a computing architecture in which tasks are divided between a client (user) and a server (backend) that communicate over a network using standardized protocols. [29]. This enables multiple clients to connect to a server or database simultaneously. The most common approach to client-server data processing is to put this responsibility on the server. This means that the clients send requests to the server, whilst the server returns the results of those requests back to the clients. This speeds up the performance of the system since the clients can then further calculate the given result inside the client-hosted applications, saving performance from the server. It also reduces data replication since there is one common storage where the final results are saved [30].

Encryption can be performed on either the server or client side, and it can be implemented separately or on both entities. When encryption is implemented separately, it is called either server-side encryption or client-side encryption, depending on which entity is responsible for performing the encryption [24].

2.8 Access Control

Access control is a mechanism responsible for implementing access rights for users. Users that want to access certain data or applications must be approved by this mechanism if it is implemented within the system. User identification is most often performed using a username and password [24]. It can also be done using services such as role-based access control (RBAC), attribute-based access control (ABAC), or open authorization (OAuth).

Role-based access control is a mechanism that divides users into roles, each role with a set of privileges [31]. Each user is assigned one or more roles, which are then used for accessing data or other resources. Before a user is assigned any role, a system administrator must first authorize the user as a member of the organization. It is most commonly used within organizations due to the complexity of its implementation. One of the key concepts within RBAC is to give the users no more privileges than necessary [32]. Additionally, in larger systems, additional security measures such as Separation of duties (SOD) are applied, preventing single users from being able to commit fraud. This is called constrained RBAC. In addition, RBAC can be implemented as a flat or hierarchical approach, to either split or inherit role permissions [33]. Due to security concerns, it is typically preferable to establish a new role when existing roles are unsuitable, rather than giving users excessive privileges. But excessive roles can also pose a security threat. RBAC is a powerful tool that provides great flexibility and security, but only if well-planned [32].

Attribute-based access control is rather similar to RBAC. Instead of assigning users to roles, users are instead assigned attributes, used for boolean auditing. ABAC is simpler than RBAC to implement due to the set-up of RBAC roles demanding a lot of planning. But, if the amount of attributes grows large, ABAC could be difficult to manage. ABAC can also adjust after real-time environmental states when auditing. Examples of attributes can be user GEO location, time of day, or even a user role. In addition, attributes can also be user actions such as: read, write, delete, add. This provides the AC system with further flexibility [33].

2.8.1 Open Authentication 2.0

Open Authentication (OAuth) is an authorization standard with the primary purpose of authorizing API-calls from third-party applications. OAuth uses long-term access tokens for authorization. OAuth implements four roles: resource owner, resource server, client, and authorization server. Each with a responsibility to deliver requested data to a third party after client verification [1].

3 Method

This chapter describes and motivates the choices of methods used to answer the problem formulated in Section 1.3. An overview of the chosen methods is provided, as well as how they are planned to be used in this thesis. Reliability, validity, and ethics are also discussed.

3.1 Research Project

Methods used in this study are well-known methods such as literature review (LR), verification and validation (V&V), and controlled experiments (CE). To acquire comprehensive knowledge and understanding, RQ1 employs LR to explore additional research within the subject and appropriate methods for implementation. Different techniques and approaches to the problem are examined and evaluated using fact-checking against books and peer-reviewed articles published from 1990 onwards. The year 1990 is selected due to the lack of change in the methods in this field since that time. Encryption is a complex topic and its next significant transformation is likely to arise from quantum computers [34]. Peer-reviewed articles are preferred due to their critical evaluation by experts, consequently providing greater evidentiary weight compared to non-peer-reviewed articles.

To answer RQ2, an implementation of encryption algorithms is made through the use of Python and the Python-based library Cryptography [35]. Python was selected at the request of the stakeholders at EdTechLnu and is well-suited for the project due to its cryptographic libraries. The study does not measure file transfer execution time because the focus lies only on the time of execution of the cryptographic techniques. Hence, a database is not required to perform the tests, and data, such as plain text and cipher text, will be written to and stored in .txt files. In a live, real-time environment, a database would be necessary to host and manage the data. But for this experiment, it is redundant and disconnected from the cryptographic execution time.

To evaluate the performance of the encryption algorithms, as formulated in RQ2, CE will be conducted to measure the dependent variables and compare the outcomes. The outcome shall be documented so that it can be used in performance graphs. The execution time of encryption and decryption and the data sample size of encrypted data will be measured for different sizes in data sets. Data sets will consist of synthetic data with varying sizes during the CE, taking the mean value of the execution time of n runs. The results will be plotted to further investigate the growth of the data.

To answer RQ3, the V&V method will be deployed to test the functional requirements in the implementation. A list of functional requirements will be set and tested, using unit tests, so that the software fulfills them accordingly.

3.2 Literature review

A literature review is implemented to find and narrow down suitable methods to be used in the controlled experiment. The review will be performed solely by searching for, and reading, peer-reviewed articles from 1990 and onwards. The year 1990 is motivated by the lack of change in the industry after this year. When searching for suitable articles, two search engines are used: Linnaeus University online library OneSearch [36] and Google Scholar [37].

To find suitable articles that are relevant, search words suited for the area of cryptography are used, such as Cryptography, Encryption, Decryption, Symmetric encryption,

Asymmetric encryption, multi-key cryptography, and Role-based encryption. When using these search words, it is expected to find information to locate suitable algorithms that fit the specific purpose of this research. It is possible that by excluding non-peer-reviewed articles, there could exist faster and more secure cryptographic algorithms that are not included in the review. But, by only including peer-reviewed articles it is known that they have been cross-checked for faults and are not factually incorrect. In addition, using different search words could generate a different result, but the selected search words matches the research group at EDTECHLnu's request and hence used.

3.3 Controlled Experiment

Experiments are often carried out to test a certain process or method. The purpose is to establish a foundation that can explain, describe or further explore a given hypothesis. In a controlled experiment, quantitative data are gathered systematically in a controlled environment, with the purpose of ruling out factors such as randomness. This requires a definition of independent and dependent variables. The independent variables are inputs that affect the results while the dependent variables are the measured results themselves. All variables except for the independent variable should be kept constant to try to minimize the impact they could have on the dependent variable [38].

In this thesis, controlled experiments will be applied by measuring three different dependent variables:

1. Encryption time (ms).
2. Decryption time (ms).
3. Cipher text size (bytes).

The experiment will be conducted on the encryption algorithms identified by the limited literature search conducted. The independent variables are the input data set and the different encryption algorithms. A total of three different-sized data sets are to be used. Hence, the experiment will be repeated three times for every dependent variable. The test will be iterated 1000 times for each data set. Data will be gathered from each iteration, and a mean of the results of all the combined iterations will be calculated. This is done to further validate the results and reduce the impact of any possible disturbances that can cause a variation in the results.

The different data sets will be generated using a Python script. The data that are to be encrypted are unique identifiers. Hence, random identifiers of the following format will be generated:

$$xx123yy$$

That is, the first two letters will be randomly generated. Followed by three random digits, followed again by two more random letters. Three separate data sets will be created. The first, *d1* contains a total of 1000 entries, separated into 25 different school classes, with a total size of 9KB. The second, *d2* contains 2000 entries, separated into 50 classes with a total size of 18KB. The third data set, *d3* contains 4000 entries, separated into 100 classes, with a total size of 36KB.

3.4 Verification and Validation

Verification and Validation are used to determine whether an implemented software meets its requirements and specifications to ensure that the intended purpose is fulfilled. The

methodology consists of two phases. The verification phase tests an implementation for all the requirements. This can be achieved using manual or automated tests. The validation phase is used to check whether the implementation performs in alignment with the expectations of a stakeholder, such as a customer or an end user.

In this work, some specific requirements for the software implementation have been proposed by the stakeholders. These are:

- R1: The system shall be able to encrypt and decrypt a data set of unique identifiers.
- R2: The system shall enable different levels of decryption based on user permissions.
- R3: The system shall be able to revoke a user's ability to decrypt data without affecting the encrypted data.

These requirements will be verified by automated tests using the *unittest* [39] framework in Python. This is done by defining one or more test cases for each requirement. The test will be carried out by creating a test environment that replicates a scenario of the system being used by a school. This implies creating a set of user objects, including one head principal and several class principles. To test and verify the R1, R2 and R3 requirements, the following test will be performed:

3.4.1 R1 unittests

- T1: This test tries to encrypt a class key using the master key. The resulting ciphertext is compared to the key in plaintext. The expected outcome of the test is that the objects differ.
- T2: Decrypts the encrypted class key using the master key. The results of the decryption are compared to the original class key in plaintext. The expected result is that the objects are equal.
- T3: Tries to encrypt a string replicating the unique id of a student in the data set, using the class key. The result of the encryption is compared to the original plaintext. The expected result of the test is that the two objects differ.
- T4: Decrypts the encrypted unique id. The result is compared to the original id in plaintext. The expected result is that the two compared objects are equal.

3.4.2 R2 unittests

- T5: This test tries to assign a new role to a user. The users' role privileges before invoking the set_role method are compared to the users' roles after. The expected result is that the compared objects are not equal.
- T6: Tests the revoke access functionality given by the $revoke\_role$ method. The test compares the user object before invoking the method to the object after invoking the method. The expected result of this test is that the objects differ.
- T7: Tests the $access\_control$ function to see if a user has access to a given class. The expected result is a boolean that equals true.

- T8: Tests the $access\_control$ function for a principal user. The expected result is a boolean that equals true.
- T9: Tests the $access\_control$ function for a revoked role of a user. The expected result is a boolean that equals false.

3.4.3 R3 unittest

- T10: This test compares ciphertext before and after revoking user access to demonstrate that access control does not require re-encryption. The expected result is that the ciphertext before revoking access equals the ciphertext after revoking access.

3.5 Reliability and Validity

In this study, reliability is difficult due to being hardware dependent on the related specifications used when performing the tests. The current state of the operating system while running the tests, plus the OS version is also an important factor. In this report, the hardware specifications are specified, but replicating these tests, a slight variation in execution time should be expected due to these reasons if a different computer model is used. The modifications made in the cryptographic library used in this study are documented. Changing them could lead to larger variations in execution time and slight differences in data storage.

The code used in the tests is designed and implemented by the writers of this thesis. The time of execution is measured using the Python library named $time$. The function $time.perf\_counter\_ns()$, is called just before encryption starts and saved in a variable named "start", and then called again directly after the encryption is done, saved into a variable named "end". The runtime is then calculated by subtracting the variables $result = end - start$, which gives the runtime in nanoseconds, dividing the resulting runtime by 1 000 000 gives us the desired result in milliseconds. This is repeated for the decryption. The time of execution for encryption and decryption is then saved separately.

In consideration of validation in this study, the tests are performed on a stationary Windows 10 PC, without considering the current state of OS, see chapter 5.1 for specifications. The choice of OS, version of current OS version, or OS settings may also affect the tests in an unforeseeable way but are not taken into consideration during the tests. The tests are based on Python 3.11.3 code designed for this purpose. The tests are repeated 1000 times per data set for increased accuracy.

3.6 Ethical Considerations

No real data is used in this thesis. Implementations and results are displayed with full transparency. Hence, no ethical dilemmas have been identified in this thesis.

4 Implementation

This chapter provides a detailed description of the software implemented in this work. The chapter is divided into several sections. The first section discusses the choice of encryption algorithms, in regard to RQ1. The second section presents the implementation of encryption algorithms and the controlled experiment carried out to test those algorithms, in regard to RQ2. Parts of the implemented code can be viewed in the appendix A. The third section presents an additional implementation of an access control mechanism, which mainly relates to RQ3.

4.1 Choice of Encryption Algorithms

This section will discuss the findings of the literature review, regarding RQ1. That is, to find suitable encryption algorithms for the unique identifiers in the student data set. The findings of the literature showed that asymmetric encryption algorithms, such as RSA perform considerably slower than symmetric encryption. Asymmetric encryption is often used for creating secure transactions over the internet and encrypting small pieces of information, such as passwords. Hence, concerning encryption of the student data set, asymmetric encryption was disregarded as a suitable alternative. Looking at symmetric encryption algorithms, previous work showed that AES outperforms most of the competition in terms of computational speed. The literature review also demonstrated that it is one of the safest and fastest symmetric algorithms widely used today. Additionally, the literature search also indicated that the Camellia algorithm is a time-effective and attack-resilient cryptographic algorithm with a performance similar to AES. In addition, both AES and Camellia are two out of three block ciphers included in the ISO/IEC 18033-3:2005 standard [40]. Furthermore, the literature study showed limited results in comparative performance evaluations between AES and Camellia. Thus, this study could also help fill that gap. Therefore, the cryptographic algorithms that will be further examined in this work are AES and Camellia, they are considered suitable for encrypting the student data set because they are both symmetric encryption algorithms that show great promise in terms of encryption speed and security, both also conform to the recommendations of symmetric block ciphers in the ISO/IEC 18033-3:2005 standard. The algorithms are imported from the Python library Cryptography, a well maintained and updated cryptographic library recommended by Google [35][41].

4.2 Experimental Implementation

To conduct the experiment, a software in Python was developed. The main purpose of this software is to generate unique randomized strings consisting of four letters and three numbers, and then subject them to encryption and decryption algorithms. Timing the cryptographic execution and saving the ciphertext results in .txt format to compare with plaintext data storage difference in Kb. Figure 4.5 displays the flow of the cryptographic execution.

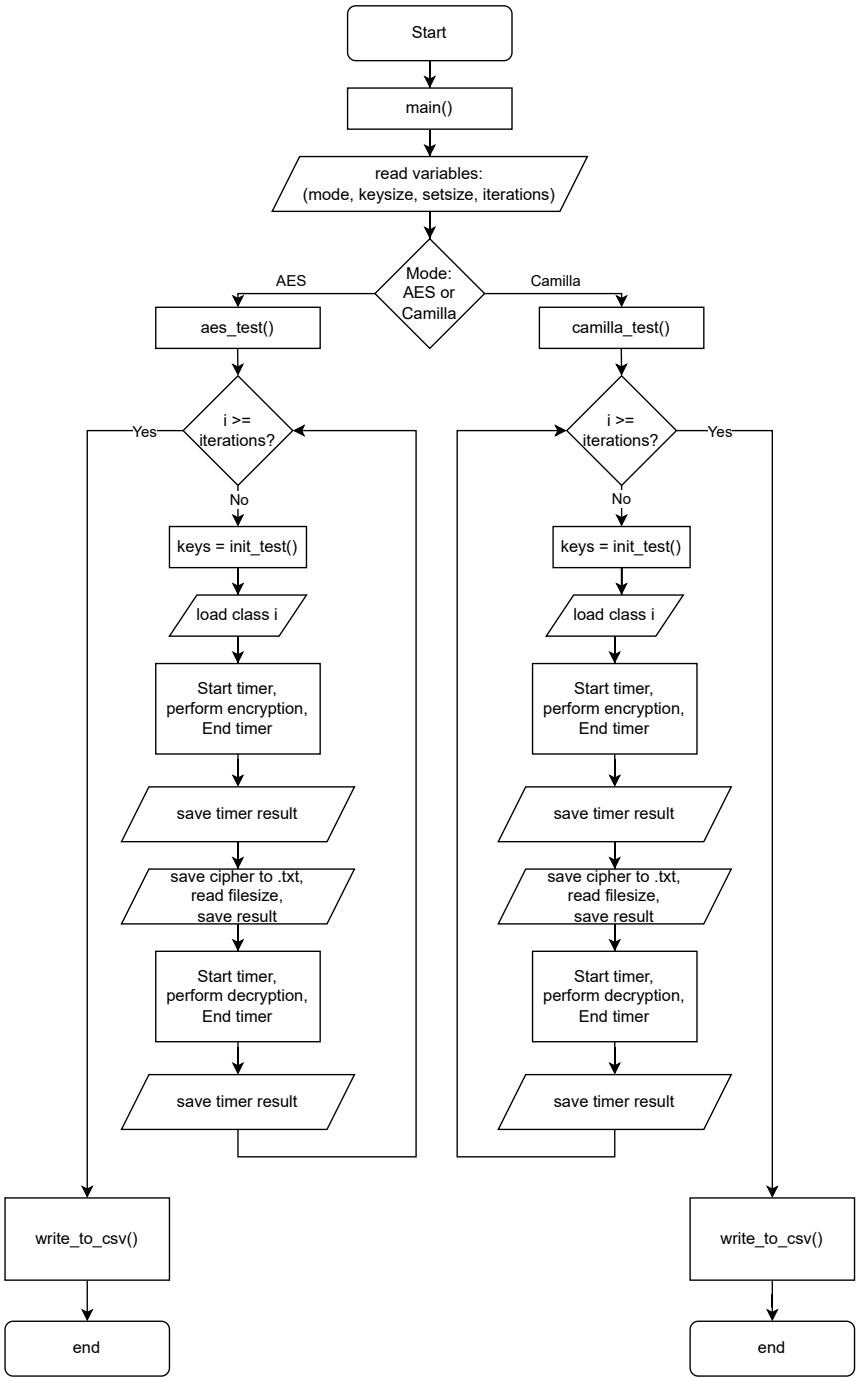

Figure 4.5: Experimental flow chart.

The unique randomized strings are created by running a script that randomizes letters and numbers, adds them to a set to ensure the unique characteristics, and loops until the set has the right amount of unique strings in the desired form of xxYYYxx, where x represents a letter from A to Z, and Y represents a number from 0 to 9. This script is run once, and the results are saved into .txt files, splitting the set into pieces of 40. The text files are also generated with a script, creating Z amount, all named "class1". The ending 1-Z represents which class that file represents, the .txt files are then saved in a folder called $database$. In this experiment, $40 * 100$ unique identifiers were created, *d1* consists of classes 1 to 25, *d2* of 1 to 50, and *d3* 1 to 100. For testing, four Python scripts was implemented: $main.py$, $key\_manager.py$, $aes\_algo.py$, and $camellia\_algo.py$. The class representations are visible in Figure 4.6.

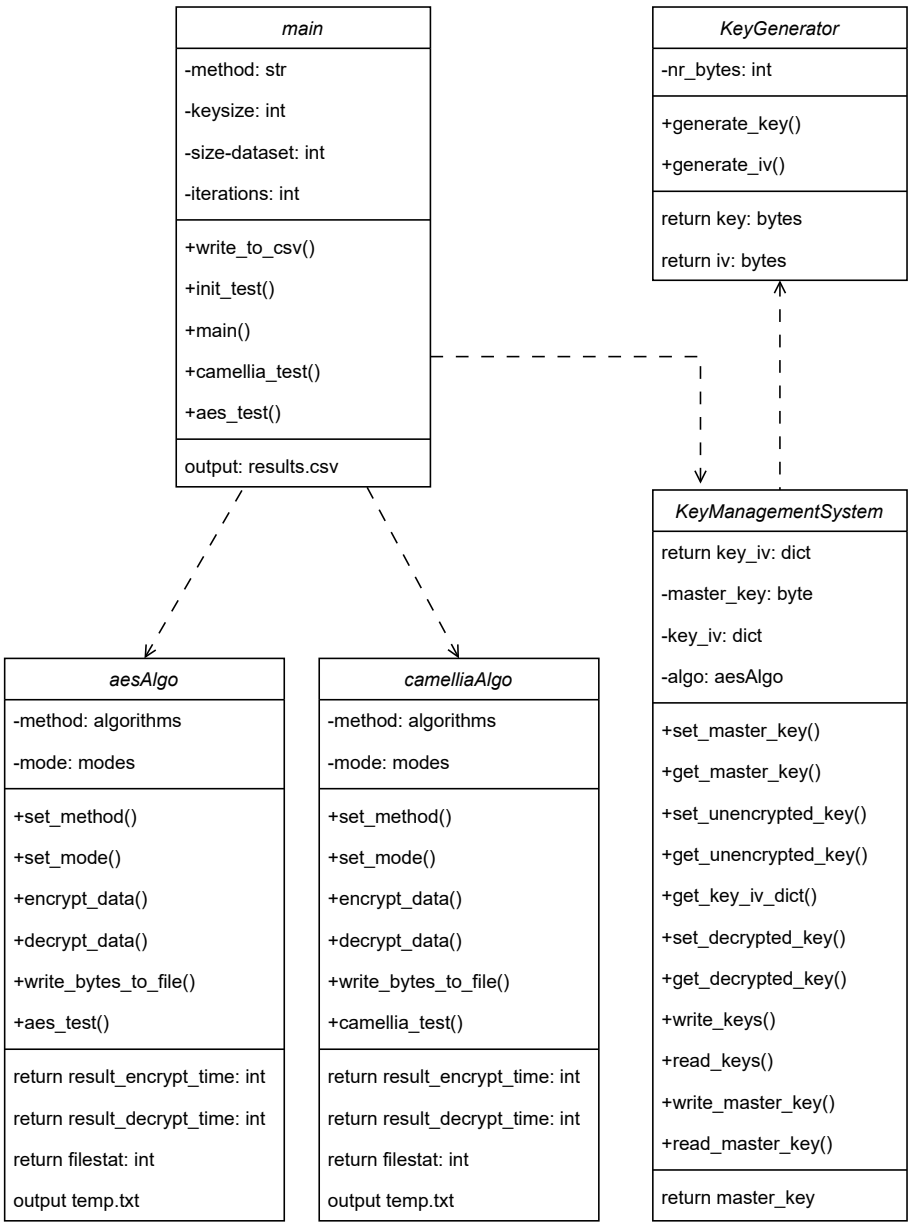

Figure 4.6: Class diagram of the software.

The main class, "main.py" is used to control the experiment inputs, such as key size, initial vector size, and the type of algorithm to be executed. It has the following functions: $main$, $write\_to\_csv$, $init\_test$, aes_test and $camellia\_test$. Main takes no arguments and consists of three tests with the following data sets: *d1*, *d2*, and *d3*. The tests are constructed in the form of aes_test("aes128", 16, 25, 1000), where "aes128" is the cryptographic method, 16 is the key size in bytes, 25 represents the number of data sets to be pre-loaded, and 1000 is the number of iterations of this test to be run. This is passed into aes_test or $camellia\_test$, depending on the algorithm. This is repeated three times with different sizes of data sets: 25, 50, and 100, previously referred to as *d1*, *d2*, and *d3*. The results are appended to a .csv file after each run so that the results can be analyzed.

Method named $init\_test$ generates pairs of keys and IVs and then saves them to a dictionary that is passed into the test classes. The amount of generated pairs equals the size of the data set. aes_test and $camellia\_test$ methods are two nearly identical methods for running the tests. Each consists of a for loop that runs as many times as the size of the data set. After each iteration, the mode used, encryption time, decryption time, and size of the ciphertext are saved in lists. The lists are then sent to $write\_to\_csv$ to be saved in the .csv file. $write\_to\_csv$ takes the result from the test classes and appends the results to a .csv file with headers $[" method", "enctime", "dectime", "size"]$.

The classes aes_algo.py and camellia_algo.py is responsible for performing the tests and creating the experimental values, such as the time and size of the cipher. Both classes are nearly identical, with small differences depending on the choice of method. Both classes have the following methods: set_method, set_mode, $encrypt\_data$, $decrypt\_data$ and $write\_byte\_to\_file$, as well as a function for testing named after the method, aes_test or $camellia\_test$. For cryptographic operations, Cipher, algorithms, and modes are imported from $cryptography.hazmat.primitives.ciphers$, used together with padding from $cryptography.hazmat.primitives$.

The set_method and set_mode functions, assign method and mode to the object depending on the method passed as input argument. The method and mode assigned are used on the first row inside $encrypt\_data$, $decrypt\_data$. $write\_byte\_to\_file$ is used to save the result from tests performed in aes_test or $camilla\_test$, to a .txt file so that the cipher size can be measured.

The function takes in a key-iv pair dictionary and the number of datasets as arguments. It then collects the respective key and iv for each class in the datasets and runs each unique ID in class$n$.txt through the encryption and decryption algorithms line by line. The time of execution is only measured before and after the call for encryption and decryption, and only measures the padding and the cryptographic operation. The ciphertext produced comes out as a binary string which is converted to a regular string by applying $.hex()$ on the variable holding the ciphertext. The conversion is performed so that before/after comparisons can be done on the same datatype, a string. When all files in the data set have been put through the algorithm, they are saved in a temporary .txt file using $os.stat()$ to measure the cipher size in bytes.

In $encrypt\_data$, a cipher object is created using the Cipher library by passing in the encryption method along with a key and an iv. Since all algorithms used in this experiment have block sizes of 128 bits, a fixed padding of size 128 is used. In $decrypt\_data$, the inverse operation is applied.

The $key\_manager$ class is used to create and handle the key-iv pair dictionaries. Each data set has its own pair of key-iv, which is stored as an object and passed on to the test functions.

In the class $KeyGenerator$, the key and iv are generated in two separate methods.

They both take nr_bytes as an argument which is the key size in bit, 16 or 32. They are called from $init\_test$ and then added to the key-iv dictionary in $key\_manager$.

Method $set\_unencrypted\_key$, called from $main.py$ is used to generate the key dictionary used in the experiment. It iterates n amount of times where n represents a school class, numbered between 1 to 100. When conducting the experiment, unencrypted keys are used because only encryption and decryption of the identifiers are of interest. Encrypting the keys using a master key, simply adds an additional layer of security when storing the keys.

4.3 Additional Implementation

An additional class named $roleController$ was implemented to test and ensure role-based access control. It creates user objects connected to teachers' unique IDs and stores user access rights. It then compares assigned roles to the requested school class when a request is made. In addition, $keyManager$ was extended so that it can replicate a light version of a Key management system. This was done by adding additional methods to the class $keyManager$ to properly support the access control solution and now handles the storage, creation, and distribution of keys. The methods implemented in $keyManager$ create and handle encrypted keys, encrypted with the master key. The master key is a single key created just as any other key but saved outside, separate from the key dictionary, and used to encrypt the school class keys for an additional layer of security.

The master key is used in the unit tests that concern the access control for the methods used to save and read the master key from the keyManager. The key is collected if the user has the correct role to access one of the school classes using the method $access\_control$.

The $access\_control$ method compares the roles assigned to the user with the name of the requested class. For example, a class principal can have a "class1" role and requests access to "class1", which is then considered a match.

If matched, $access\_control$ returns True and the master key can be collected. The master key is then used to decrypt the encrypted class key using $get\_decrypted\_key$, which is then used to decrypt the encrypted unique IDs of the requested school class.

5 Experimental Setup and Results

This chapter presents the empirical data gathered throughout the course of this work. The chapter begins with a description of the experimental setup. It provides the necessary context for understanding the experimental process and the conditions that may influence the results. Subsequent to the setup follows three subsections, each dedicated to presenting the results of the controlled experiment conducted on the dependent variables: encryption time, decryption time, and cipher text size. Additionally, the chapter incorporates a section dedicated to the verification and validation of the software implemented. The section presents the results of the automatic unit testing performed.

5.1 Experimental Setup

The experimental tests were conducted in a Windows 10 Pro N environment, specifically version 22H2 of the x64-bit OS with version number 19045.2846. Before running the tests all external programs were terminated. The tests were executed through the Windows CMD window by calling the main function, "main.py," from within the project folder using the command "python main.py." To prevent any interruptions, the computer was left alone during the tests, and power saving mode was disabled. The code for the experiments was written in Python version 3.11.3 and was developed using the cryptography library, which includes the cipher, algorithms, modes, and padding modules. Visual Studio Code version 1.78.0 was used to write all the code.

The experiments were performed on a computer with the following hardware specifications: an Intel(R) Core(TM) i7-8700K CPU clocked at 3.70 GHz, 16 GB of Kingston DDR4 RAM with a speed of 2400 MHz, and a Samsung SSD hard drive with a maximum reading/writing speed of 560/530 MBs. Additionally, the computer was equipped with an NVIDIA GeForce RTX 3070 GPU.

5.2 Encryption Time

This section presents the results of the encryption experiment conducted. Figure 5.7 displays a bar chart with the result of the measured encryption time using the two different encryption algorithms AES and Camellia with key sizes 128 and 256 bits. Encryption time is plotted on the y-axis in milliseconds. The different data sets and their sizes are plotted on the x-axis.

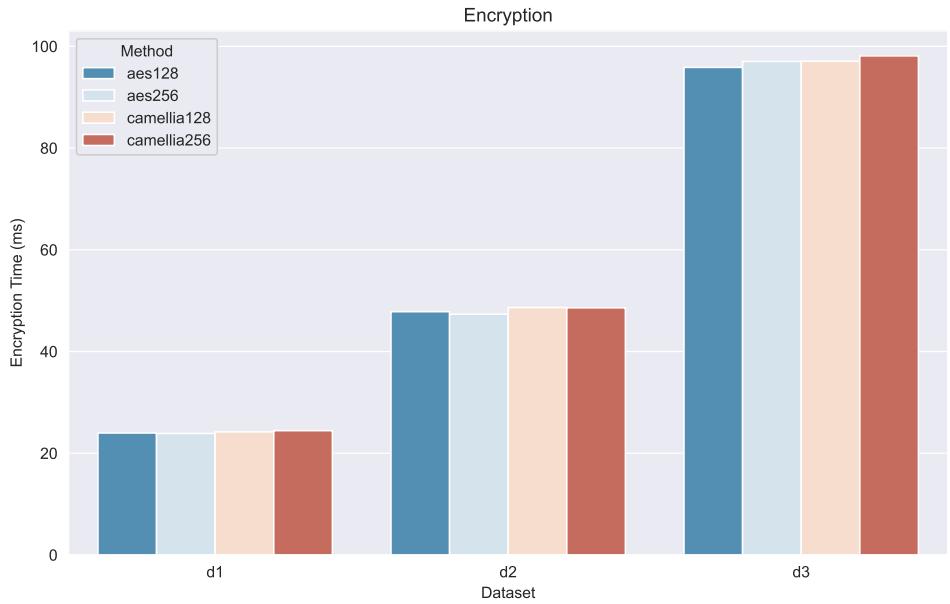

Figure 5.7: Encryption time of the different algorithms and key sizes plotted against each data set in a barplot.

Table 5.2 displays the same data as Figure 5.7. That is, the data gathered in the encryption experiment. The mean encryption time over the 1000 iterations has been calculated, along with the standard deviation of that data. The values are displayed rounded to two decimal points. The first column specifies which algorithm and key size are used. The second column specifies which data set. The third and fourth columns are the mean and standard deviation in milliseconds. The raw data from all conducted experiments can be viewed in Appendix A or handed out upon request.

Table 5.2: Mean and standard deviation of encryption time. Boldface indicates the best-performing algorithm.

Algorithm	Data set	Mean (ms)	Standard Deviation (ms)
AES128	d1	23.99	2.27
	d2	47.84	2.36
	d3	**95.87**	2.46
AES256	d1	**23.89**	2.20
	d2	**47.35**	2.20
	d3	97.04	2.59
Camellia128	d1	24.20	2.29
	d2	48.65	2.30
	d3	97.08	2.30
Camellia256	d1	24.44	2.11
	d2	48.60	2.32
	d3	98.13	2.45

5.3 Decryption Time

This section presents the results of the controlled experiment with decryption time as the dependent variable. Figure 5.8 illustrates the decryption time over the different data sets and algorithms as a barplot, in the same manner as the encryption time in Figure 5.7.

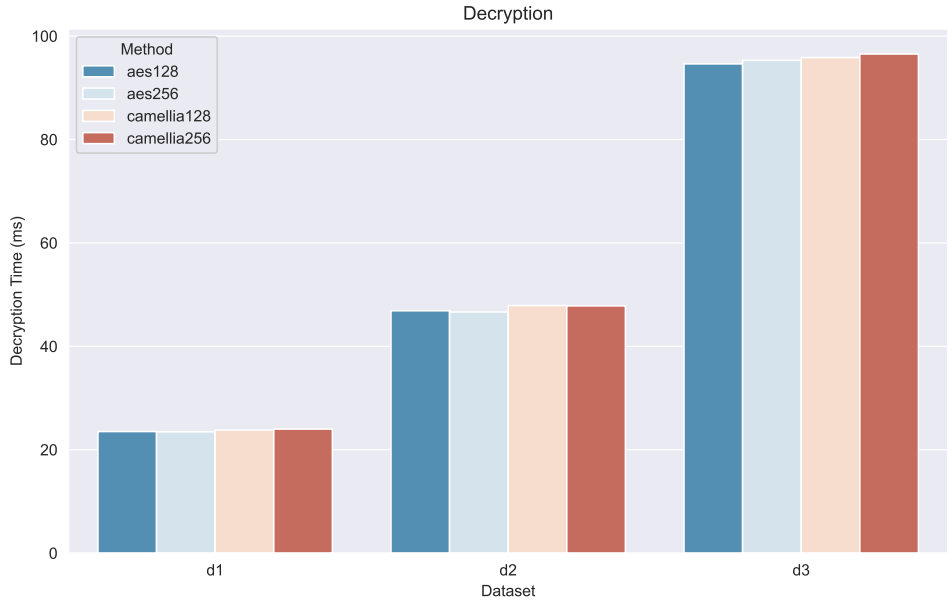

Figure 5.8: Decryption time of the different algorithms against each data.

Table 5.3 shows the calculated mean and standard deviation for each algorithm and data set over the 1000 iterations. The mean and standard deviation are presented in milliseconds.

Table 5.3: Mean and standard deviation of decryption time. Boldface indicates the best-performing algorithm.

Algorithm	Data set	Mean (ms)	Standard Deviation (ms)
AES128	d1	23.52	0.29
	d2	46.88	0.51
	d3	**94.63**	0.76
AES256	d1	**23.46**	0.18
	d2	**46.64**	0.30
	d3	95.34	1.19
Camellia128	d1	23.81	0.19
	d2	47.89	0.26
	d3	95.86	0.69
Camellia256	d1	23.99	0.19
	d2	47.82	0.43
	d3	96.54	1.08

5.4 Ciphertext Growth

This section provides the results of the controlled experiment with cipher text size as the dependent variable. Figure 5.9 displays the growth of the cyphertext size in the form of a barplot. The ciphertext size in kilobytes is plotted on the y-axis and the different data sets are plotted on the x-axis. The algorithms are separated by bars of different colors.

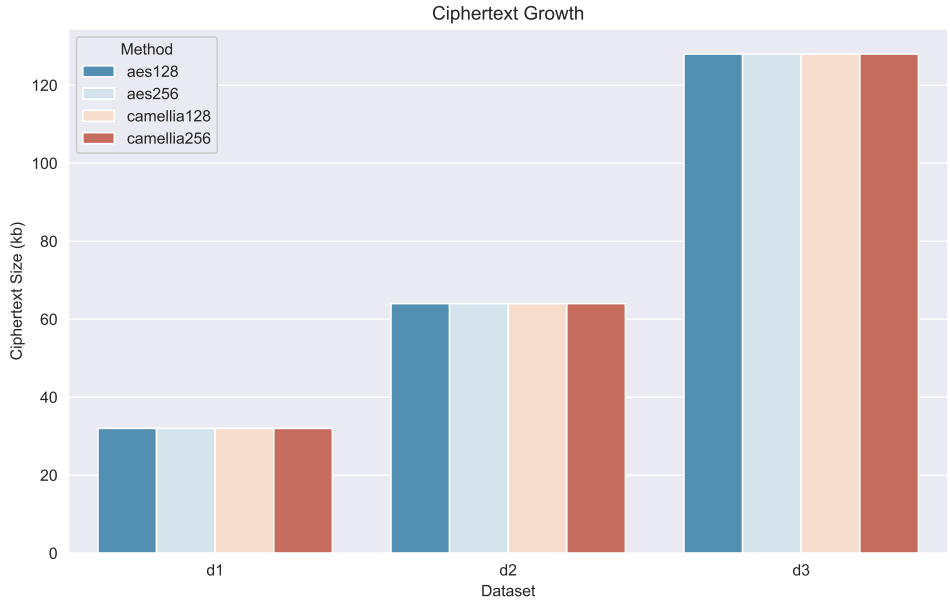

Figure 5.9: Growth of plain text after performing encryption on the different data sets.

Table 5.4 presents the numerical data on memory utilization gathered in the experiment. The first column specifies which algorithm is used. The second specifies how many characters that are to be encrypted in the test file. The third column specifies the size of the plaintext test file in bytes. The fourth column specifies the size of the encrypted test file in bytes.

Table 5.4: Memory utilization for the four encryption algorithms.

Algorithm	Characters in test file	Plaintext (bytes)	Ciphertext (bytes)
AES128	7000	9000	32000
	14000	18000	64000
	28000	36000	128000
AES256	7000	9000	32000
	14000	18000	64000
	28000	36000	128000
Camellia128	7000	9000	32000
	14000	18000	64000
	28000	36000	128000
Camellia256	7000	9000	32000
	14000	18000	64000
	28000	36000	128000

5.5 Verification and Validation

Performing test cases using $unittest$ resulted in all tests passing the requirements. The tests were conducted by running the test.py -v command in the command prompt inside the project folder of the source code. The fact that all tests pass the set requirements indicates that the developed software responds to the project's predefined requirements. Thus, the implementation passes the verification phase.

Table 5.5: Unittest results and type of test.

Test no.	Requirement	Type of test	Result
T1	R1	Cryptographic	Passed
T2	R1	Cryptographic	Passed
T3	R1	Cryptographic	Passed
T4	R1	Cryptographic	Passed
T5	R2	Access control	Passed
T6	R2	Access control	Passed
T7	R2	Access control	Passed
T8	R2	Access control	Passed
T9	R2	Access control	Passed
T10	R3	Data control	Passed

Results seen in Table 5.5 represent the outputs of the conducted unit tests and what type of test they belong to. Rows displaying R1 tested the cryptographic operations in the software, used for conducting the experiments. Rows displaying requirement R2 show the output of the tests that are responsible for testing role-based access to the encrypted unique identifiers. The result in the requirement row named R3 represents the output of a single test, responsible for role revocation without the need for re-encryption of the data for which the role is responsible.

6 Analysis

This analysis chapter will further analyze the empirical data gathered in the controlled experiment. The chapter is divided into subsections, each concerning one of the measured dependent variables. For encryption time and decryption time, the Kruskal-Wallis [42] test is performed to investigate the statistical significance between the algorithms. Additionally, the Shapiro-Wilk [43] test is performed to investigate the distribution of the results. Furthermore, the chapter draws conclusions based on the results of the statistical testing, as well as a discussion on the actual difference in effect size between the algorithms. Lastly, the ciphertext growth is further analyzed and reasoned about.

6.1 Statistical Analysis Setup

This section is dedicated to providing a foundation for the statistical analysis performed in Section 6.2 and 6.3, namely the Kruskal-Wallis test. The test objective is to check if the difference seen between the ranks of the algorithms reflects a significance, or if the difference is caused by random noise inside the groups.

The null hypothesis, H_0 is defined as follows:

$$H_0 : MR_1 = ... = MR_K$$

The alternative hypothesis, H_1 states the opposite:

$$H_1 : \neg(MR_1 = ... = MR_K)$$

Where MR is the mean rank, K is the group, or in this case the encryption algorithm. In simpler terms, the test examines whether choosing a value from any of the algorithms will have an equal chance of selecting the highest value. The test calculates a probabilistic value (p-value) which is compared to a significance level α. If the p-value is less than the significance level, H_0 is rejected. In addition to examining whether there is a significant difference between the group, the test will also perform multiple comparisons between the mean ranks of each group to determine which of the groups differ. For further explanation on the computations of the Kruskal-Wallis test, please refer to the original documentation provided in [42].

6.2 Encryption time

The data presented in 5.2 shows that the mean encryption time is similar across all algorithms. There is a slight advantage to the AES128 method on the largest data set, and AES256 on the smallest and middle-sized data set. However, the difference looks minor. To further explore whether there is any significant difference between the encryption time of the algorithms, some statistical analysis can be performed. First, the data are tested for normality using the Shapiro-Wilk test. This is done to determine which statistics can be used to examine the significance of the difference between the algorithms. Table 6.6 shows the results of the Shapiro-Wilk test on encryption time with the different algorithms and data sets. First, we note that all p-values produced by the test are very small, with the exception of the AES128 which has a p-value that equals 0. The statistics software used for the calculations rounds small p-value to 0. Hence, the value also indicates a very small number. With that in mind and using a significance level of 0.05, all tests reject the null hypothesis. Meaning there is strong evidence against the hypothesis that the data originate from a normal distribution.

Table 6.6: Shapiro-Wilk test for normality

Algorithm	Data set	P-value	Reject Null Hypothesis
AES128	d1	0.0	Yes
	d2	$1.10e-42$	Yes
	d3	$1.54e-21$	Yes
AES258	d1	$8.83e-25$	Yes
	d2	$5.03e-12$	Yes
	d3	$8.00-28$	Yes
Camellia128	d1	$4.30e-07$	Yes
	d2	$2.54e-14$	Yes
	d3	$6.35e-26$	Yes
Camellia256	d1	$5.45e-18$	Yes
	d2	$5.12e-35$	Yes
	d3	$9.78e-26$	Yes

Working with the hypothesis that the data are not normal-distributed, the Kruskal-Wallis test can be performed to analyze if there are any statistical differences between the algorithms. The test will also identify which pairs of methods are significantly different. The test is performed on the complete sample from the experiment, meaning the results from all 1000 iterations. The result of the Kruskal-Wallis test on the encryption times is presented in Table 6.7.

Table 6.7: Kruskal-Wallis statistical test on encryption time.

Data set	P-value	α	Reject Null Hypothesis	Pairs
d1	0.0	0.05	YES	AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256
d2	0.0	0.05	YES	AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256
d3	0.0	0.05	YES	AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256

The test provides evidence against the null hypothesis, with a significance level of 0.05. Meaning that the test support the theory that the differences we see between the data set are statistically significant. It also indicates that the difference is significant between all pairs. However, it is important to note that with a large sample size, even small

differences in the distributions can produce a significant result. Hence, it could be more valuable to further analyze the effect size to determine whether the difference is substantial. From the mean values presented in Table 5.2. The largest difference in absolute value is between the AES128 and Camellia256 in data set d3, with a difference of 2.26 ms and an average of 1.16 ms across all three data sets. The difference between the algorithms seems to grow slightly when the input size is larger, in favor of the two AES algorithms. The standard deviation for encryption time is on average approximately 2.3 ms across all data sets and algorithms. Meaning that the data collected, on average deviates $\pm$ 2.3 ms from the mean.

In conclusion, the data gathered from the encryption time experiment gives no clear result on which algorithm performs best in terms of encryption time. The statistical analysis, as presented in Table 6.7 gives evidence for the hypothesis that the difference seen between the algorithms is statistically significant. However, that difference does not translate to a substantial amount in reality. There's also not a substantial difference inherently between the algorithms and key sizes, which is a slight surprise considering a larger key size amount to more rounds performed by the algorithm. Hence, one would expect that the runs with a key size of 256 bits would be slower than the ones with 128 bits. The slight edge in this experiment by only considering the mean values is given to AES128, followed by AES256, Camellia128, and Camellia256 in that order.

6.3 Decryption Time

In this section, the data presented in 5.8 will be further analyzed. Similar to the previous section, the Kruskal-Wallis test is performed to analyze whether there is a significant difference between the methods. However, here on the decryption time. The results are presented in Table 6.8.

Table 6.8: Kruskal-Wallis statistical test on decryption time.

Data set	P-value	α	Reject Null Hypothesis	Pairs
d1	0.0	0.05	YES	AES128-AES256
				AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256
d2	0.0	0.05	YES	AES128-AES256
				AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256
d3	0.0	0.05	YES	AES128-AES256
				AES128-AES256
				AES128-Camellia128
				AES128-Camellia256
				AES256-Camellia128
				AES256-Camellia256
				Camellia128-Camellia256

The test again strengthens the hypothesis that there is a significant difference between the algorithms when performing decryption. As well as a pairwise difference between all algorithms. By further analyzing the effect size, the mean values between the respective algorithm and key size are compared to each other. AES128 performs better than the others on the largest data set. However, AES256 is slightly faster on data sets *d1* and *d2*. Again, the difference between the fastest (AES128) and slowest (Camellia256) performing algorithm amounts to 1.91 ms, or in percent, AES128 is approximately 2% faster than Camellia256 on the *d3* data set. The indication that the difference between the methods seems to grow is also seen here, where both Camellia versions tend to be slower than the others as the data set increases in size. However, not by a great amount. The standard deviation between all data sets and algorithms is on average approximately 0.5 ms, which is smaller in comparison to encryption time.

In conclusion, no algorithm outperforms any other algorithms by a clear amount. The statistical test again gives evidence of a significant difference between the algorithms. The effect size between the slowest and fastest performing algorithm amounts to 1.91 ms for a data set with unique ids corresponding to 4000 students. Which could be considered negligible in the use case of the problem formulation of this thesis. By only analyzing the

mean value of decryption time, the same conclusion as in encryption time is drawn. That is, AES128 performs the fastest, followed by AES256, Camellia128, and Camellia256. Consequently. By combining the results of encryption and decryption time the same conclusion is drawn.

6.4 Ciphertext Growth

In 5.4 it is seen that the algorithms produce the same constant size of the ciphertext file. For a test file of 9 000 bytes, the ciphertext file grows to 32 000 bytes, corresponding to approximately a 3.6 times increase from plaintext to ciphertext. The same is true for the test files of 18 000 and 36 000 bytes respectively. Considering the similarities between the algorithms, a result like this is to be expected. The four algorithms are all 128-bit block ciphers. Meaning that fundamentally a 128-bit block of plaintext translates to a 128-bit block of ciphertext. Using the CBC mode with an initial vector, the size of the ciphertext would increase by a constant. Additionally, adding padding to the 56 bits of plaintext increases the size of each ciphertext by an additional 72 bits. The experiments are conducted on the same data, hence the same amount of padding should be expected.

In conclusion, the size of the file with the ciphertext is approximately 3.6 times larger than the original file with the plaintext using any of the four algorithms. The number of characters in the test files also translates to approximately 4.6 bytes per character after encryption.

7 Discussion

The experiments conducted with encryption time, decryption time, and cipher text size as the dependent variables gave some interesting findings. The performance results of AES in comparison to Camellia showed that there is a significant difference between the algorithms. However, it was found that the difference did not amount to a substantial amount in reality. The results also indicate that the sample collected on encryption time is more spread than the sample on decryption time. The results between the two encryption methods AES and Camellia and their respective larger key-size alternative showed no substantial difference in terms of performance, which is not to be expected. This is in contrary to the findings in [5], where the encryption and decryption time of AES on average increased by 21% for each increment in key size while using the CBC mode. Consequently, going from AES128 to AES256 would indicate a 42% increase. In this work, the 256-bit AES version performed marginally faster with the 9KB and 18KB data sets. Only when the data set grew to 36KB, the 128-bit version was faster. This was also true for the Camellia algorithm. The reason for not seeing a larger difference could be that the data sets used in the experiment are not large enough to get a measurable difference. There could also be a problem with the implementation and how the experiments are conducted, however, any issues that would somehow favor the 256-bit version could not be identified.

In the study carried out by [4], the AES algorithm with a key size of 256 bits with CBC mode encrypted a 1KB file in 52.7 ms on average. Compared to the results gathered in this thesis, the AES 256 algorithm encrypted a 9KB data set in 23.89 ms on average. That would amount to approximately 2.65 ms per KB under the assumption of linear growth, which is considerably faster than the implementation in [4]. On the other hand, the authors of [4] also performs experiments on considerably larger input sizes, namely 100KB, 1MB, 10MB, and 100MB. The results differ from the findings of this thesis. Namely, the 100KB file took 58.4 ms, followed by 1MB at 64.3 ms. Meaning there is not a major increase in encryption time when the input size of 1KB increases by 100 and 1000 times. The results in this thesis point to a somewhat linear increase. Meaning the encryption time approximately doubles when the size of the input doubles. The belief is that the difference partly depends on the implementation of the algorithms. In this work, the unique ids read from the file are encrypted as a string one by one sequentially which might not be the case in [4]. Additionally, it's hard to draw any conclusions from the comparison considering their implementation is done in C#, not to mention the hardware differences.

Looking at the performance analysis made by [6], which is done in Python. The AES128 algorithm encrypted 1MB of plaintext in approximately 40 ms, which is also considerably faster than our implementation, while also using a hardware configuration similar to or worse than the configuration used in this work. The cipher-to-text ratio in [6] is approximately 1.6, compared to 3.6 from the results of this study. On the other hand, their study does not use the CBC mode of operations with an initial vector, which leads to a smaller ratio. In addition, this study performs cryptographic operations on strings smaller than one block. Hence, for each produced ciphertext, execution time and data storage for 72 bits of padding is added to the results.

Even though the findings of this thesis did not give a clear winner in terms of performance, the results give some indications that favor AES. When choosing an encryption method to use in a particular system is it important to also consider other factors other than performance. For example, security aspects. On the one hand, AES256 is more resistant

to a brute-force attack than its 128-bit variant [44]. However, comparing their brute-force resistance is not a meaningful metric since both would take an unrealistic amount of time to break. On the other hand, AES128 has a stronger key schedule [44], making it less likely to have an attack developed against it. AES128 also has broader support in Python libraries. For these reasons, the AES128 version is used together with the access control mechanism in the proposed implementation which is tested in the verification and validation method.

In reference to the problem formulated in 1.3, the two symmetric encryption algorithms AES and Camellia were identified by the literature review as candidates for encryption of the unique identifiers of the students. These two were tested and evaluated for performance using a series of controlled experiments. The evaluation gave a slight edge to the AES128 algorithm. However, it's also important to acknowledge that the hardware implementation of both algorithms was not taken into consideration. While the focus of this thesis was to examine the computational performance of the AES and Camellia encryption algorithms, it's worth noting that the performance in a real-time scenario could be significantly affected by hardware capabilities and optimizations. For instance, many Intel processors come with a dedicated AES instruction set [45] that can significantly improve the performance of AES encryption. Likewise, other encryption algorithms may be better suited for devices where power consumption is a constraint, such as IoT devices. Not considering these factors could affect the analysis of the results gathered in this thesis. For instance, an algorithm that was considered to perform better in the context of this work might perform poorly in a context with different conditions.

To solve the problems related to role-based access to the data, a system where each class data get encrypted with a separate key and each user gets assigned a role was implemented. The role determines if the user gets access to a master key which is then used to decrypt the class key related to that role. By doing this, the implementation ensures that a head principal can decrypt all data and a class principal can only decrypt the data of its own class. This solution was tested using the verification & validation method. The implementation passed the verification phase, using automated testing created through the *unittest* framework. The solution was also validated by the stakeholder of this project, who stated that it fulfilled the basic requirements agreed upon. By disconnecting the roles from the encryption of the unique identifiers, the problem of having to re-encrypt the data when changes to access occurred was also eliminated. Hence, the research questions formulated in 1.3 are considered to have been answered.

8 Conclusion

In this thesis, two cryptographic algorithms with two different key sizes are implemented in Python, AES128, AES256, Camellia128, and Camellia256. Both are tested regarding performance when it comes to the time of execution and the size of data growth between plaintext and ciphertext when encrypting and decrypting unique identifiers. In all tests, the algorithms performed equally both in time of execution and growth of data, but with AES128 performing marginally faster.

The findings are relevant in the matter of industry and society where cryptography is a vital part of protecting individuals' anonymity and integrity when stored in databases all over the world. Due to the generality of encryption, the results presented in this thesis can be used as ground when choosing a cryptographic algorithm to implement in any system. Results in this thesis in particular can be used for anonymization of usernames or short, unique identifiers. For example, digital grading systems where student anonymity is a priority to prevent biased grading or simply protect students in case of an IT breach.

Test results in this thesis can be applied anywhere where any of the tested algorithms are used. But it is worth noting that the data used in the experiment consists of small strings of length 7. Hence, bigger sizes of data might give different results due to the block size and added padding.

The results could have been optimized if performed on a machine with Linux installed and more advanced hardware. When performing the tests, no consideration was taken of the current state of OS. Hence, uncontrolled background processes can affect the results, such as built-in anti-virus.

8.1 Future work

In the beginning, when starting writing this thesis, the plan was to find and use, a cryptographic method that made use of multi-key encryption. But the literature review showed, that there exists a very limited amount of multi-key encryption methods all of which are computationally heavy. Since computational speed is an important factor in selecting an algorithm, we decided to choose an algorithm that only handles a cryptographic operation, and instead implement separate access control. By doing that, the cryptographic operations will be executed faster while maintaining secure access.

Additionally, a web interface with administrative responsibilities would be implemented to demonstrate an RBAC solution to perform live changes to access rights of users for further tests in functionality and security. The web interface would be responsible for displaying an interactive admin interface used to administer access rights for teachers. It would show specific school classes from a teacher's perspective based on their access rights. Furthermore, the interface would offer the ability to re-encrypt all or some data, providing further control in the event of data compromise.

A database would be set up with synthetic data so that the data could be displayed in the web interface as well as consider making the data inference safe by further hiding additional unique identifiers for the individuals.

In addition, further research to find a fitting solution for sending data securely over the internet would be considered to prevent MITM attacks. Further explanation of this can be seen in the literature review, see 2.5.1 and 2.8.1 where HTTPS and OAuth 2.0 were reviewed.

Within the limited time of this degree project, a light version of a key management system was implemented to demonstrate a prototype. However, in continuous work within this project an outsourced key management system such as Microsoft Azure Key Vault

[46], AWS Key Management System [47] or Google Cloud Key Management [48] should be considered to further enhance security within key storage and transfer. The key manager would be responsible for securely storing all cryptographic keys in an encrypted format. It would only decrypt and retrieve the requested key for a specific school class when a user with the correct login credentials and role requests the data from the server. By separating the keys from the medium which holds the data, a higher level of security would be achieved.

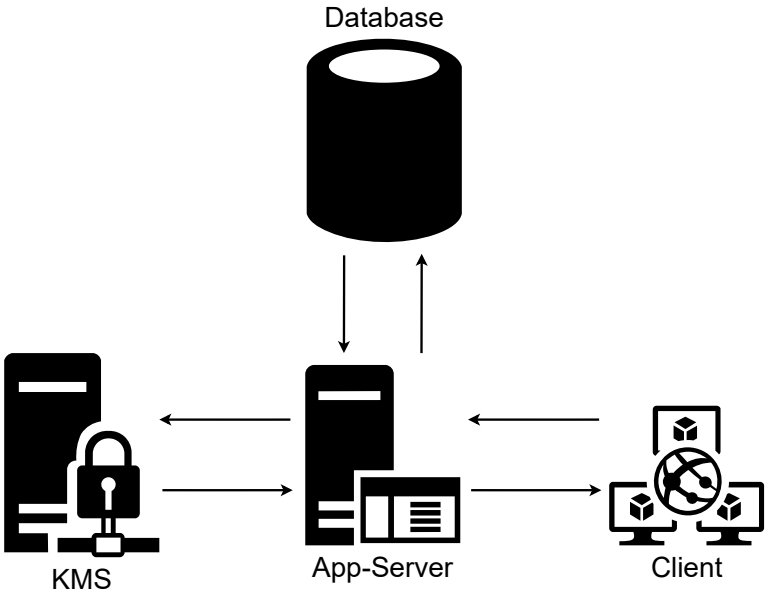

Figure 8.10: Client-server communication flow.

Figure 8.10 shows an abstraction of how the final system can look when all parts are implemented.

Going forward, a more comprehensive analysis of encryption algorithm performance could include factors such as hardware optimization and applicability to specific use cases. Such an approach would provide a more realistic and robust assessment of algorithm performance, thus allowing for more accurate comparisons and more informed decision-making in the selection and implementation of encryption algorithms.

References

[1] C. P. Pfleeger, *Security in computing*, 5th ed. Pearson, 2015.

[2] "Cloud computing - statistics on the use by enterprises — ec.europa.eu," https://ec.europa.eu/eurostat/statistics-explained/index.php?oldid=416727#Use_of_cloud_computing:_highlights, [Accessed 24-Apr-2023].

[3] "Eddtechlnu," 2023, accesesed Mars. 29, 2023. [Online]. Available: https://lnu.se/en/research/research-groups/edtechlnu/

[4] H. Dibas and K. E. Sabri, "A comprehensive performance empirical study of the symmetric algorithms:aes, 3des, blowfish and twofish," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 344–349.

[5] D. F. García, "Performance evaluation of advanced encryption standard algorithm," in *2015 Second International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, 2015, pp. 247–252.

[6] A. Kubadia, D. Idnani, and Y. Jain, "Performance evaluation of aes, arc2, blowfish, cast and des3 for standalone systems : Symmetric keying algorithms," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 2019, pp. 118–123.

[7] "Python offical site." [Online]. Available: https://www.python.org/

[8] A. Abdullah, "Advanced encryption standard (aes) algorithm to encrypt and decrypt data," 06 2017.

[9] J. J. V. Nayahi and V. Kavitha, "Privacy and utility preserving data clustering for data anonymization and distribution on hadoop," *Future Generation Computer Systems*, vol. 74, pp. 393–408, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X16304459

[10] European Union Agency for Cybersecurity, "Threat landscape report 2022," European Union Agency for Cybersecurity, Tech. Rep., 2022. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022

[11] "The hipaa privacy rule," 2022, accesesed Mars. 29, 2023. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/index.html

[12] J. Salas and J. Domingo-Ferrer, "Some basics on privacy techniques, anonymization and their big data challenges," *Math. Comput. Sci.*, vol. 12, no. 3, pp. 263–274, Sep. 2018.

[13] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," 1998.

[14] R. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *21st International Conference on Data Engineering (ICDE'05)*, 2005, pp. 217–228.

[15] G. J. Simmons, "History of cryptology," "accessed : 14-apr-2023".

[16] E. Nyman, "Cryptography: A study of modern cryptography and its mathematical methods," Master's thesis, Uppsala universitet, 752 36 Uppsala, 2021.

[17] A. Nadeem and M. Javed, “A performance comparison of data encryption algorithms,” 09 2005, pp. 84 – 89.

[18] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, “Report on the development of the advanced encryption standard (AES),” *J. Res. Natl. Inst. Stand. Technol.*, vol. 106, no. 3, pp. 511–577, May 2001.

[19] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, “Advanced encryption standard (aes),” 2001-11-26 2001.

[20] Y. Deng, T. Xie, H. Shi, and J. Gong, “Research on the f-function of camellia,” in *2011 International Conference on Electrical and Control Engineering*, 2011, pp. 1232–1235.

[21] M. M. J. Nakajima, “A description of the camellia encryption algorithm,” 04 2004, pp. 1–15. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3713

[22] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, “Camellia: A 128-bit block cipher suitable for multiple platforms — design and-analysis,” in *Selected Areas in Cryptography*, D. R. Stinson and S. Tavares, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 39–56.

[23] G. J. Simmons, “Rsa encryption,” "accessed : 14-apr-2023". [Online]. Available: "https://www.britannica.com/topic/RSA-encryption"

[24] D. E. Comer, *"Computer Networks and Internets, 6th ed"*. Pearson Education, 2015.

[25] F. Callegati, W. Cerroni, and M. Ramilli, “Man-in-the-middle attack to the https protocol,” *IEEE Security privacy*, vol. 7, no. 1, pp. 78–81, Feb. 2009.

[26] A. Satapathy, J. Livingston *et al.*, “A comprehensive survey on ssl/tls and their vulnerabilities,” *International Journal of Computer Applications*, vol. 153, no. 5, pp. 31–38, 2016.

[27] D. Wagner, B. Schneier *et al.*, “Analysis of the ssl 3.0 protocol,” in *The Second USENIX Workshop on Electronic Commerce Proceedings*, vol. 1, no. 1, 1996, pp. 29–40.

[28] N. H. Sultan, V. Varadharajan, L. Zhou, and F. A. Barbhuiya, “A role-based encryption (rbe) scheme for securing outsourced cloud data in a multi-organization context,” *IEEE Transactions on Services Computing*, pp. 1–14, 2022.

[29] T. E. of Encyclopaedia Britannica, “client-server architecture,” "accessed : 16-apr-2023". [Online]. Available: "https://www.britannica.com/technology/client-server-architecture"

[30] H. S. Oluwatosin, “Client-server model,” *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67–71, 2014.

[31] L.Zhou, V. Varadharajan, and M. Hitchens, “Generic constructions for role-based encryption,” vol. 14, pp. 417–430, 2014.

[32] D. Ferraiolo, J. Cugini, and R. Kuhn, “Role-based access control (rbac): Features and motivations,” Dec. 1995.

[33] E. Bertino, "Rbac models — concepts and trends," *Computers Security*, vol. 22, no. 6, pp. 511–514, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404803006096

[34] V. Chamola, A. Jolfaei, V. Chanana, P. Parashari, and V. Hassija, "Information security in the post quantum era for 5g and beyond networks: Threats to existing cryptography, and post-quantum cryptography," *Computer communications*, vol. 176, pp. 99–118, 2021.

[35] "Cryptography, Python encryption Library." [Online]. Available: https://cryptography.io/en/latest/

[36] "Linnaeus university online-library." [Online]. Available: https://lnu.se/ub/

[37] "Google scholar." [Online]. Available: https://scholar.google.com/

[38] P. Bhandari, "What Is a Controlled Experiment? | Definitions 038; Examples," *Scribbr*, 12 2022. [Online]. Available: https://www.scribbr.com/methodology/controlled-experiment/

[39] "Unittest library." [Online]. Available: https://docs.python.org/3/library/unittest.html

[40] "ISO/IEC 18033-3:2005." [Online]. Available: https://www.iso.org/standard/37972.html

[41] "Including the Pyca cryptography library." [Online]. Available: https://cloud.google.com/kms/docs/crypto

[42] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952. [Online]. Available: http://www.jstor.org/stable/2280779

[43] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965. [Online]. Available: http://www.jstor.org/stable/2333709

[44] E. Tobias, "128 or 256 bit encryption: Which should i use?" Feb 2022. [Online]. Available: https://www.ubiqsecurity.com/128bit-or-256bit-encryption-which-to-use/#:~:text=AES.

[45] Intel. (2012) Intel® advanced encryption standard instructions (aes-ni). Accessed: 2023-07-11. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-encryption-standard-instructions-aes-ni.html

[46] "Azure key vault." [Online]. Available: https://learn.microsoft.com/en-us/azure/key-vault/general/overview

[47] "AWS key management." [Online]. Available: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html

[48] "Google cloud key management." [Online]. Available: https://cloud.google.com/kms/docs

A Appendix 1

The raw data from the experiments are uploaded to a public google drive folder and can be accessed by clicking this link:
https://drive.google.com/drive/folders/1hO4zpMo-NvFnIMEvxRKPaQU3nvkVbF_8

A.1 Code snippets

```
#
# Size: D1
#
# AES 128
aes_test("aes128", 16, 25, 1000)
# AES 256
aes_test("aes256", 32, 25, 1000)
# Camellia 128
camellia_test("camellia128", 16, 25, 1000)
# Camellia 256
camellia_test("camellia256", 32, 25, 1000)
```

Figure 1.11: D1 initiated from main.

```
def set_method(self, method: str) -> algorithms:
    if method == "aes128":
        return algorithms.AES128
    if method == "aes256":
        return algorithms.AES256
    else:
        return None
```

Figure 1.12: Set method and set mod functions in the AES class.

```
def aes_test(self, key_dict: dict, class_entries: int):
    result_encrypt_time = 0
    result_decrypt_time = 0
    cipher_result = []

    for i in range(class_entries):
        txt_file = f"class{i+1}.txt"
        key = key_dict[f"class_{i+1}"][0]
        iv = key_dict[f"class_{i+1}"][1]
        with open("database/"+txt_file, 'r') as out_f:
            for line in out_f:
                line = bytes(line.replace("\n",""),
                       encoding='utf-8')

                start_enc = time.perf_counter_ns()
                cipher = self.encrypt_data(line, key, iv)
                end_enc = time.perf_counter_ns()

                result_encrypt_time += end_enc - start_enc
                hex_string = cipher.hex()
                cipher_result.append(hex_string)

                start_dec = time.perf_counter_ns()
                cipher = self.decrypt_data(cipher, key, iv)
                end_dec = time.perf_counter_ns()

                result_decrypt_time += end_dec - start_dec

    self.write_bytes_to_file(cipher_result, "results/temp.txt")
    filestat = os.stat("results/temp.txt").st_size

    return result_encrypt_time, result_decrypt_time, filestat
```

Figure 1.13: AES test function inside the class aesAlgo.

```
def encrypt_data(self, data: bytes, key: bytes, iv: bytes):
    cipher = Cipher(self.method(key), self.mode(iv))
    encryptor = cipher.encryptor()

    padd = padding.PKCS7(128).padder()
    padded_data = padd.update(data)
                + padd.finalize()

    cipher_text = encryptor.update(padded_data)
                + encryptor.finalize()

    return cipher_text
```

Figure 1.14: AES encryption function.

```
def decrypt_data(self, data: bytes, key: bytes, iv: bytes):
    cipher = Cipher(self.method(key), self.mode(iv))
    decryptor = cipher.decryptor()

    padded_pt = decryptor.update(data)
              + decryptor.finalize()
    unpadd = padding.PKCS7(128).unpadder()

    plain_text = unpadd.update(padded_pt)
               + unpadd.finalize()

    return plain_text
```

Figure 1.15: AES decryption function.

```
class KeyGenerator:

    def generate_key(nr_bytes: int) -> bytes:
        key = os.urandom(nr_bytes)
        return key

    def generate_iv(nr_bytes: int) -> bytes:
        iv = os.urandom(nr_bytes)
        return iv
```

Figure 1.16: KeyGenerator which generates key and iv based on argument size.

```
def set_unencrypted_key(self, class_id: str, key_size: int,
iv_size: int) -> None:
    self.key_iv[class_id] =
    [KeyGenerator.generate_key(key_size),
    KeyGenerator.generate_iv(iv_size)]
```

Figure 1.17: Method inside key_manager called from main to create a key dictionary for the experiment.

```
def write_master_key(self):
    with open("bin/masterKey.bin", "wb") as writer:
        pickle.dump(self.master_key_iv, writer)

def read_master_key(self):
    with open("bin/masterKey.bin", "rb") as reader:
        self.master_key_iv = pickle.load(reader)
```

Figure 1.18: Methods used to save and read the master key.

```
def access_control(self, requests_access_to: str) -> bool:
    if ('principal' in self.roles):
        return True
    if (requests_access_to in self.roles):
        return True
    else:
        return False
```

Figure 1.19: Methods inside roleController, used to save and read the master key.

```
def get_decrypted_key(self, class_id) -> bytes:
    if class_id in self.encrypted_key_iv:
        return self.algo.decrypt_data(
        self.encrypted_key_iv[class_id][0],
        self.master_key_iv[0], self.master_key_iv[1]
        )
    return None
```

Figure 1.20: Method inside keyManager, used to decrypt an encrypted key, using the master key.