



IL246X DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND CYCLE,
SWEDEN 2022

Performance Evaluation of Cryptographic Algorithms on ESP32 with Cryptographic Hardware Acceleration Feature

QIAO JIN

Performance Evaluation of Cryptographic Algorithms on ESP32 with Cryptographic Hardware Acceleration Feature

Qiao Jin

2022-02-24

Master's Thesis

Examiner
Zhonghai Lu

Academic adviser
Matthias Becker

Abstract

The rise of the Internet of Things (IoT) and autonomous robots/vehicles comes with a lot of embedded electronic systems. Small printed circuit boards with microcomputers will be embedded almost everywhere. Therefore, the security and data protection of those systems will be a significant challenge to take into consideration for the future development of IoT devices. Cryptographic algorithms can be used to provide confidentiality and integrity for data transmitted between those embedded devices. It is important to know what kind of algorithm is the most suitable for the specified task and the selected embedded device.

In this thesis, several commonly used cryptographic algorithms are evaluated and an ESP32 based IoT device is chosen as the evaluation platform. ESP32 is a series of low cost and low power System-on-Chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. Additionally, ESP32 has the hardware acceleration feature for commonly used cryptographic algorithms. The goal of this thesis is to evaluate the performances of different cryptographic algorithms on the ESP32 with and without using the hardware acceleration feature. The execution times of different cryptographic algorithms processing data with varying sizes are collected, and the performance of each cryptographic algorithm is then evaluated.

A data logging scenario is evaluated as a case study where the ESP32 periodically sends data to a remote database. Under different configurations of the ESP32, the transmission time of encrypted and non-encrypted communications via Hypertext Transfer Protocol Secure (HTTPS) and Hypertext Transfer Protocol (HTTP) will be compared.

The results can be used to simplify the calculation of performance/protection trade-offs for specific algorithms. It also shows that the built-in hardware acceleration has a significant impact on increasing those algorithms' performances. For Advanced Encryption Standard (AES), the throughput for encryption increased by 257.8%, and for decryption 222.7%. For Secure Hash Algorithm (SHA-2), the throughput increased by 165.2%. For Rivest-Shamir-Adleman (RSA), the encryption throughput has a decrease of 40.7%, and decryption has an increase of 184%. Furthermore, the results can also aid the design and development of a secure IoT system incorporating devices built with ESP32.

Keywords

Embedded systems; ESP 32; Cryptographic algorithms; Cryptographic hardware acceleration; AES; RSA; SHA-2; HTTPS

Sammanfattning

Uppkomsten av Internet of Things (IoT) och autonoma robotar / fordon kommer med många inbyggda elektroniska system. Små kretskort med mikrodatorer kommer att vara inbäddade nästan överallt. Därför kommer säkerheten och dataskyddet för dessa system att vara en betydande utmaning att ta hänsyn till för den framtida utvecklingen av IoT-enheter. Kryptografiska algoritmer kan användas för att ge sekretess och integritet för data som överförs mellan de inbäddade enheterna. Det är viktigt att veta vilken typ av algoritm som är bäst lämpad för den angivna uppgiften och den valda inbäddade enheten.

I denna avhandling utvärderas flera vanliga kryptografiska algoritmer och en ESP32-baserad IoT-enhet väljs som utvärderingsplattform. ESP32 är en serie av låga och lågeffektiva system-on-chip-mikrokontroller med integrerat Wi-Fi och dual-mode Bluetooth. Dessutom har ESP32 hårdvaruaccelereringsfunktionen för vanliga kryptografiska algoritmer. Målet med denna avhandling är att utvärdera prestanda för olika kryptografiska algoritmer på ESP32 med och utan att använda hårdvaruaccelereringsfunktionen. Exekveringstiderna för olika kryptografiska algoritmer som behandlar data med olika storlekar samlas in och prestanda för varje kryptografisk algoritm utvärderas sedan.

Ett dataloggningsscenario utvärderas som en fallstudie där ESP32 regelbundet skickar data till en fjärrdatabas. Under olika konfigurationer av ESP32 jämförs överföringstiden för krypterad och icke-krypterad kommunikation via Hypertext Transfer Protocol Secure (HTTPS) och Hypertext Transfer Protocol (HTTP).

Resultaten kan användas för att förenkla beräkningen av prestanda / skydda avvägningar för specifika algoritmer. Det visar också att den inbyggda hårdvaruaccelerationen har en betydande inverkan på att öka dessa algoritmers prestanda. För Advanced Encryption Standard (AES) ökade genomströmningen för kryptering med 257,8% och för dekryptering 222,7%. För Secure Hash Algorithm (SHA-2) ökade kapaciteten med 165,2%. För Rivest-Shamir-Adleman (RSA) har krypteringsflödet minskat med 40,7% och dekryptering har ökat med 184%. Dessutom kan resultaten också hjälpa till att utforma och utveckla ett säkert IoT-system som innehåller enheter byggda med ESP32.

Nyckelord

Inbyggda system; ESP 32; Kryptografiska algoritmer; Kryptografisk hårdvaruacceleration; AES; RSA; SHA-2; HTTPS

Acknowledgments

I would like to thank my examiner Zhonghai Lu for providing the opportunity for me to do this thesis project and offer valuable resources and guidance during the thesis period. I would also like to thank my supervisor Matthias Becker for his support and supervision throughout the entire time. He took his time to help me whenever I had any questions and provided me with valuable feedback.

Stockholm, January 2022

Qiao Jin

Table of contents

Abstract	i
Keywords.....	i
Sammanfattning	iii
Nyckelord.....	iii
Acknowledgments.....	v
Table of contents.....	vii
List of Figures.....	xi
List of Tables	xiii
List of Algorithms.....	xiv
List of acronyms and abbreviations	xv
1 Introduction	1
1.1 Background	1
1.2 Problem.....	2
1.3 Purpose.....	2
1.4 Goals	2
1.5 Ethics and Sustainability	3
1.6 Research Methodology	3
1.7 Delimitations.....	3
1.8 Structure of the Thesis.....	3
2 Background	5
2.1 Symmetric Cryptography.....	5
2.1.1 The Advanced Encryption Standard (AES).....	6
2.2 Asymmetric Cryptography.....	9
2.2.1 RSA	10
2.3 Hashing	12
2.3.1 Secure Hash Algorithms (SHA)	13
2.4 Summary of Selected Algorithms	15
2.5 Related Work	15
2.5.1 Security Concerns and Cryptography of IoT System.....	15
2.5.2 Performance Analysis of Symmetric Key Algorithms	16
2.5.3 Performance Analysis of Cryptographic Algorithms on Resource Constraint Devices	16
2.5.4 Comparing Elliptic Curve Cryptography and RSA	16
2.5.5 Performance Analysis of Security Algorithms for IoT Devices.....	17
2.5.6 Analysis of Cryptographic Algorithms on Raspberry Pi	17
2.5.7 Attacking the ESP32 IoT Devices	17
2.5.8 Secure Data Communication	17
2.5.9 Performance Evaluation of Cryptographic Ciphers on IoT Devices....	18
2.5.10 Security and Performance in IoT	18
2.6 ESP32.....	19
3 System Design and Methodology.....	21
3.1 Research Process	21
3.1.1 Research Methodology.....	21
3.1.2 Development Environment	22

3.1.3	Implementation of Measurement Program	22
3.1.4	Data Collection	22
3.1.5	Data Evaluation	23
3.2	Research Paradigm	23
3.2.1	Positivist Paradigm	23
3.3	Data Collection	23
3.3.1	Sampling.....	23
3.3.2	Sample Size	24
3.4	Experimental Design	24
3.4.1	Evaluation Environment.....	24
3.4.2	Hardware and Software Environment	25
3.5	Assessing Reliability and Validity of the Data Collected	27
3.5.1	Reliability	27
3.5.2	Validity	27
3.6	Planned Data Analysis	28
3.6.1	Data Analysis Technique	29
3.7	Summary.....	29
4	Encryption Performance Measuring Programs and Hardware Acceleration	31
4.1	The Evaluation Programs.....	31
4.1.1	Pseudo-random Number Generator (PRNG).....	31
4.1.2	Time Measurement Function.....	32
4.1.3	AES Algorithm	32
4.1.4	RSA Algorithm	32
4.1.5	SHA-2 Algorithm	33
4.2	Hardware Acceleration	34
4.3	Summary.....	34
5	Results and Evaluation	37
5.1	Major Results.....	37
5.1.1	General experimental description.....	37
5.1.2	AES algorithm.....	37
5.1.3	SHA algorithm	40
5.1.4	RSA Algorithm	43
5.2	The Multi-Precision Integer Function	49
5.3	Discussion	52
5.4	Summary.....	53
6	Case Study: Secure Communication with Databases	54
6.1	The Setup.....	54
6.1.1	HTTP	55
6.1.2	HTTPS	56
6.2	The Measurements and Data Collection	59
6.3	Measurements Result.....	59
6.4	Discussion.....	61
7	Conclusions and Future work.....	63
7.1	Conclusions.....	63
7.2	Future Work.....	64
	References	65

Appendix A: RSA Key Pairs.....	69
Appendix B: Self-signed Certificate and the Private Key	71
Appendix C: Random Generated Primes	73

List of Figures

Figure 2.1: Symmetric-key cryptosystem.....	6
Figure 2.2: High-level AES encryption process	8
Figure 2.3: High- level AES decryption process	9
Figure 2.4: Asymmetric-key cryptosystem.....	10
Figure 2.5: High-level Process of the RSA Algorithm.....	12
Figure 2.6: Process of digital signature using hash function	13
Figure 2.7: High-level Diagram of SHA-1	15
Figure 2.8: ESP-WROVER-KIT block diagram [34].....	19
Figure 2.9: Function block diagram of ESP32 taken from [35].....	20
Figure 3.1: Steps for the research process.....	21
Figure 3.2: Development of applications for ESP32 [36].....	22
Figure 3.3: Flow chart on how the measurement program works. The X is the number of data points and its value varies between 100, 7 and 15 with the corresponding algorithm.	26
Figure 3.4: Overhead of <i>gettimeofday ()</i> function.	28
Figure 4.1: The process how the data message is encrypted and decrypted using the AES algorithm.	32
Figure 4.2: The process of how the data message is encrypted and decrypted using RSA 1024/ 2048.....	33
Figure 4.3: The process of how the data message is hashed using SHA-256.	33
Figure 4.4: Espressif IoT development framework configuration.	34
Figure 5.1: Performance of AES 128-bits CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	37
Figure 5.2: Encryption of AES 128-bits in CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	38
Figure 5.3: Decryption of AES 128-bits in CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	38
Figure 5.4: The throughput of the AES-128, the left diagram shows the throughput of AES-128 encryption, the right diagram shows the throughput of AES- 128 decryption.	40
Figure 5.5: Performance of SHA-2 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	41
Figure 5.6: The throughput of SHA-256 performing under different CPU frequencies.	42
Figure 5.7: Performance of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	43
Figure 5.8: Encryption of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	44
Figure 5.9: Decryption of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	44
Figure 5.10: The throughput of RSA-1024 performing under different CPU clock frequencies, the left diagram shows the throughput of the encryption, the right diagram shows the throughput the decryption.	45
Figure 5.11: Performance of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	46
Figure 5.12: Encryption of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	46
Figure 5.13: Decryption of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.	47

Figure 5.14: The throughput of RSA-2048 performing under different CPU clock frequencies, the left diagram shows the throughput of the encryption, the right diagram shows the throughput the decryption.	48
Figure 5.15: Execution time of <code>MBEDTLS_MPI_EXP_MOD()</code> with small <i>a</i> and <i>e</i>	50
Figure 5.16: Execution time of <code>MBEDTLS_MPI_EXP_MOD()</code> with large <i>a</i> and <i>e</i>	51
Figure 5.17: Execution time of <code>MBEDTLS_MPI_EXP_MOD()</code> with a large <i>a</i> and a varying <i>e</i>	52
Figure 6.1: The base setup for communication between ESP32 WROVER-KIT and InfluxDB database.	54
Figure 6.2: Diagram of HTTP connection between EPS32 WROVER-KIT and the database.	56
Figure 6.3: Diagram of HTTPS connection between EPS32 WROVER-KIT and the database.	58
Figure 6.4: The TLS connection process.	59
Figure 6.5: Transmission time for HTTP and HTTPS in different CPU frequencies and with/ without hardware acceleration.	60

List of Tables

Table 2-1:	Properties of AES, RSA and SHA.	15
Table 3-1:	Summary of the number of data points and the data sizes for all measured algorithms	24
Table 3-2:	Time difference between two <i>gettimeofday ()</i> functions	28
Table 4-1:	Summary of the configuration for all evaluated cryptographic algorithms.	35
Table 5-1:	Performance of AES 128-bits CBC mode with CPU clock frequency 160MHz.....	39
Table 5-2:	Performance of AES 128-bits CBC mode with CPU clock frequency 240MHz.....	39
Table 5-3:	Performance of SHA-2 with CPU clock frequency 160MHz.....	41
Table 5-4:	Performance of SHA-2 with CPU clock frequency 240MHz.....	41
Table 5-5:	Performance of RSA 1024 with CPU clock frequency 160MHz.....	45
Table 5-6:	Performance of RSA 1024 with CPU clock frequency 240MHz.....	45
Table 5-7:	Performance of RSA 2048 with CPU clock frequency 160MHz.....	47
Table 5-8:	Performance of RSA 2048 with CPU clock frequency 240MHz.....	47
Table 5-9:	Values of random generated 512-bit primes and other parameters.....	50
Table 5-10:	A summary of the throughput increased with hardware acceleration for all three cryptographic algorithms.	53
Table 6-1:	The average transmission time of ESP32 sending data to the database in different configuration.	60

List of Algorithms

Algorithm 4–1: The pseudo-random number generator.....	31
Algorithm 5–1: The hardware accelerated version of <i>MBEDTLS_MPI_EXP_MOD()</i>	49
Algorithm 6–1: Command for generate the self-signed certificate and the private key.	55

List of acronyms and abbreviations

AES	Advanced Encryption Standard
CA	Certificate Authority
ECC	Elliptic Curve Cryptography
ESP-IDF	Espressif IoT Development Framework
FIPS	U.S Federal Information Processing Standard
FLOPS	Floating-point Operations Per Second
GCHQ	Government Communications Headquarters
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
NIST	National Institute of Standards and Technology
PGP	Pretty Good Privacy
PKC	Public Key Cryptography
PRNG	Pseudo-Random Number Generator
RAM	Random-Access Memory
RSA	Rivest–Shamir–Adleman
SHA	Secure Hash Algorithms
SSL	Secure Sockets Layer
S/MIME	Secure/Multipurpose Internet Mail Extensions
SSH	Secure Shell
TLS	Transport Layer Security

1 Introduction

The rise of IoT devices in recent years has made our life easier, but at the same time, it also introduced security and privacy issues [1]. These IoT devices are often small printed circuits board packed with sensors and are usually connected with either a bigger device or each other through wireless communication, such as Wi-Fi or Bluetooth. Those IoT devices will sense, compute, store, and transmit data in different domains. In some domains like home automation, surveillance, or healthcare, the data's security and privacy aspects will require data protection mechanisms [2]. The security aspects challenge the design and implementation of the IoT systems to have a particular function to protect data integrity, confidentiality, authenticity, and anonymization. It is a challenge for some IoT devices with limited computation power or memory capacity to achieve all these criteria.

In this thesis, different cryptographic algorithms processing data with varying sizes will be executed on Espressif's ESP32 device[3]. Later, the execution times of those cryptographic algorithms will be measured and evaluated. Those cryptographic algorithms are AES, RSA, and SHA. AES is a symmetric-key algorithm; it is used in many commercial systems and can be found in most modern applications and devices that need encryption functionality. RSA is an asymmetric algorithm, also one of the first public-key cryptographic algorithms [4]. It is mostly used for secure data transmission because it can encrypt messages without the need to secretly exchange the key separately. SHA is a hash algorithm designed for data security and is published by the NIST [5]. SHA is commonly used for encrypting users' passwords into message digests with a fixed length instead of keeping the actual passwords, the message digests will then be stored in the database.

1.1 Background

The ESP32 is a well-known low cost, low power System-on-Chip microcontroller with integrated Wi-Fi and Bluetooth, well suited for many different IoT applications. One of the ESP32's features is cryptographic hardware acceleration; it makes the calculations of several cryptographic algorithms like AES, RSA, SHA-2, Elliptic Curve Cryptography (ECC) much faster. For many applications, data security, authenticity, and integrity are the most critical aspects. The requirements in those aspects are making the encryption algorithms a must-have for many applications. The hardware acceleration aids the low CPU power device and enables the device to run resource-heavy encryption algorithms. Faster execution of those algorithms may also decrease the power consumption, particularly suitable for IoT applications which in most cases have restricted energy source.

There are many different types of cryptographic algorithms, such as symmetric-key algorithms, asymmetric key algorithms, and hash algorithms. In this thesis, one cryptographic algorithm of each type mentioned before is selected and further available as hardware acceleration modules on the ESP32. They are AES for the symmetric key algorithm, RSA for the asymmetric key algorithm, and SHA for the hash algorithm. The performance of the selected cryptographic algorithms will be measured and evaluated.

AES is a symmetric-key algorithm. The secret key has a different key size depending on the security level needed for the task. For the AES algorithm, the key sizes are 128, 192, and 256 bits. There are also five modes of operation of the AES algorithm: ECB (electronic code book), CBC (Cipher block chaining), CFB (cipher feedback), OFB (output feedback), and CTR (counter) [6]. In this thesis, the AES algorithm in CBC mode with 128 bits key length will be evaluated.

RSA is an asymmetric key algorithm, also known as a public-key algorithm. The RSA algorithm's key sizes are 1024 bits, 2048 bits, and 4096 bits; with increased key size comes a higher

security level and the demand for more computational power to execute the algorithm. In this thesis, RSA with key size 1024 and 2048 bits will be evaluated.

SHA is a hash algorithm. The hash algorithm is keyless compared with the symmetric or asymmetric key algorithms. It is designed to be a one-way function making it nearly impossible to transform the hashed text back to the original form. The SHA family includes SHA-0, SHA-1, SHA-2, and SHA3; each of them is more secure than the previous one. The SHA-2 from the SHA family will be evaluated in the thesis.

1.2 Problem

The security issues caused by the vast growth of IoT devices are a big problem for engineers to solve when designing and developing system. When designing and developing systems on the ESP32 SoC, it is good to know what the performance limitations of different cryptographic algorithms. Especially those algorithms supported by the built-in hardware acceleration. Some IoT devices are operating in a restricted mode with constrained resources. This highlights the important to know what the trade-offs are to get the desired performance and the security level for those IoT devices. The execution time and the throughput of each cryptographic algorithms can help the developer to figure out the optimal way to utilize the constrained resources within IoT devices.

1. What are the performance limitations of the ESP32 when performing selected cryptographic algorithms? Also, how does the cryptographic hardware acceleration on the EPS32 affect the performance of those selected cryptographic algorithms?
2. What is the throughput rate of each selected cryptographic algorithm on the ESP32?

1.3 Purpose

This thesis will evaluate the performance, mainly the execution time of selected cryptographic algorithms on ESP32. This will show how well each algorithm performs on the hardware and how the hardware acceleration feature improves the performance compared to a pure software implementation. The evaluation can help developers designing system on ESP32 to select the security level they needed and maintain sufficient performance at the same time.

1.4 Goals

The goal of this project is to evaluate the runtime characteristics of different cryptographic algorithms on the ESP32. This has been divided into the following sub-goals:

1. Design a benchmark program that utilizes three different cryptographic algorithms AES-128, SHA-2, RSA 1024, and RSA 2048 on the ESP32.
2. Measure the execution time of encryption/decryption on varying data sizes, with and without hardware acceleration. The same measurement is also executed on different clock frequencies, which are 160MHz and 240MHz.
3. Evaluate those algorithms' performance with the help of graphs and tables.

A written report and an oral presentation are to be delivered and presented to the examiner and other KTH students by the end of this project.

1.5 Ethics and Sustainability

This thesis has its focus on benchmarking different cryptographic algorithms' performance on a specific hardware device. It does not involve social interactions or social research or interfacing with real people. The cryptographic algorithms are used to encrypt data and protect users' data integrity, confidentiality, authenticity, and anonymization. From an ethical point of view, this is beneficial for the users, especially when the IoT devices are used in domains like health care, home automation, and surveillance. The data stored on the IoT device or transferred between different IoT devices are sensitive and private. The security and privacy of data are very important for the users. Benchmarking cryptographic algorithms on ESP32 makes it easier to understand their performance and limitations on that device. By evaluating their performance on ESP32, developers can improve these cryptographic algorithms' reliability and better apply them in real-life applications.

From a sustainability point of view, choosing the right cryptographic algorithm to use in a specific situation is beneficial for the environment. Suppose the security level of the cryptographic algorithm run by the IoT device is unnecessarily high and exceeds the required range. In that case, the energy consumed by performing the tasks is wasteful. When the hardware runs with the optimal/right cryptographic algorithm for the tasks at hand, the waste of energy will be minimized. Even though a single IoT device's energy consumption is not much, but in the future, when there will be tens or even hundreds of these devices connected to each other, the energy consumption cannot be ignored. It is an important issue for a more sustainable future to find the right cryptographic algorithm for the right device to execute the tasks that need to be completed.

1.6 Research Methodology

In this thesis project, the quantitative methodology with a positivism assumption/paradigm will be used [7] since the aim is to gather data in numerical form, which can be measured in units of measurement and later on use those data to construct graphs and tables. The qualitative methodology is descriptive and regards phenomenon which can be observed but not measured, therefore not suitable for this thesis [7]. An experimental strategy fits the problem. By implementing a measurement program, the performance data such as each cryptographic algorithm's execution time will be collected and analyzed. The deductive approach is used to draw conclusions based on the collected data.

1.7 Delimitations

The ESP32's hardware acceleration feature only supports the most common cryptographic algorithms. In this thesis, only those supported algorithms will be taken into consideration. Other cryptographic algorithms or security methods will not be implemented since they would not be relevant and also it will be impossible to implement all of them due to the time limitation of this thesis project. Additionally, there is no accelerator for those algorithms on the ESP32.

1.8 Structure of the Thesis

Chapter 2 presents the theoretical background of cryptographic algorithms like AES, RSA, and SHA. Additionally, the features of ESP32 will also be presented. In chapter 3, the methodology and the experiment will be described. Chapter 4 covers all the different configurations of the cryptographic algorithms and how execution times are measured. The results are then compared, analyzed, and discussed in chapter 5. Finally, in chapter 6, the conclusions and future work suggestions will be presented.

2 Background

This chapter provides background information about different cryptographic algorithms and the ESP32's features. Symmetric and Asymmetric cryptographic algorithms that are studied in this thesis are presented in section 2.1 and section 2.2, respectively. Section 2.3 is about the hashing algorithm, and section 2.4 shows a summary of all compared algorithms. Related works are presented in section 2.5. In section 2.6, ESP32's features are shown.

2.1 Symmetric Cryptography

Symmetric cryptography can also be referred to as symmetric-key, secret-key, and single-key algorithm [8]. The same key is used to encrypt and decrypt messages between two users. For example, between user A and user B. User A want to send a message to user B but is afraid that someone else may read the message. By using a symmetric cryptographic algorithm, user A can encrypt the message so that no one can understand it. The process of encryption consists of running a message (input) through an encryption algorithm (a cipher) and generates a ciphertext (output). User B can decrypt the encoded message with the secret key shared with user A. Decryption is the inverse action of encryption. Figure 2.1 shows this symmetric-key cryptosystem. There are two variables, x and y in the figure, x is the message or plain text and y is the ciphertext. If the chosen encryption algorithm is strong, the encrypted text will look like random bits to adversaries and will not contain any useful information. The catch for this is the key needs to be distributed to both users through a secret and secure channel. The key only needs to be shared once between users and can then be used to secure the conversations among them.

Any cryptographic algorithm requires a multi-bit key to encrypt the data. The key size determines the difficulty of a brute-force attack. The difficulty will increase exponentially with longer keys. A brute-force attack involves systematically checking all the possible key combinations until the correct key is found. To break a 4-bit key, it will take a maximum of $2^4 = 16$ rounds to check all the possible key combinations. Start with "0000" and end with "1111", plus all the possible combinations in between. A key with a key length of 128-bit is computationally secure against brute-force attack. As of June 2020, the fastest supercomputer on the Top 500 supercomputer list is Fugaku in Japan [9]. Fugaku has a performance score of 415 PFLOPS, which is $415 * 10^{15}$ FLOPS (floating-point operations per second) [9]. To crack a 128-bit symmetric key with Fugaku it will take:

$$2^{128} \div 415 * 10^{15} \approx 8.19 * 10^{20} \text{ seconds} \approx 2.6 * 10^{13} \text{ years}$$

It is important to keep in mind that the time it is required for cracking the key is for checking all the possible combinations. In reality, it will probably take much less time if the correct key is in the first half of the possible combinations. This is an assumption in the worst-case scenario when all possible combinations must be checked to find the correct key. However, it still shows that it can take billions of years to crack a 128-bit symmetric key using the top supercomputer in the world. The longer the key is, the more secure it will be. A 256 bits key is highly secure and theoretically resistant to brute-force attack by quantum computers. For each bit increased, the difficulty of breaking the encryption through a brute force attack increases exponentially.

Symmetric algorithms are efficient to encrypt and decrypt the messages while at the same time, they provide high levels of security. Compared to asymmetric algorithms, it requires less computing power due to the relatively less complicated mathematical functions. While symmetric algorithms have many advantages, there is one major disadvantage. It is the problem with transmitting the secret key securely without being intercepted by the third party. To solve this problem, many developers are combining the symmetric and asymmetric algorithms to set up secure connections between users.

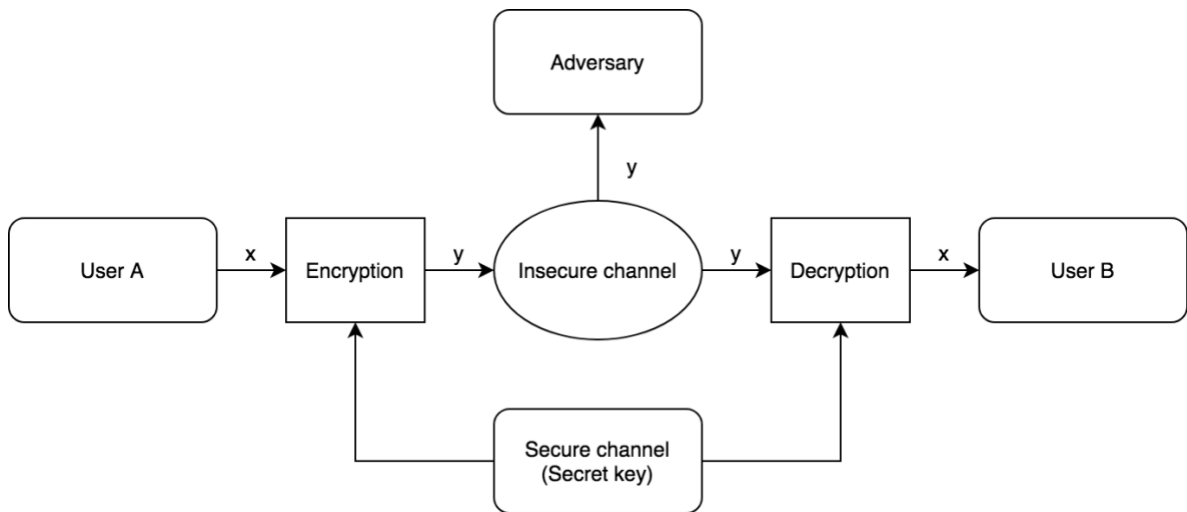


Figure 2.1: Symmetric-key cryptosystem

2.1.1 The Advanced Encryption Standard (AES)

The Advanced encryption standard is the most widely used symmetric cipher today. It is used in many commercial systems like Internet security standard IPsec, Transport Layer Security (TLS), the secure shell network protocol SSH (Secure Shell), the Wi-Fi encryption standard IEEE 802.11i and can be found in most modern applications and devices that need encryption functionality.

The AES was chosen by the National Institute of Standards and Technology (NIST) because of its performance on both hardware and software, simplicity of implementation and its security level. The all-around abilities of AES were the deciding factor why NIST chose this encryption algorithm.

The Rijndael block cipher is developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. They submitted their proposal [10] to NIST during the AES selection process [11], and later on, a part of the Rijndael block cipher was chosen by NIST to be the AES. The Rijndael is a family of ciphers with different key sizes and block sizes. The key sizes and block sizes can be specified in any multiple of 32 bits, with a minimum key size of 128 bits. The maximum for the block size is 256 bits, but there is no theoretical maximum for the key size.

AES is a subset of Rijndael block cipher, and NIST selected three members from the Rijndael block cipher family. They all have the same block size of 128 bits but different key sizes: 128 bits, 192 bits, and 256 bits. The key sizes specify the numbers of repetitions or “rounds” required to put the plaintext through the cipher and convert it to ciphertext. For a key of 128 bits, it will require 10 rounds, 12 rounds for a key of 192 bits and 14 rounds for a key of 256 bits. In essence, the 192 bits and 256 bits provide a higher level of security than the standard 128 bits. However, in the current applications, 128 bits is enough for most cases; unless it is highly sensitive data with an extreme threat level, then it should use 192 bits or even 256 bits. The higher security level provided by a more extended key size does come with a cost. With the extra two rounds for 192 bits and four rounds for 256 bits, the efficiency will drop by 20 or 40 percent, respectively.

A high-level description of how the AES algorithm works is presented in the following section. There are several key steps:

- dividing data into blocks
- key expansion

- add round key
- substitute bytes
- shift rows
- mix columns

Some steps will be executed multiple times during an AES encryption.

First, the data message is divided into blocks. For example, if the data message is “This is an example”, it will be divided into a four-by-four column. The first block looks like this:

T		a	x
h	i	n	a
I	s		m
s		e	p

If the message is longer than 16 bytes, the rest of the message will be divided into a new block.

The key expansion involves taking the initial secret key and using it to produce a series of subkeys for each round of the encryption process. This is done by using Rijndael’s key schedule [12]. In the first round, the initial secret key will be added to the data message’s first block. It might sound impossible to add characters together but keep in mind that all the process is done in binary. After this process, the first block of the data message might look like this:

s9	3e	e1	87
h4	69	8f	bt
x6	ej	v7	nm
gj	7h	zd	bb

Each byte in the block is substituted according to a predetermined table, AES S-Box [12]. For example, s9 in the block becomes jd. After this step, the block might look like this:

jd	wc	5j	nf
ut	om	bx	tr
7j	s9	sj	r5
aq	sb	w1	g8

Then the rows will be shifted. In this step, the second row is shifted one space to the left, the third row is shifted two space to the left, and the fourth row is shifted three space to the left. The block becomes like this:

jd	wc	5j	nf
-----------	----	----	----

om	bx	tr	ut
sj	r5	7j	s9
g8	aq	sb	w1

The major diffusion part of the AES algorithm is called mix columns. The mathematical details are a bit complicated. To simplify the explanation, this step uses mathematical equations on each column to further diffuse the block. The block might look like this now:

rn	g0	nw	el
gq	x1	fl	jt
x6	8c	bj	xw
j7	xy	ff	y1

After the mix columns step, a round key will be added to the block. Take the result from mix columns and add the first-round key derived from the initial secret key. This is the end of the first round and the beginning of the second round. As mentioned earlier, the AES algorithm has different key sizes, and each key size requires a different number of rounds. For example, AES-128 requires ten rounds. The AES encryption is done when it finishes all the rounds it is required. To make things clearer, Figure 2.2 shows the entire encryption process.

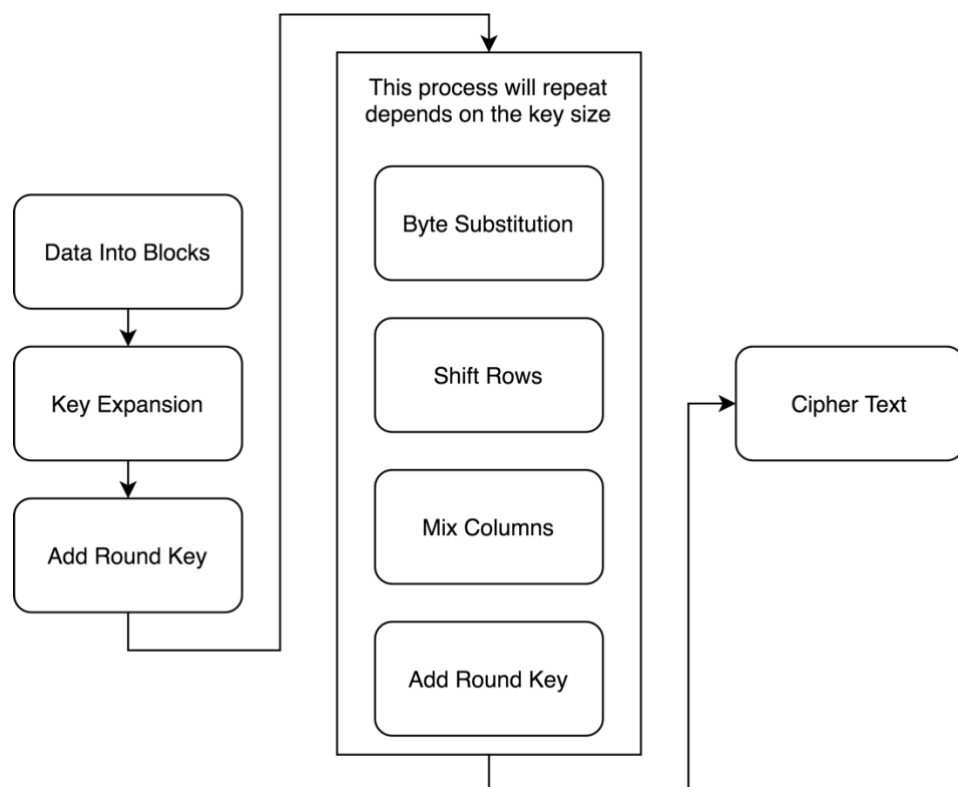


Figure 2.2: High-level AES encryption process

For AES decryption, the process is very similar to the AES encryption, but everything is done in reverse. It goes like add reversed round key, then the inverse shift rows, and the inverse byte substitution. Figure 2.3 shows the high-level decryption process. After the decryption process, the ciphertext will be transformed back to the original data message.

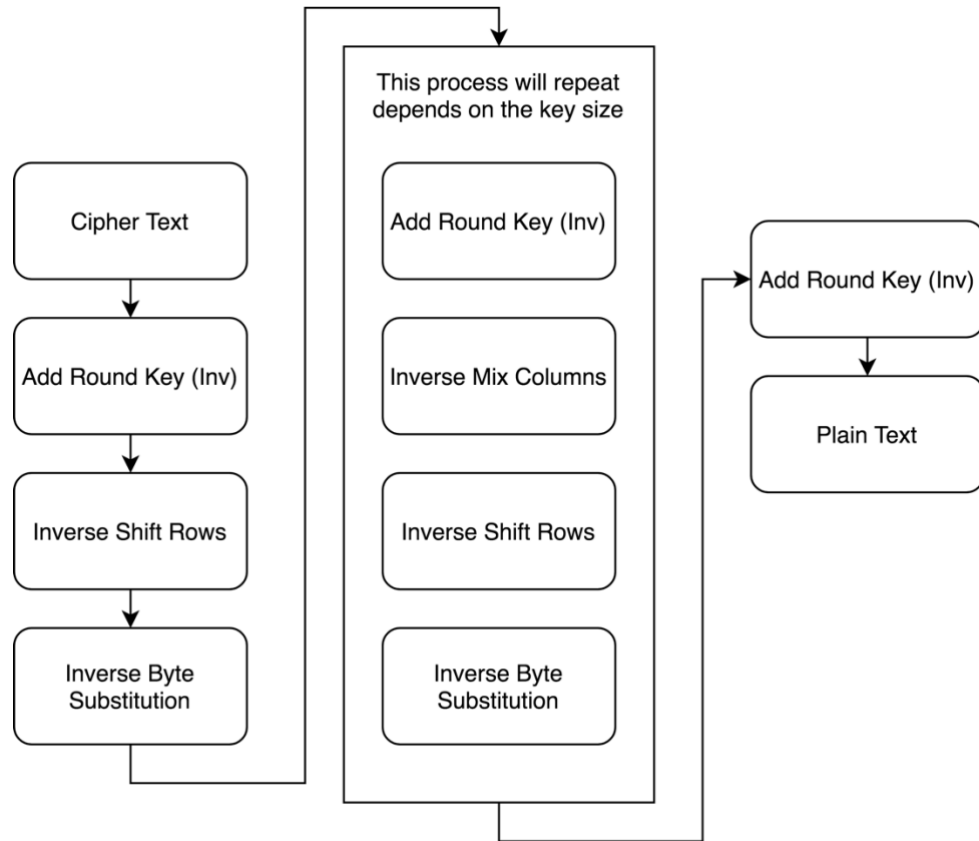


Figure 2.3: High- level AES decryption process

2.2 Asymmetric Cryptography

Asymmetric encryption is also known as public-key cryptography (PKC) [8]. It was introduced by Whitfield Diffie and Martin Hellman in 1976 [13]. In PKC, there are two keys, one is public, and the other is private. The public key is used for encryption and can be shared safely over any connection; the private key is used to decrypting information and needs to be kept secret. The asymmetric key pair is unique; data encrypted using a public key can only be decrypted by the person who has the corresponding private key. An example of user A sending a message to user B using an asymmetric cryptographic algorithm is shown in Figure 2.4. The asymmetric key pair are much longer than the key used in symmetric algorithms, and the longer key length makes it extremely difficult to compute the private key from the public counterpart. Those advantages solved the security issue that symmetric algorithms have with the key distribution.

PKC also made authentication more robust. User A can combine his/her message with his/her private key to create a digital signature on the message. User B can then take user A's public key to combine with the message to verify whether the signature is authentic (made by user A with the corresponding private key).

PKC does have limitations; one of them is the complex mathematical operations involved in encryption and decryption. Compared to symmetric algorithms, PKC is much slower when dealing with larger data and requires more computational power. The other limitation is the same as symmetric algorithms, the security with the secret key. If the private key is exposed or shared by accident, the security of all encrypted messages will be compromised. Also, losing the private key will make it impossible to decrypt data that was encrypted using the corresponding public key.

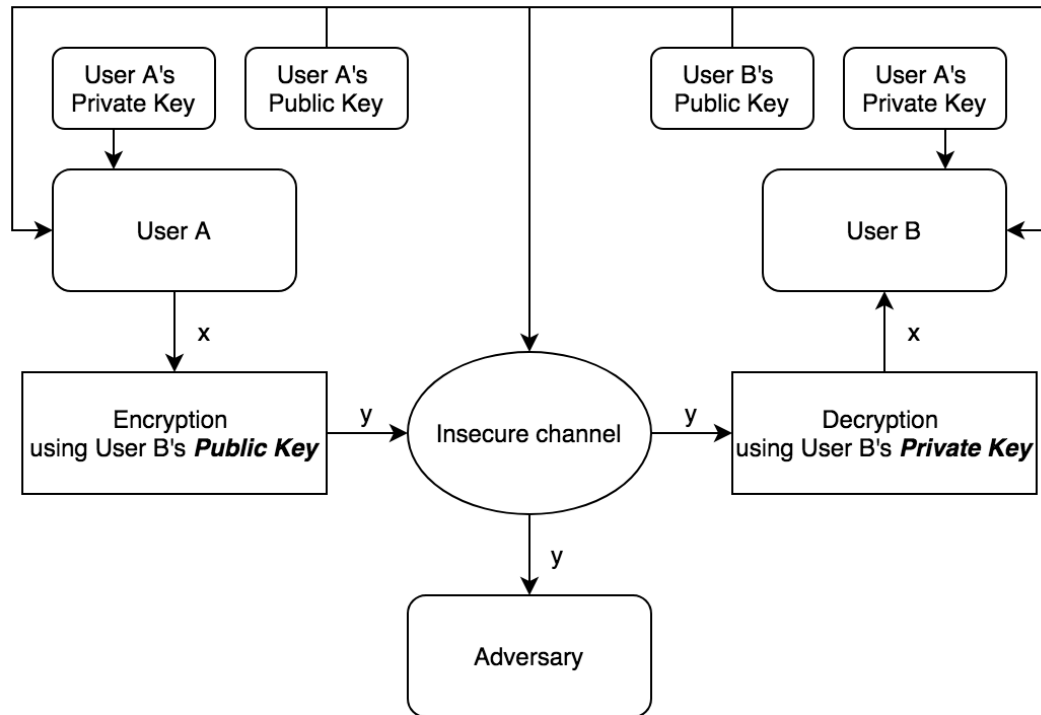


Figure 2.4: Asymmetric-key cryptosystem

2.2.1 RSA

The RSA algorithm was developed by Ron Rivest, Adi Shamir, Len Adleman in 1977. An English mathematician called Clifford Cocks who works for the British intelligence agency Government Communications Headquarters (GCHQ) had developed an equivalent system in 1973, but it was kept classified until 1977 [14]. RSA is the most used public key cryptography algorithm for secure data transmission, because it can encrypt messages without the need to secretly exchange the key separately. The key sizes for the RSA algorithm are 1024 bits, 2048 bits, and 4096 bits.

RSA is one of the first public-key cryptographic algorithms. As explained in the previous section, users can send messages without the initial step of delivering the secret key to all users using the RSA algorithm. Instead, each user has their own key pair, including a public key and a private key. The message encrypted with a public key can only be decrypted by the corresponding private key.

RSA is a relatively slow and resource-heavy algorithm, and because of this, it is not commonly used in direct encryption of larger data blocks. It is often used to encrypt the secret keys for symmetric key cryptography, which in turn can encrypt/decrypt larger data sets in a more efficient way. There are many applications for RSA, but it is mostly used in two areas. One is encrypting

small data like keys for key transport, the other is digital signatures for digital certificates used to authenticate message senders and receivers on the internet.

There are several key concepts in the RSA algorithm. Those include trapdoor functions, generating primes, Carmichael's totient function, generating primes, and the process of calculating the public keys and private keys.

Trapdoor functions are easy to compute in one direction but almost impossible in reverse without knowing certain information. The RSA algorithm uses the primes as the trapdoor function. For example, it is hard to figure out which two prime numbers' product is 7081. However, it is not so hard to calculate $73 \times 97 = 7081$. Another interesting aspect of this trapdoor function is that if one of the prime numbers is already obtained, it is simple to figure out the other one. Just divide the product with one of the prime numbers to obtain the other prime number. In the real-life implementation, much larger prime numbers are used for the RSA algorithm. This forms the basis of how public key and private key encryption scheme works, allowing the public key to be shared without risking the data message's security.

Generating primes (key pair) is the first step of encrypting a data message with the RSA algorithm. Two very large prime numbers (p and q) are selected with a primality test, by an algorithm that efficiently finds prime numbers. The prime numbers need to be larger and far apart to make it more secure against attacks. After the prime numbers are selected, the modulus n will be discovered which is the product of the two prime numbers. In this example $n = 73 \times 97 = 7081$.

Carmichael's totient function is used on the discovered modulus n . The function is shown below:

$$\lambda(n) = lcm(p - 1, q - 1)$$

There $\lambda(n)$ is the Carmichael's totient of n and lcm means the lowest common multiples. The Carmichael's totient of 7081 is $\lambda(7081) = lcm(96, 72) = 288$.

Using the Carmichael's totient value calculated before, a number e will be determined. The number e can be any number between 1 and $\lambda(n)$, which in this example is 288. Additionally, the number e needs to satisfy another condition, which is:

$$gcd(e, \lambda(n)) = 1$$

Where gcd means greatest common divisor. The number e does not need to be randomly picked, and a larger number e may cause inefficient encryption time.

The public key of the RSA algorithm is made up of the number e and the modulus n . Using the public key and the following equation, the data message x can be transformed into a ciphertext c :

$$c = x^e \bmod n$$

Where \bmod refers to a modulus operation.

To decrypt the ciphertext c back to the data message x , the corresponded private key must be generated. To generate the private key d , the following equation is needed:

$$d = 1/e \bmod n$$

Where $1/e$ is not one divided by e but the inverse of e . The inverse of e can be calculated using the extended Euclidean algorithm. After obtaining the private key, the ciphertext can then be decrypted back into the data message using the following equation:

$$x = c^d \bmod n$$

In real-life RSA implementation, the data message will be padded before it gets encrypted. Padding is a way to make a data message hide from the original format in order to keep the data message more secure and prevent the adversaries from figuring out the structure of the data message. In RSA, when a data message is padded, randomized data is added to the data message to hide the formatting clues and prevent the encrypted message from being cracked. Adding this padding before the data message is encrypted makes the RSA algorithm more secure.

To summarize everything, Figure 2.5 shows the high-level process of how the data message is encrypted and decrypted using the RSA algorithm.

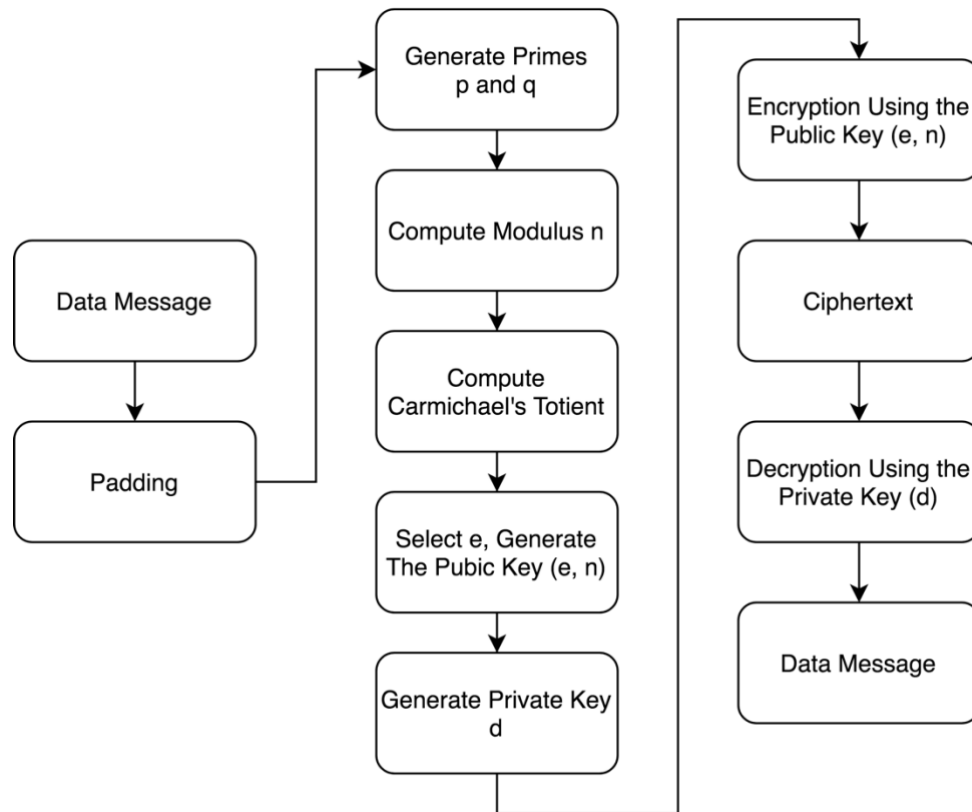


Figure 2.5: High-level Process of the RSA Algorithm.

2.3 Hashing

Hashing is the transformation of the original message of data into a shorter fixed-length value called hash value using mathematical functions. No matter how big the original data is, the hashed version of it will always be in fixed length. It is faster to find items using shorter hashed value than the

original data message, that is why hashing is commonly used in indexing and retrieving items. Hashing is also used in encrypt and decrypt digital signatures. User A using a hash function to transform the digital signature into a hash value known as message-digest. Then user A sends the digital signature and the message-digest to user B in separate transmissions. User B uses the same hash function to derive a message-digest from the received digital signature and compares it with the one from user A. Both message-digest should be the same to prove the authenticity of the message. Figure 2.6 shows this operation's process; here value x is the message sent by user A; values A and B are message-digests from user A and user B, respectively. Hashing is a one-way operation, which means there is no way to reverse the hashed value back to the original data message. A good hash function should also not produce the same hash value known as a collision from two different input data messages.

Hashing is also used in many encryption algorithms; known hash functions include the message-digest hash functions MD2, MD4, MD5, and Secure Hash Algorithm (SHA), SHA-1, SHA-2, SHA-3. There will be more details about SHA-2 since it is one of the algorithms that is going to be evaluated in this thesis project.

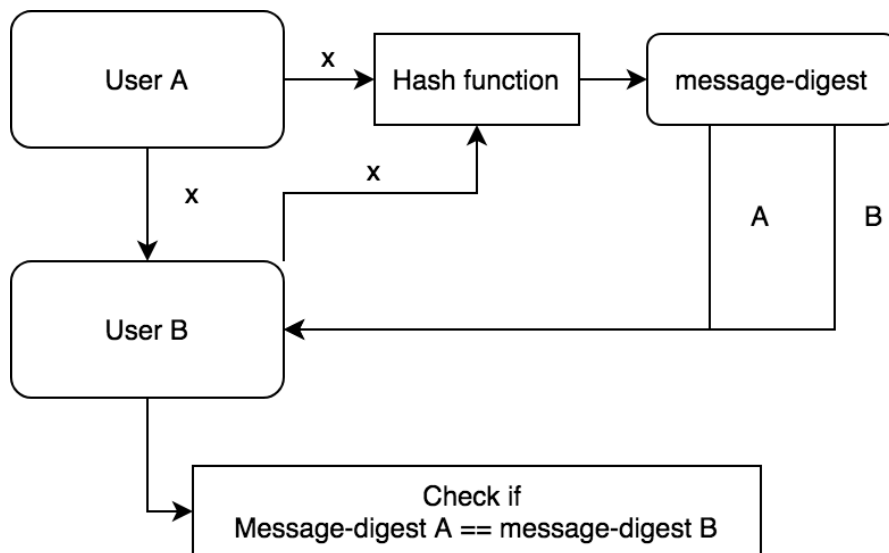


Figure 2.6: Process of digital signature using hash function

2.3.1 Secure Hash Algorithms (SHA)

The Secure Hash Algorithms are a family of cryptographic functions that are designed for data security and is published by the NIST as a U.S Federal Information Processing Standard (FIPS) [5]. The SHA family includes SHA-0, SHA-1, SHA-2, and SHA-3. These algorithms are designed to be a one-way function making it nearly impossible to decrypt/transform it back to the original message from its hash value. SHA is commonly used in encrypting passwords; instead of keeping a user's actual password, the hash value will be stored. This proves to be rather effective against adversaries who attack databases, as they will only get the hash values of the actual passwords. Additionally, SHA is also good to detect data's tempering and hiding information regarding the original data message because a small change in the original data message will produce a big difference in the hash value. The hash value is also in fixed-length rather than the length of the original data message.

SHA-2 is published in 2001 [15] and is more secure than its predecessor SHA-1. It contains two hash functions, SHA-256 using 32 bits words and SHA-512 using 64 bits words. There are also four truncated versions of SHA-256 and SHA-512, which are SHA-224, SHA-384, SHA-512/224, and SHA-512/256. The number after SHA is the size of the digest (the hash value) in bits. The bigger the digest, the higher the level of security it provides. The brute-force attack is much less effective against SHA-2 than SHA-1. Using brute force to find a message that corresponds to a given digest of length S would require 2^S evaluations. For SHA-256, the number of evaluations will be 2^{256} . This makes SHA-2 much safer against this kind of attack than SHA-1 since its digest size is only 160 bits.

The SHA-2 hash function is implemented in many security applications and protocols like TLS, SSH, IPsec, Secure Sockets Layer (SSL), Pretty Good Privacy (PGP), and Secure/Multipurpose Internet Mail Extensions (S/MIME).

The process of the SHA-1 algorithm is simple. It contains two main steps. These steps are padding and compression function. First, the data message is padded to fit the input format, which has a size of a multiple of 512 bits. Then the padded data message is divided into 512-bit blocks. Each 512-bit block is further divided into 16 words of size 32 bits. There is also an initial value H_0 ; it contains five predefined 32-bit words. This value is used as part of the input for the compression function, and it can determine the length of the output of the compression function. Then all the padded message blocks are going through the compression function. The compression function consists of 80 rounds, which is divided into four stages of 20 rounds each. The output of the last iteration of the compression function will be the hash digest. In this case, the hash digest has a length of 160 bits. Figure 2.7 shows the high-level diagram of SHA-1.

SHA-2, as the successor of SHA-1, has a similar internal structure. The differences are the length of the input is longer for SHA-384 and SHA-512 in the SHA-2 family, which is 1024 bits, double as long as its predecessor SHA-1. The hash digest is longer as well, which goes from 224 bits to 512 bits. Additionally, the number of rounds for the compression function is different for the different variant of SHA-2. For SHA-224 and SHA-256, the number of rounds is 64, and for SHA-384 and SHA-512, the number of rounds is 80. [12]

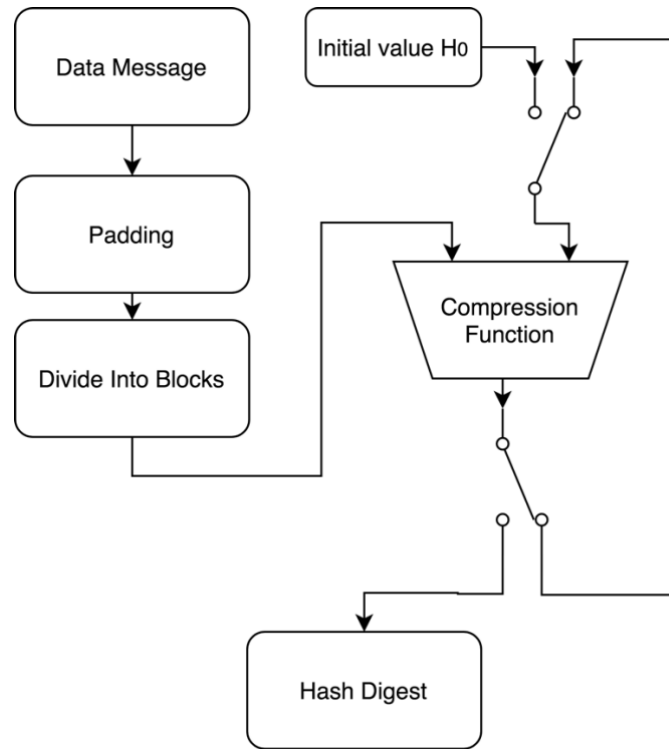


Figure 2.7: High-level Diagram of SHA-1

2.4 Summary of Selected Algorithms

This section summarizes the properties of all three algorithms to view their differences quickly. Information like the type of cryptography, key size, and speed are shown in table 2-1.

Table 2-1: Properties of AES, RSA and SHA.

	Type	Key sizes	Rounds	Speed
AES	Symmetric cryptography	128,192,256 bits	10, 12, 14	Fast
RSA	Asymmetric cryptography	1024, 2048,4096 bits	1	Slow
SHA-2	Hash function	Keyless	64, 80	Very Fast

2.5 Related Work

This section presents related work in the performance analysis of different cryptographic algorithms. The results obtained from those researches are summarized and discussed.

2.5.1 Security Concerns and Cryptography of IoT System

An IoT system can be divided into four different layers: perception layer, network layer, middleware layer, and application layer. Farooq et al. provided an overview of possible threats and scenarios on all four layers [16]. In conclusion, the majority of threats are found in the network

layer. Trade-offs like encryption/decryption speed, energy consumption, and the level of security have to be made to lower the risk of threats in the network layer.

In [17], Arsalan Mosenia and Niraj K. Jha investigated IoT security. They summarized a comprehensive list of vulnerabilities and countermeasures against them on three different levels which are: edge nodes, communication, and edge computing. The conclusions are that the security threats are not well-recognized in the IoT domain, and these threats should be addressed proactively and aggressively by researchers and manufacturers.

In [18], Nicolas Sklavos and I. D. Zaharakis collected models, schemes, and implementation aspects regarding IoT's alternative technologies and devices. Flexible solutions are proposed for efficient optimizations of the operation framework of trust management. They state that most current cryptographic models and security schemes have been designed for general purpose uses. Detailed analysis is needed in order to ensure those models can be implemented in resource-constrained IoT devices.

2.5.2 Performance Analysis of Symmetric Key Algorithms

Today, data security is an important aspect of data transmission. Cryptographic algorithms help to improve data security, authenticity, confidentiality, and integrity. There are many different cryptographic algorithms. In order to find a suitable algorithm that is efficient, robust, high performance, and satisfies the user's requirement, a performance analysis is needed. The paper wrote by Vyakaranal Shashidhara, and Kengond Shivaraj reveals that the AES algorithm performs well in overall performance analysis among symmetric key algorithms [19].

2.5.3 Performance Analysis of Cryptographic Algorithms on Resource Constraint Devices

Performance evaluation of popular symmetric and asymmetric key encryption algorithms to selecting better utilization on resource constraint devices is conducted by Hague Md Enamul et al., in their paper published in 2018 [20]. The symmetric key algorithms they focused on were AES, RC4, Blowfish, CAST, Twofish, and 3DES. For the asymmetric key algorithms, DSA and ElGamal were chosen for this evaluation. They also evaluated hashing algorithms like SHA-1, SHA-256, MD2, MD5, etc. The performance is mainly comparing the encryption and decryption speed and the memory usage on text files. They concluded that DES3, Twofish, and RC4 outperform other selected algorithms in terms of encryption/ decryption time. Additionally, they also state that increasing the key size is a time-consuming approach for data encryption and decryption, which leads to higher energy consumption.

2.5.4 Comparing Elliptic Curve Cryptography and RSA

Strong asymmetric key algorithms are often considered to be too computationally expensive for small devices if not accelerated by cryptographic hardware. In [21], Nils Gura et al. compared two asymmetric key algorithms, elliptic curve cryptography (ECC) and RSA, on 8-bit CPUs. They implemented and evaluated elliptic curve point multiplication for 160-bit, 192-bit, 224-bit, and RSA 1024 and RSA 2048 on two 8-bit microcontrollers. As a result, on small devices, ECC point multiplication becomes more comparable in performance to RSA operations, especially when the key's size becomes large.

In [22], Manuel Suárez-Albela et al. compared two of the most used TLS authentication algorithms i.e. ECDSA and RSA using a resource-constrained IoT system based on the ESP32 System-on-Chip. The security level, energy consumption, and the average time per transaction are measured and compared between those algorithms with various levels of security. The result shows that the ECDSA algorithm outperforms RSA in all aspects. However, a real-world scenario test is

still needed to find the best security configuration for the hardware platform. Also, software optimizations and implementation can play a significant role where the same algorithm with a higher security level has better performance than the one with a lower security level in terms of energy consumption and throughput. The results also showed that due to optimization made on the libraries for ECC operations, the curve secp256r1 outperforms the curve secp224r1 while providing a higher security level, lower energy consumption, and higher throughput.

2.5.5 Performance Analysis of Security Algorithms for IoT Devices

IoT ecosystem is growing and has gained much attention in today's world. However, the security of IoT device's is at risk. Most security algorithms are not compatible with the resource-constrained IoT devices. In [23], Nuzhat Khan et al. compared several security algorithms performances, mainly the execution time and processing cycle in Raspberry Pi. Two different cryptographic libraries, FLECC_IN_C and Crypto++ were used, and their performance was measured in a constrained environment. The result showed RSA 1024 and LUCDIF 512 are the algorithms that performed the security routine in different tests fastest. Additionally, an algorithm like ECMQVC takes far more time to process than the other algorithms.

2.5.6 Analysis of Cryptographic Algorithms on Raspberry Pi

Traditional cryptographic algorithms might not be suitable to be implemented on small IoT devices with limited resources. The tradeoff between performance and security is the main focus when utilizing a certain cryptographic algorithm on IoT devices. A benchmark of most known algorithms on the Raspberry Pi 3 model B is presented in [24]. The authors in this paper provide measurements of four different types of cryptographic algorithms, symmetric block ciphers, hashing algorithms, key generation/ exchange algorithms, and digital signature algorithms. Both the energy performance and the throughput were measured. The conclusions are that the best symmetric algorithm for Raspberry Pi 3 is AES. Hashing algorithms presented a similar throughput as symmetric algorithms. In key generation/exchange algorithms, Elliptic-curve Diffie–Hellman (ECDH) is better than RSA. Last, the digital signature algorithm, Elliptic Curve Digital Signature Algorithm (ECDSA) is better in signing than RSA and DSA, while RSA is the best when it comes to verification.

2.5.7 Attacking the ESP32 IoT Devices

In [25], Oleksii Barybin, Elina Zaitseva, Volodymyr Brazhnyi implemented an IoT system based on ESP32 to measure the temperature and send the data to a server through Wi-Fi. The experiment's primary goal is to attempt to gain unauthorized access to the network, create a fake ESP32 client and disconnect the original ESP32 from the server and then send fake data to the server. The result showed that with basic knowledge and skills in common wireless hacking tools such as Airmon-ng, Aircrack-ng, Airodump-ng, Wireshark and basic knowledge of ESP32, one can access the system and send fake data to the server. The conclusion is to use TCP instead of UDP to reduce the probability of the proposed scenario.

2.5.8 Secure Data Communication

The Internet of Things has high-security requirements because most of the information exchanged between IoT devices is sensitive, critical, and private. Symmetric or asymmetric cryptography or a combination of both can be used to secure the communication between IoT devices. Michelle S

Henriques and Prof. Nagaraj K. Vernekar proposed a schematic consisting of symmetric and asymmetric cryptography to secure the communication between the gateway and the IoT devices in [26]. The combination of both symmetric and asymmetric cryptography improves the performance compares to only using the asymmetric algorithm in terms of encryption time. The key for symmetric cryptography is randomly generated using the timestamp as a seed. This further improves the security aspect of the IoT system.

In [27], Quist-Aphetsi Kester et al. proposed a scheme for securing communication among IoT devices. Both authentication of the nodes and the security of the messages transmitted between the source node and the node are accomplished. This is done using the Diffie-Hellman algorithm for the key-sharing and authentication among IoT nodes, AES for encrypting the transmitted messages, and MD5 for validation of those messages.

In [28], Irfan A. Landge and Hannan Satopay proposed a system that allows the use of MD5 for securing the IoT devices from attacks when connecting to the internet or intercommunicate with each other. MD5 is a hash algorithm that will transform a data message with an arbitrary length into a message digest with a fixed length. The proposed system uses both the password and a Unix timestamp to make the message digest unique, thereby making the system impossible to penetrate through a brute force attack. However, due to the collision attack, the MD5 algorithm has been replaced with better hash algorithms such as SHA after 2004.

2.5.9 Performance Evaluation of Cryptographic Ciphers on IoT Devices

The advent of IoT and the increasing use of application-based processors leads to a greater security risk as IoT devices thrive in an eco-system of co-existence and interconnection. It is essential to test the existing cryptographic algorithms on IoT devices and determine if they are viable in terms of execution time and memory consumption. In [29], Kedar Deshpande and Praneet Singh tested various symmetric key algorithms with various block sizes and key sizes on two IoT devices which are the Raspberry Pi 3 and Beagle Bone Black. A comparison of the execution time between these IoT devices for a variety of different data sizes was created. The conclusion is that the Raspberry Pi 3 performs better than Beagle Bone Black on execution time, power, and memory consumption. However, the Beagle Bone Black has better available functionality with its replete GPIO pins to add/modify interfaces.

2.5.10 Security and Performance in IoT

The prediction shows there will be over 18 billion IoT devices by 2020. Performance and security on these devices are crucial and balancing between performance and security is also necessary. In [30], Luke E. Kane et al. contributed a framework to measure the performance of cryptographic algorithms on IoT devices. Power consumption, time cost, energy cost, and random-access memory (RAM) usage and flash usage are the measured areas. An insightful comparison of the performance of the ATmega328, STM32F103C8T6, and ESP8266 is presented as well. Three cryptographic algorithms were measured on those IoT devices: AES, ChaCha, and Acorn. The results show that the ESP8266 was the fastest performing device, but the STM32F103C8T6 device has the best overall energy cost while still performing well in terms of execution time. Both ChaCha and Acorn are faster and consume less energy than AES on three devices and should be considered for lightweight encryption uses.

In this thesis, the symmetric algorithm AES, asymmetric key algorithm RSA and hashing algorithm SHA will be evaluated on ESP32. The performance evaluation is mainly focused on the encryption/ decryption time of data messages of various sizes. The impact on the performance of selected algorithms when using cryptography hardware acceleration is also evaluated.

2.6 ESP32

ESP32 is a series of low-cost, low power System-on-Chip microcontrollers with integrated Wi-Fi and Bluetooth. ESP32 is created and developed by Espressif Systems [31] (a Chinese company based in Shanghai) and manufactured by TSMC (Taiwan Semiconductor Manufacturing Company) using their 40nm process. The ESP32 WROVER-KIT V4.1 [32] is used in this thesis, and it is a development board build around ESP32. The SoC on the ESP32 WROVER- KIT is an ESP32-DoWDQ6, and it contains two low-power Xtensa 32-bit LX6 microprocessor with a clock frequency of up to 240Mhz. The ESP32 WROVER-KIT V4.1 also features support for an LCD and a MicroSD card [33]. The main components and their interconnections of the ESP32 WROVER-kit V4.1 are illustrated in the block diagram in Figure 2.8. ESP32's function block diagram is shown in Figure 2.9. The cryptographic hardware acceleration part shows the algorithms it supports; those are the algorithms evaluated in this thesis project. The Wi-Fi blocks are useful for the case study about secure communication with a remote database.

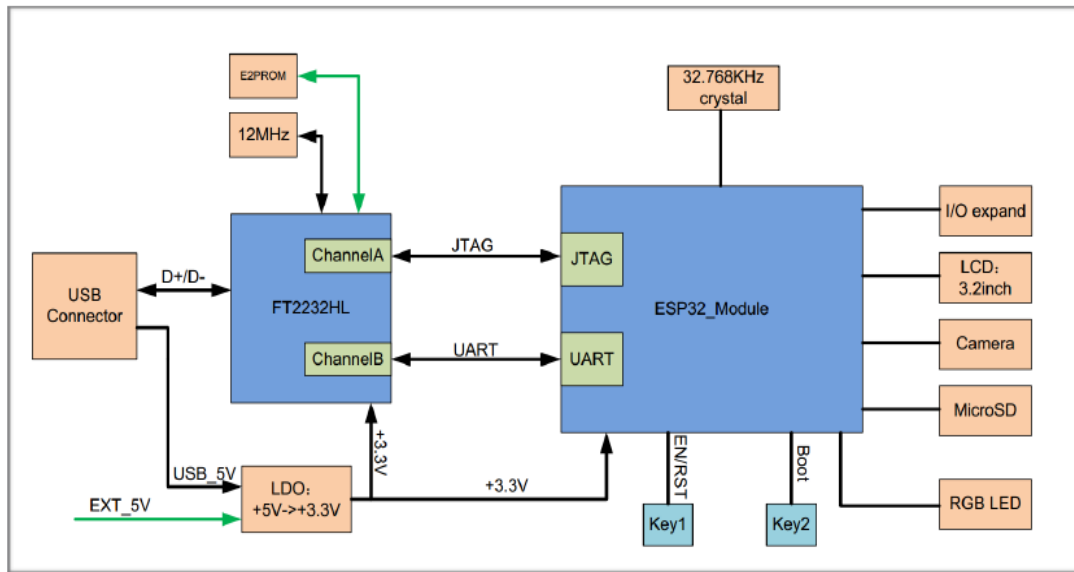


Figure 2.8: ESP-WROVER-KIT block diagram [34].

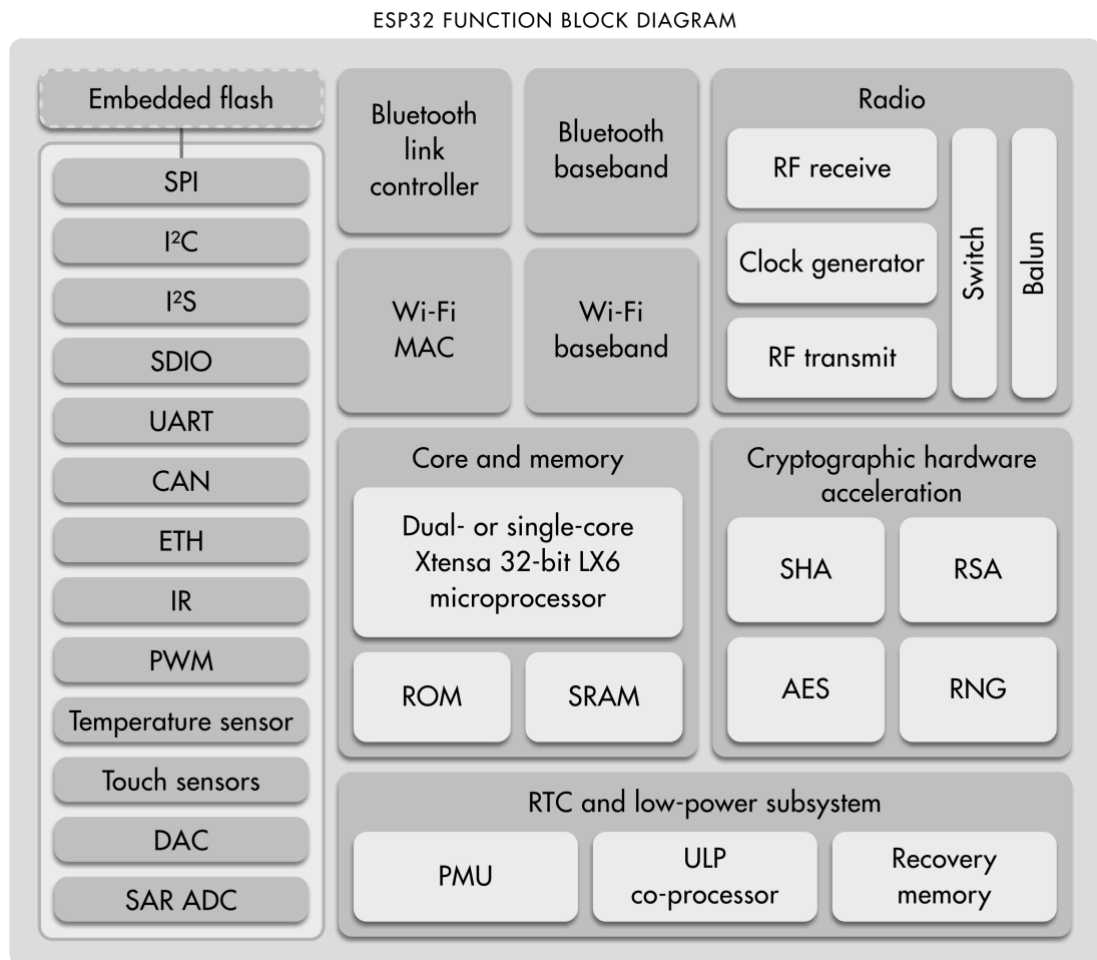


Figure 2.9: Function block diagram of ESP32 taken from [35].

3 System Design and Methodology

The purpose of this chapter is to provide an overview of the research method used in this thesis. Section 3.1 describes the research process. Section 3.2 details the research paradigm. Section 3.3 focuses on the data collection techniques used for this research. Section 3.4 describes the experimental design. Section 3.5 explains the techniques used to evaluate the reliability and validity of the data collected. Section 3.6 describes the data analysis methods that will be used in the thesis project. A summary of the entire chapter is presented in section 3.7.

3.1 Research Process

In this section, the research process is shown. Figure 3.1 presents the diagram showing each step. First, the research problem is identified, and a literature study is carried out by reading research papers and books in fields related to this thesis project. A suitable data collection method is chosen, and a large set of data is collected by using the implemented measurement program. The data is then analyzed and evaluated to conclude the research question. Key steps are explained in detail in the subsections below.

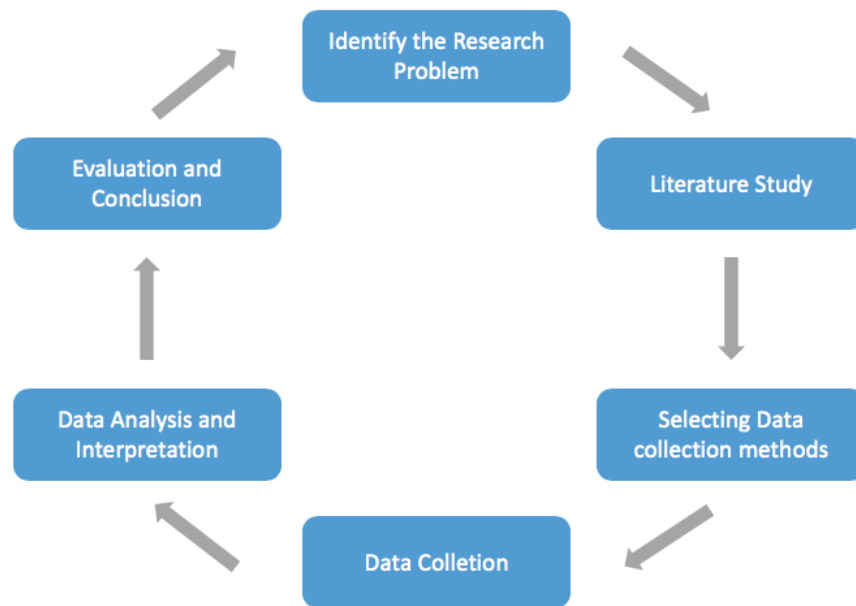


Figure 3.1: Steps for the research process

3.1.1 Research Methodology

The quantitative research methodology supports experiments by measuring variables to verify or falsify theories and hypotheses. This methodology also requires a large set of data and the use of statistics to test the research question and make the research project valid. For this thesis project, the quantitative methodology is chosen as the research methodology since it is about the performance of different cryptographic algorithms on a specific hardware platform. There are many numerical measurements.

3.1.2 Development Environment

Different cryptographic algorithms' performance on the ESP32 WROVER-KIT V4.1 will be evaluated in this thesis project. To develop applications for ESP32, we need to set up the toolchain for building applications, getting Espressif IoT development framework (ESP-IDF), which contains all the API and scripts, and installing Eclipse, an IDE for write programs in C. The guide to getting started with ESP32 application development is provided by Espressif [36]. Figure 3.2 shows how each step described is linked to each other. As shown in the figure, the project is built using ESP-IDF and the toolchain. Later on, the application is built by using make or Eclipse. The application will be uploaded through the USB connection to the ESP32.

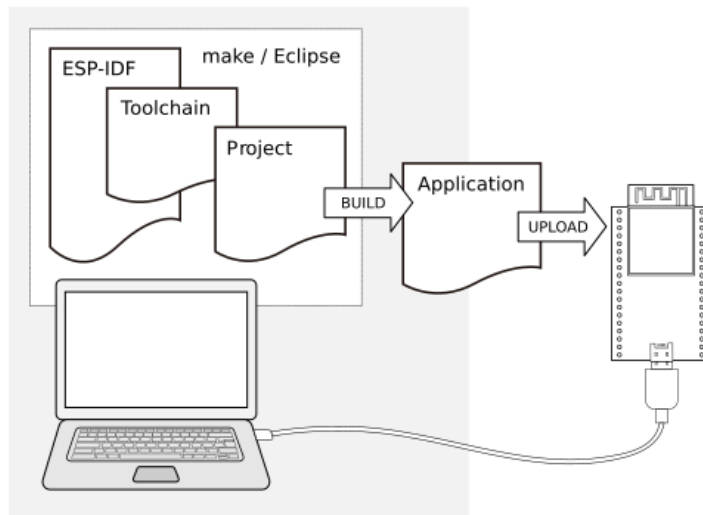


Figure 3.2: Development of applications for ESP32 [36].

3.1.3 Implementation of Measurement Program

To measure the performance of those cryptographic algorithms, a measurement program needs to be implemented. The measurement program needs to have several functions, such as:

- A pseudorandom data message generator to generate input data messages.
- A timer to collect the execution time of those algorithms.
- One or several loop functions to generate more data samples.

The measurement program also needs to execute the algorithms correctly. It should be able to encrypt the pseudo-randomly generated data messages and later decrypt them back to the original data message, so the execution time of the algorithm collected is accurate and reliable.

3.1.4 Data Collection

The execution time of the encryption and decryption of all chosen algorithms will be measured through the measurement program. The ESP32 WROVER-KIT V4.1 will also run on different CPU frequencies, 160Mhz, and 240Mhz to evaluate the impact of the CPU frequency on the cryptographic algorithms' performance. Since the program is run on the ESP32 WROVER-KIT V4.1, to check if the measurement program is measuring the execution time of the selected cryptographic algorithm and the cryptographic algorithm being measured is working properly i.e. encrypting the data messages or decrypting the ciphertext. The board needs to be connected to the computer with a USB cable and using the IDF monitor application to see the program is running without any errors.

The measurement program will print the execution time of the specified algorithm for each iteration, and the result is shown in the terminal. Additionally, the pseudo-randomly generated data messages, and the decrypted data messages will be presented in the terminal to show the cryptographic algorithms are successfully encrypting and decrypting the data.

3.1.5 Data Evaluation

The execution times will be analyzed and evaluated after the collection. All the raw data will be analyzed using MATLAB. Graphs can show patterns and relations between data size, execution time, and how well those algorithms perform on the ESP32 WROVER-KIT V4.1.

3.2 Research Paradigm

The term paradigm was first used by Kuhn in his work [37]. He defined the research paradigm as “an integrated cluster of substantive concepts, variables, and problems attached with corresponding methodological approaches and tools”. A research paradigm is an approach on how to conduct research. There are three major paradigms: positivism, interpretivism, and critical theory [38]. For this thesis is most appropriate to use the positivist paradigm. More details about the positivist paradigm and why it is chosen for this thesis project are shown in the subsection.

3.2.1 Positivist Paradigm

Positivism assumes that reality is objectively given and exists independently of humans. The researchers usually test the theories in a deductive manner to increase the predictive understanding of the phenomenon. This paradigm is most used in projects that are of experimental and testing character. Research that uses the positivist paradigm often generates numerical data. They draw conclusions about a phenomenon through quantifying measurements of variables and testing hypothesis [7]. Another paradigm like interpretivism is about assuming that reality is accessed only via social constructions. They try to explore richness, depth, and complexity in an inductive way to understand the phenomenon and discover the meaning that people give it. Often used in projects with opinions, perspectives, and experiences characters to get context for phenomenon [7]. The positivist paradigm is more suitable for this thesis project since it is about drawing conclusions through quantifying measurements of variables and not about opinions or perspectives.

3.3 Data Collection

Data collection methods are used to collect information from all the relevant sources to find answers to the research problem. It is divided into two categories: primary and secondary data collection methods. Primary data collection methods are about collecting new data and can be divided into two groups: quantitative and qualitative. The quantitative data collection methods involve numbers and mathematical calculations. On the contrary, qualitative data collection methods do not involve numbers or mathematical calculations. Instead, it is associated with words, languages, emotions, and other non-qualifiable elements. Secondary data collection methods are about using the already existing data from published books, journals, research papers, and etc. It shows that the primary quantitative data collection methods are most suitable for collecting data for this thesis.

3.3.1 Sampling

For the AES algorithm, the data samples need to be in a specific size because the algorithm can only encrypt 16 bytes each time. Other algorithms like RSA and SHA-2 will use the same data size as AES for convenience. The data reliability and viability are essential for this thesis project. To make data

quality better and lower uncertainties in the measurement, each data point will be measured multiple times.

3.3.2 Sample Size

For AES, the sample size can be quite large. The input data messages used in the AES algorithm are between 16 bytes and 1600 bytes, there are 100 steps and the size of each step is 16 bytes. The same goes for SHA-2, and each data point will be executed and measured 500 times. RSA is not designed for encrypting large data files, and it is much demanding on the hardware and cannot run as many data points as the AES algorithm. Additionally, the max data size that can be encrypted using RSA is restricted by the key length (1024 bits = 128 bytes and 2048 bits = 256 bytes). The number of data points has then been decided to be 7 and 15 for key length 1024 bits and 2048 bits, respectively. Each data point will be executed 500 times, and the execution time of encryption and decryption will be measured separately. A summary of the number of data points and the data sizes of all measured algorithms is shown in Table 3-1.

Table 3-1: Summary of the number of data points and the data sizes for all measured algorithms

	Number of data points	Size of each step (bytes)	Data size range (bytes)	Samples of each data point
AES	100	16	16-1600	500
SHA-2	100	16	16-1600	500
RSA 1024	7	16	16-112	500
RSA 2048	15	16	16-240	500

3.4 Experimental Design

This section explains the evaluation environment and the structure of the measurement program. More details of the hardware and the software to be used during this project will be presented as well.

3.4.1 Evaluation Environment

The measurement program will generate pseudo-random data messages for the cryptographic algorithms to encrypt/decrypt, and it then measures the execution time of the encryption/decryption. In the measurement program, there will be a loop. Each iteration of the loop will increase the size of the input data by 16 bytes, and then the data will be encrypted by the specified algorithm. Later the encrypted text will be decrypted. To make sure the algorithm works properly, the input text and the decrypted text will be compared with each other. The execution time of both encryption and decryption will be measure separately, and the result will be collected. A flow chart illustrates the process is shown in Figure 3.3. The program will first check if $i < 500$ because each data point will be measured 500 times. Then, it checks if $j < X$, The X (in the condition $j \leq X$) is the number of data points, and its value varies between 7, 15, or 100 with the corresponding algorithm. The PRNG will generate a pseudo-random data message with a length of 16 bytes times j . The corresponding cryptographic algorithm will encrypt the data message, and afterward, the encrypted data message will be decrypted. The execution time of encryption and decryption will be measured, respectively. The I/O data stings are compared to ensure the encryptions/decryptions

are performed correctly. The execution time will then be stored. After the execution time is stored, the j will increase by 1, and the same process will be performed until all the data strings are checked, and the execution times are collected.

The data input is generated by a pseudo-random number generator (PRNG) [39], and it is seeded with a seed value of 128. The PRNG generates random numbers that are used to build the data messages with a given size. In this project, it is 16 bytes or multiples of 16 bytes; each byte will be randomly selected between 0 to 9 and all lowercase alphabets. One of the reasons to choose PRNG is to conduct the measurement with random data messages to eliminate the possibilities that certain combinations of data may affect the performance. The other reason to use PRNG with a seeded value is to reproduce the same experiment results possible.

3.4.2 Hardware and Software Environment

The hardware used in the project is an ESP32 WROVER-KIT V4.1 [34]. Section 2.5 presented more information about the ESP32 WROVER-KIT V4.1. One of ESP32 WROVER-KIT's many features, the MicroSD card feature is useful in this project since the pre-generated key files for the RSA algorithm will be stored in the SD card.

For the software, The ESP-IDF SDK v.3.3 [40] is used for programming the ESP32 module and Eclipse is used as the IDE for implementing the measurement program. The ESP-IDF is the official development framework for ESP32. MATLAB is using the data to generate graphs or charts and then they will be compared/analyzed.

Mbed TLS libraries will be used to provide easy access to cryptographic algorithms. Mbed TLS is an implementation of the TLS and SSL protocols and the respective cryptographic algorithms and support code required. It makes it easy for developers to include cryptographic and SSL/TSL capabilities to their embedded products with a minimal coding footprint. For this thesis project, the mbed TLS libraries will be used for AES, SHA-2, and RSA algorithms.

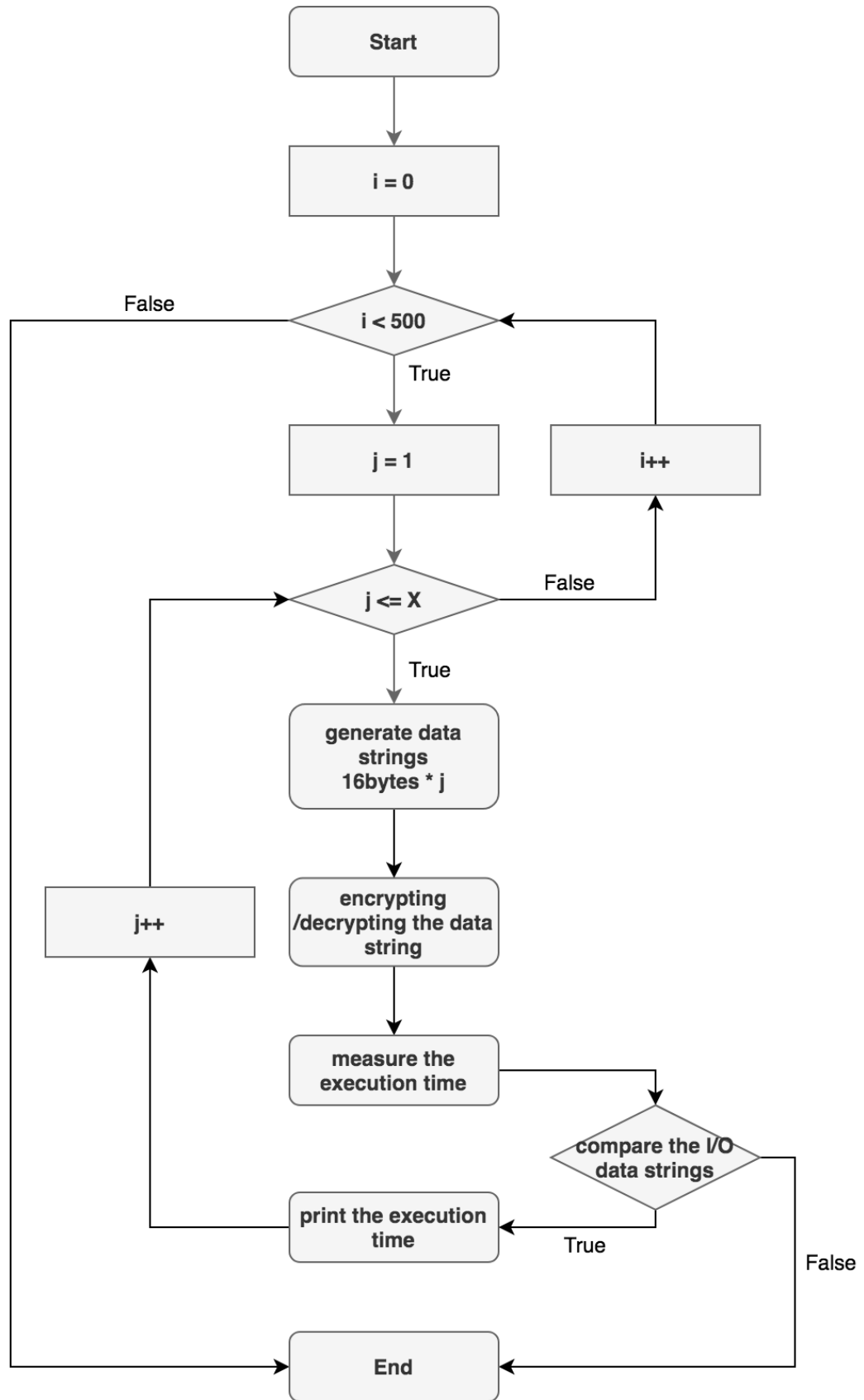


Figure 3.3: Flow chart on how the measurement program works. The X is the number of data points and its value varies between 100, 7 and 15 with the corresponding algorithm.

3.5 Assessing Reliability and Validity of the Data Collected

This section is about the reliability and validity of the data collected with the measurement program. Methods used to improve reliability and validity are also mentioned.

3.5.1 Reliability

Reliability is about whether or not the same result can be produced using the same instrument to measure something more than once. The test-retest reliability is about obtaining similar results of successive measurements of the same measure made under the same measurement conditions [41].

To increase the reliability of the data in this thesis project. The input data messages used are generated by the pseudo-random number generator (PRNG). The PRNG has a seed value; using the same seed value will generate the same set of random data messages, makes it possible to reproduce the same experiment inputs. Another way to increase the reliability is to increase the number of observations; a larger sample of data is more representative and reliable. When measuring the selected algorithms' execution, each data point will be measured 500 times to prove the consistency of the measured data. To standardizing the conditions under which the experiment is taken, all the algorithms are measured using similar methods under the same evaluation environment.

3.5.2 Validity

Validity is about the extent to which a concept, conclusion, or measurement is likely corresponding accurately in the real world. It is about how well an instrument measures what it is expected to measure [41].

The time measurement function *gettimeofday ()* is used to measure the execution time of all the algorithms in this thesis project. The *gettimeofday ()* function shall obtain the current time, expressed as seconds and microseconds [42]. To measure an algorithm's execution time, two *gettimeofday ()* function will be placed in the beginning and end of part of the algorithm that will be measured. The time difference of those two *gettimeofday ()* function will be the execution time of the measured algorithm. To check if the execution time measured using this function is valid, a small experiment will be conducted. The experiment includes several parts:

- Measure the execution time of an algorithm's encryption and decryption separately. Then measure the execution time of the same algorithm performing encryption and decryption together. The AES-128 algorithm in CBC-mode will be measured, data string size is 32 bytes.
- Running an empty *gettimeofday ()* function pair. In an ideal scenario, the empty *gettimeofday ()* function pair should provide a result of 0 seconds.
- Measure the execution time of single and double *gettimeofday ()* functions, respectively.
- Perform the same tests above, but with a higher CPU clock frequency (240 MHz). This is to check if the CPU clock frequency will affect the *gettimeofday ()* function.

Table 3-2 shows the different time measurements from the experiment. The result shows that the function *gettimeofday ()* has an overhead of around 23 microseconds with a CPU clock frequency of 160MHz. Figure 3.4 shows how the overhead is interpreted. Assuming there are two kinds of overheads, overhead 1 is the time it requires initialize the function, and overhead 2 is the time it needs to execute the function. For the empty function, the execution time is the sum of overhead 1 and overhead 2, which is 23 microseconds. For the execution time of one *gettimeofday ()* function, there will be one additional overhead 1 and overhead 2; thus the execution time is around

48 us. This conclusion proves to be true with the execution time measurement of the AES algorithm. The sum of the encryption time and decryption time will have two overheads (overhead 1 + overhead 2) since they are measured separately. However, they only have one overhead when they are measured together. The difference between them is 23 microseconds, which agree with the time measurements. To make the values of execution time measurement more valid, the execution time of all algorithms measured in this project should be corrected by the corresponding overhead or consider this when analyzing the data.

Table 3-2: Time difference between two *gettimeofday ()* functions

CPU clock frequency	Empty	One <i>gettimeofday ()</i>	Two <i>gettimeofday ()</i>	Encryption and decryption separately	Encryption and decryption together
160 MHz	23 us	48 us	74 us	72 us	50 us
240 MHz	15 us	32 us	52 us	50 us	34 us

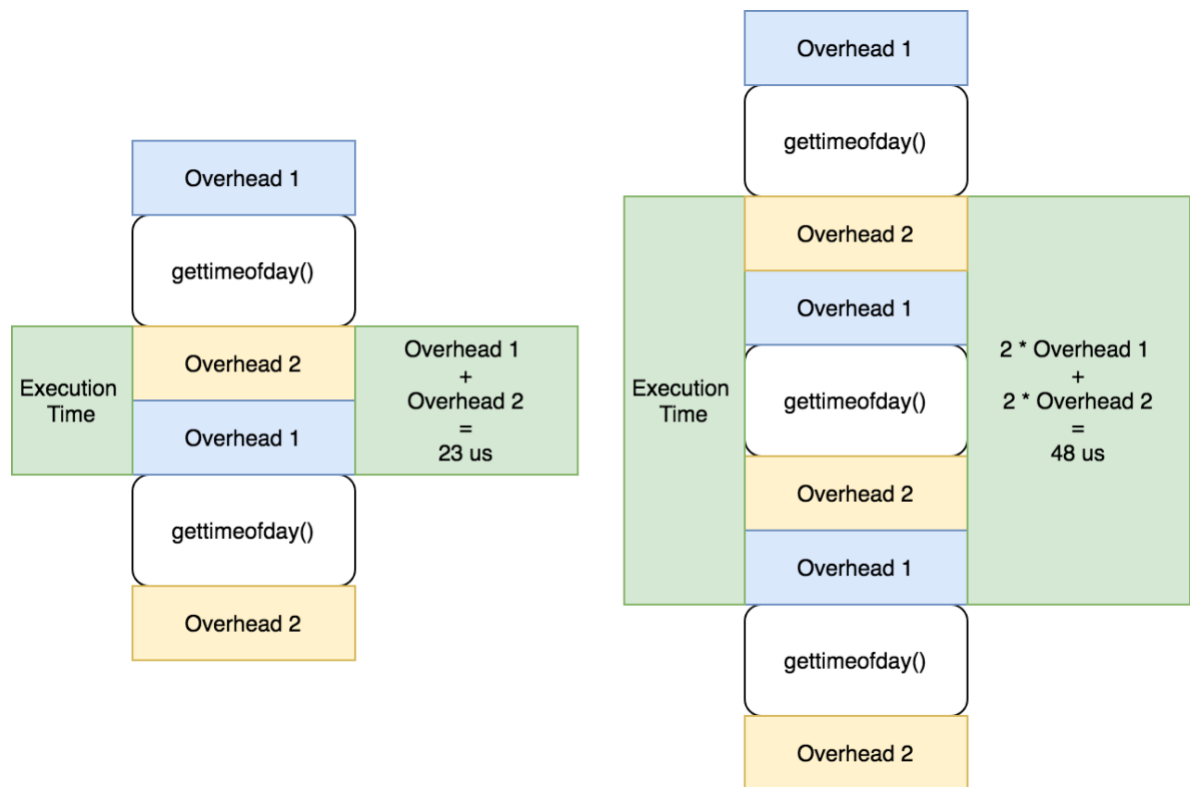


Figure 3.4: Overhead of *gettimeofday ()* function.

3.6 Planned Data Analysis

This section is about the data analysis technique used in this thesis project to analyze the primary data collected utilizing the data collection methods described in section 3.3.

3.6.1 Data Analysis Technique

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to interpret and draw conclusions from the collected data. Based on the kind of research, data analysis can either be quantitative, or qualitative or mixed. For this thesis, quantitative data analysis will be performed. Quantitative data analysis involves critical analysis, interpretation of figures, and numbers and tries to find the logic behind the main findings. There are two commonly used analysis methods for quantitative data analysis, statistics, and computational mathematics [7]. For this thesis, the statistics method will be used to evaluate the significance of the collected data. There are two categories of this method – descriptive statistics and inferential statistics [43].

In descriptive statistics, there are three types of measures of central tendency to comparing the distribution of scores. They are the mean, the median, and the mode. The mode is most used for qualitative data, the kind of nominal or categorical. So, it will not be used in this thesis project. Mean and median measures are useful and will be applied to describe the collected data.

Additional ways to describe a set of data are measures of variability. Measures of variability describe the amount of variability in a set of data. There are two types of measures of variability, the range and the standard deviation. The range is the difference between the highest and lowest value in a set of data. The standard deviation is the average amount that each of the individual data points varies from the mean of the data set. If all the data points in the data set are identical, the standard deviation is 0 [43].

For the software part, MATLAB is used to generate graphs of the collected data. The mean values and the standard deviations of the raw data will be calculated in MATLAB as well. The range of each data points' max and min values will be sorted out, then the graph of each algorithm's performance is compared and analyzed.

3.7 Summary

The research process and research paradigm are discussed at the beginning of the chapter. The research process is shown in Figure 3.1. The thesis's research paradigm is chosen to be a positivist paradigm because it is suitable for this project since this project is of experimental and testing character and generates a lot of numerical data.

Data collection techniques and how the sample size, data size, and data range are decided in this chapter. There is a summary of data size and other parameters in Table 3-1. For data collection techniques, it will be a primary data collection with a quantitative data collection method.

A flowchart of how the experiment design of the measurement program is presented and each step of the flowchart is explained. Figure 3.3 presents the flowchart. The hardware chosen for this project is ESP32 WROVER-KIT v.4.4. For the software part, the ESP-IDF SDK v.3.3 is chosen as the development and implementation tool for the ESP32 WROVER-KIT.

To increase the reliability of the data collected, a pseudo-random number generator is implemented. Also, a large sample of data makes it more reliable and more representative. The validity of the time measuring function *gettimeofday()* is discussed, and a solution is selected to increase the validity of the data collection.

Data analysis techniques are discussed, and for this thesis project, a quantitative analysis is most suitable. There are two common methods in the quantitative analysis; the statistic and computational mathematics. The statistic method with descriptive statistics is chosen for analyzing

the data collection. MATLAB is the software used for analyzing all collections of data. It is also used for making graphs and figures.

4 Encryption Performance Measuring Programs and Hardware Acceleration

This chapter presents how all the selected cryptographic algorithms are measured using the measurement program. Also, the hardware acceleration feature on EPS32 is mentioned. Section 4.1 describes the evaluation program for all the selected cryptographic algorithms. Section 4.2 is a brief mention of the hardware acceleration feature on the ESP32 WROVER-KIT. Section 4.3 summarizes the entire chapter.

4.1 The Evaluation Programs

For the evaluation of all the chosen algorithms, a suitable evaluation program for each algorithm is required. This program should work and have the same functionality as the flow chart in Figure 3.3 in chapter 3. The execution time will be measured using the *gettimeofday ()* function, and the result will be collected.

4.1.1 Pseudo-random Number Generator (PRNG)

The pseudo-random number generator will generate random data messages within number 0-9 and all lower cases of the alphabets. This PRNG can also be seeded with an integer value. The length of the generated data message can be varied depending on the needs. Algorithm 4–1 shows the code of the PRNG. The *static const char number []* contains the letters and numbers that can appear in a generated data string. The random number generator function *srand ()* is used to seed, and the seed value should be an integer. *[rand () % (sizeof(number)-1)]* will randomly generate a number from 0 to 35 (the size of the number, which contains all the characters – 1). The corresponding character in *number* will be selected and stored in *s []*. Depend on the length of the data message that needs to be generated; each character is pseudo-randomly generated and composed into a data message.

The quality of the pseudo-random number generator depends on the *rand ()* function. The *rand ()* function returns a pseudo-random number in the range of 0 to *RAND_MAX*, a constant whose default value may vary between implementations, but it is granted to be at least 32767. The *rand ()* function in C language makes no guarantees of the quality of the random numbers produced. The random numbers generated by the *rand ()* function have a comparatively short cycle, and the numbers can be predictable [44]. But since the purpose of this pseudo-random number generator is to generate random data messages for the cryptographic algorithms to encrypt. The quality of randomness is not essential. Also, because of the feature of *srand ()*, the program will generate the same value every time using the same seed number. This makes it much easier for future researchers to reproduce this project.

Algorithm 4–1: The pseudo-random number generator.

```
void gen_random(char *s, const int len){
    static const char number[] = "0123456789"
                                "abcdefghijklmnopqrstuvwxyz";
    srand(128);
    for (int i = 0; i < len; i++){
        s[i] = number [rand() % (sizeof(number)-1)];
    }
    s[len] = 0;
}
```

The input data message is generated using the PRNG with seed value 128. The evaluation program will increase the length of the data message with 16 bytes each iteration. The data message will be encrypted by the chosen algorithm and later decrypted using the same algorithm.

4.1.2 Time Measurement Function

The time measurement is built with the *gettimeofday ()* function. This function takes the current time expressed as seconds and microseconds. To measure the execution time of e.g. AES encryption, the *gettimeofday ()* function is placed before the encryption function and immediately after the encryption function. The time difference is the execution time of the encryption function. The unit of the *gettimeofday ()* function is in microseconds, and since the algorithm is quite fast, it is a suitable choice for the execution time units. All three cryptographic algorithms' execution time is measured under different CPU clock frequency, 160MHz, and 240MHz, respectively.

4.1.3 AES Algorithm

For the AES algorithms, the AES 128 CBC mode was evaluated. The CBC mode is from the mbed TLS library. The choice of the secret key is also important for the future replication of this project. In this case, the secret key is randomly generated by the PRNG with a seed value of 128. Usually, the secret key should be generated by a good random source because the strength of the key depends on the unpredictability of the random seed. The secret key needs to have a key length of 16 bytes (128bits) since it is used for AES 128.

The CBC mode can take the input data with a size of 16 bytes or more per iteration. However, if the input is not a multiple of 16 bytes, it needs to be filled with zeros to make the input length a multiple of 16 bytes. This is one of the reasons why the size of input data messages is chosen to be 16 bytes or a multiple of 16 bytes.

Figure 4.1 shows the high-level process of how the data message is encrypted and later decrypted using AES 128 with CBC mode. It also shows how the execution time of encrypt the data message and decrypt it back to the original message is measured using *gettimeofday ()* function. The AES encryption or decryption can be measured separately as well; just move the *gettimeofday ()* function to the desired place.

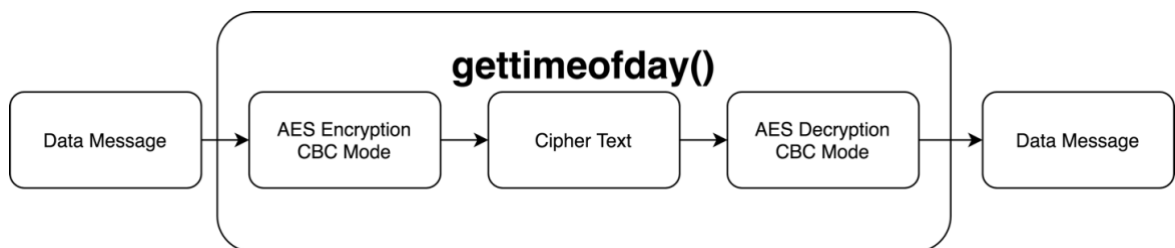


Figure 4.1: The process how the data message is encrypted and decrypted using the AES algorithm.

4.1.4 RSA Algorithm

RSA algorithms have different key lengths. In this project, two different key lengths were evaluated: 1024 bits and 2048 bits. RSA 2048 provides higher levels of security than RSA 1024 but requires more computational power and a long time since more calculations need to be executed. Key pairs with the key size 1024 and 2048 are generated using OpenSSL [45].

RSA has a maximum encryption data size. It can only encrypt data to a maximum amount equal to the key size. In this case, RSA with a key size of 1024 bits can encrypt 128 bytes of data minus any padding and header data. This shows that RSA is not the best encryption algorithm to encrypt large data directly. Since RSA can only encrypt data as large as the key size, the number of iterations is mainly affected by this.

When doing RSA encryption and decryption the program will crash when the iterations are too high. This problem can be solved by increasing the main task stack size. The main task stack size has to be raised from 3584 bytes to 7168 bytes to run and execute the RSA algorithm properly on the ESP32. Otherwise, the execution will be terminated midway because of the lack of memory.

Similar to the AES algorithm, the process of how the data message is encrypted and decrypted using RSA 1024/ 2048 is shown in Figure 4.2. Because the keys for the RSA algorithm are a bit more complicated compares to the AES secret key. These are pre-generated and store in the SD card on the ESP32.

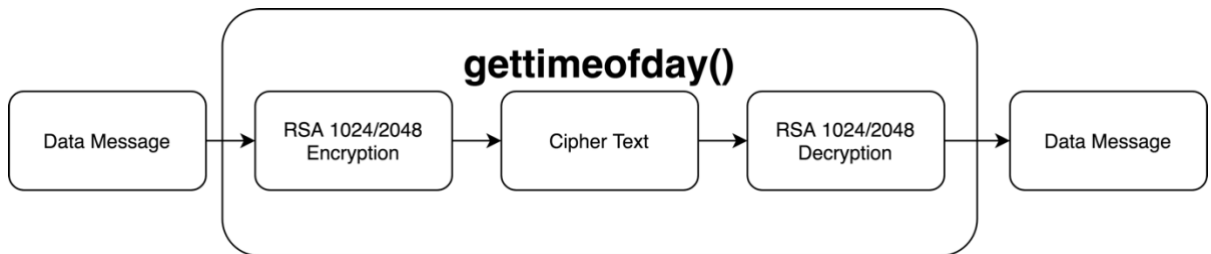


Figure 4.2: The process of how the data message is encrypted and decrypted using RSA 1024/ 2048.

4.1.5 SHA-2 Algorithm

The SHA-256 is chosen from the SHA-2 family. For the hashing algorithm, the same data messages used in AES were used here since it is quite convenient. SHA-256 is much quicker compared to the other algorithms evaluated in this project. It is efficient because SHA algorithms are built for encrypting large data files.

Since the SHA-256 is efficient at encrypting large data files, the iterations can be many as well.

Compare SHA with those other algorithms, the process of the SHA algorithm is more straightforward. The data message will be hashed with SHA-256, and a hash digest/ hash value will be produced. Figure 4.3 shows the process and how the execution time is measured. Because there is no encryption or decryption, the only time that needs to be measured here is the hash function's execution time.

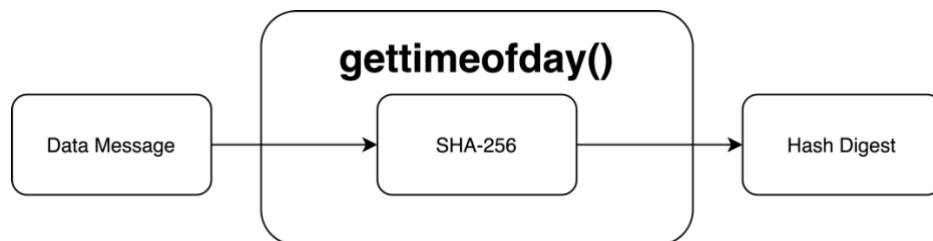
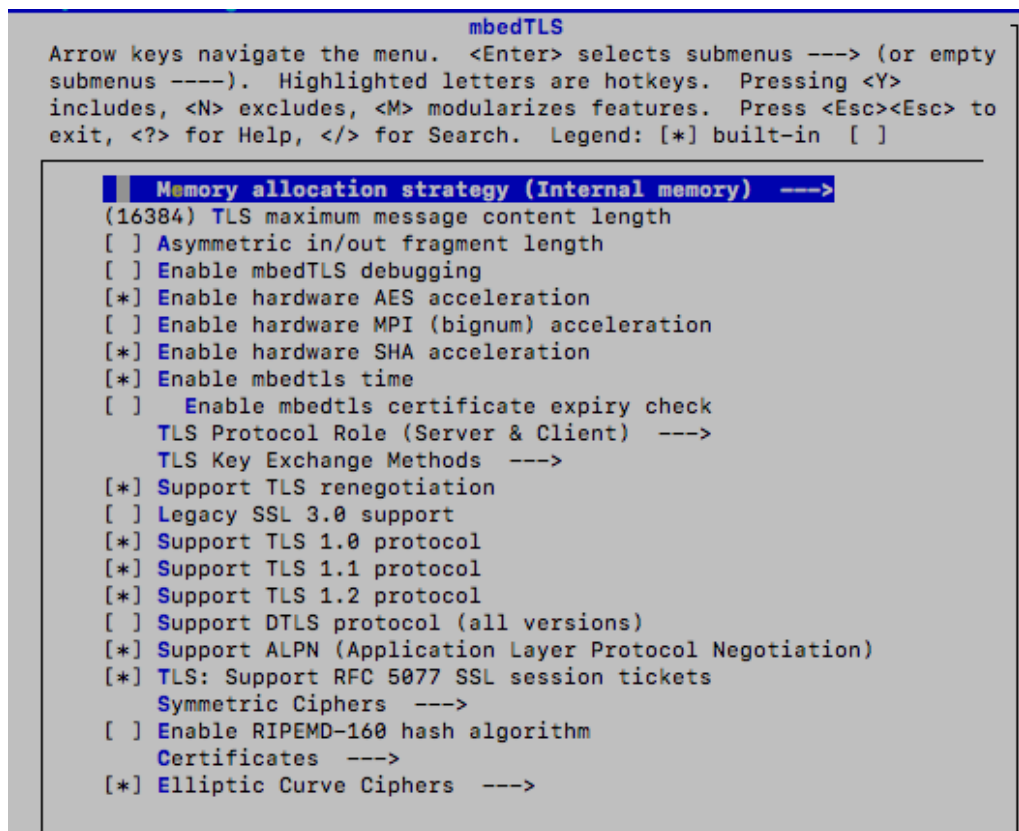


Figure 4.3: The process of how the data message is hashed using SHA-256.

4.2 Hardware Acceleration

The ESP32 is equipped with hardware accelerators of general algorithms, such as AES, RSA, and SHA. The hardware acceleration feature can be toggled on/off in the Espressif IoT development framework configuration.

The IoT development framework configuration utility from Espressif is used to toggle the hardware acceleration for cryptographic algorithms. Navigate to *Component config* --> *mbedtls*, then select "Enable hardware AES acceleration" if the AES algorithm needs to be accelerated. Select "Enable hardware SHA acceleration" for SHA. For the RSA algorithms, Select "Enable hardware MPI (bignum) acceleration". The hardware acceleration can be turned on for all algorithms simultaneously but keeping it only for the algorithm that is going to measure is recommended. Since the hardware acceleration feature does consume CPU power, it will make the general performance of the cryptographic algorithm slower. Figure 4.4 shows that hardware acceleration support AES, MPI (bignum), and SHA.



```

mbedtls
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

Memory allocation strategy (Internal memory) --->
(16384) TLS maximum message content length
[ ] Asymmetric in/out fragment length
[ ] Enable mbedtls debugging
[*] Enable hardware AES acceleration
[ ] Enable hardware MPI (bignum) acceleration
[*] Enable hardware SHA acceleration
[*] Enable mbedtls time
[ ] Enable mbedtls certificate expiry check
  TLS Protocol Role (Server & Client) --->
  TLS Key Exchange Methods --->
[*] Support TLS renegotiation
[ ] Legacy SSL 3.0 support
[*] Support TLS 1.0 protocol
[*] Support TLS 1.1 protocol
[*] Support TLS 1.2 protocol
[ ] Support DTLS protocol (all versions)
[*] Support ALPN (Application Layer Protocol Negotiation)
[*] TLS: Support RFC 5077 SSL session tickets
  Symmetric Ciphers --->
[ ] Enable RIPEMD-160 hash algorithm
  Certificates --->
[*] Elliptic Curve Ciphers --->

```

Figure 4.4: Espressif IoT development framework configuration.

4.3 Summary

In this chapter, the following subjects were described, the pseudo-random number generator, the time measure function, the chosen cryptographic algorithms, and how the ESP32 hardware acceleration should be configured for this thesis project.

The pseudo-random number generator is implemented for generated pseudo-random data messages according to the requirements. The data message generated contains lower letters and

numbers, the length of the data message must be 16 bytes or a multiple of 16 bytes. The data messages are later used as input for all the cryptographic algorithms.

All three types of algorithms and their configuration are described. The process of how each algorithm is executed and how the execution time is measured is presented with figures. Table 4-1 shows a summary of the configuration of all the chosen cryptographic algorithms.

The ESP32's hardware acceleration feature is also introduced, and how to use the Espressif IoT development framework configuration to set and enable the hardware acceleration for the specified algorithm is explained.

Table 4-1: Summary of the configuration for all evaluated cryptographic algorithms.

	Mode/variation	Key size (bits)	Max input data size (bytes)	Main task stack size (bits)	Key generation method
AES	CBC	128	1600	3584*2	PRNG
RSA	-	1024, 2048	112, 240	3584*2	OpenSSL
SHA	SHA-256	Keyless	1600	3584*2	-

5 Results and Evaluation

In this chapter, the results are presented and discussed. The major results are presented in section 5.1. It follows with a discussion in section 5.2. A summary of the results is presented in section 5.3.

5.1 Major Results

This section provides the experimental results on measured cryptographic algorithms. It has been divided into three parts, AES 128, SHA-256, and RSA 1024/2048.

5.1.1 General experimental description

All three cryptographic algorithms are performed under the same configuration. There are four different configurations: hardware acceleration on/off with 160MHz CPU clock frequency and hardware acceleration on/off with 240MHz CPU clock frequency. The execution time will be measured for all three cryptographic algorithms in all four different configurations.

5.1.2 AES algorithm

Figure 5.1 shows the execution time of AES 128-bits with CBC mode encrypting and decrypting data messages with different sizes from 16 to 1600 bytes and different CPU clock frequency. The peak at the beginning should be ignored because it only happens in the first iteration and should not be taken into consideration. There are two different lines in those graphs, max: the highest recorded value for that data point and min: the lowest recorded value of that data point. It shows that the max and min are very close to each other and it is difficult to distinguish them from the graphs. The curves grow linearly corresponding to the increased data size. The hardware acceleration feature does make the overall performance of AES 128-bits faster especially when the size of the data becomes larger.

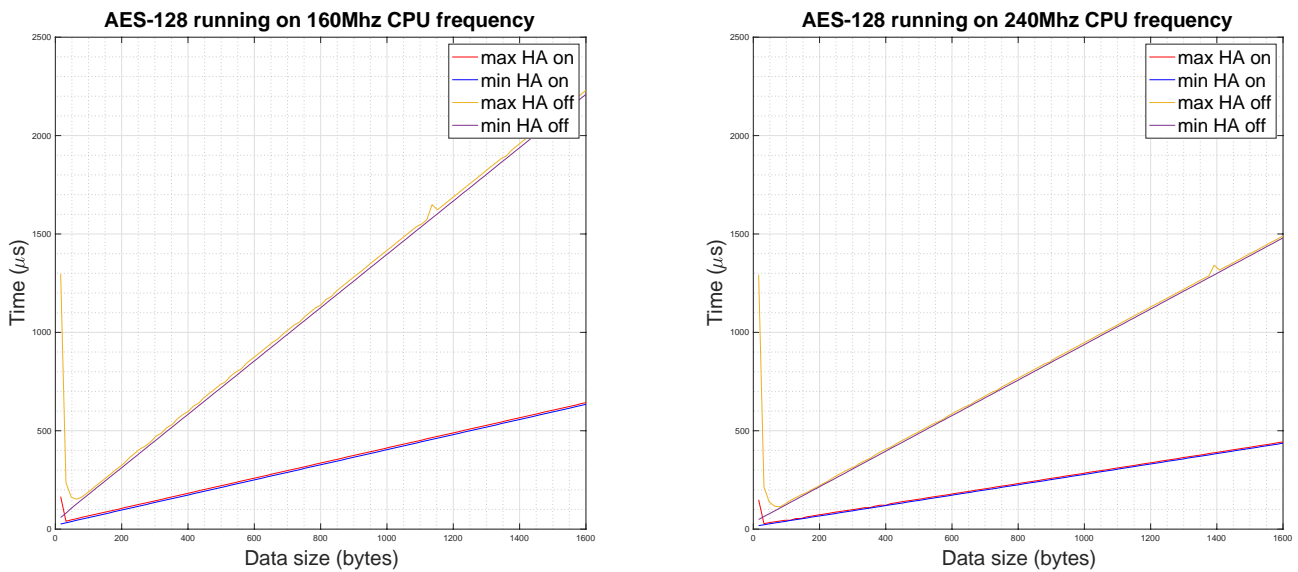


Figure 5.1: Performance of AES 128-bits CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

The execution time of the encryption and decryption of AES 128 bits in CBC mode are measured separately, and the results are shown in Figure 5.2 and Figure 5.3, respectively. The curves for both encryption and decryption performance are very similar. With higher CPU clock frequency, the execution times for both scenarios are faster. For encryption, the execution time for encryption dropped about 25%, and similarly, the execution time for decryption also dropped as much.

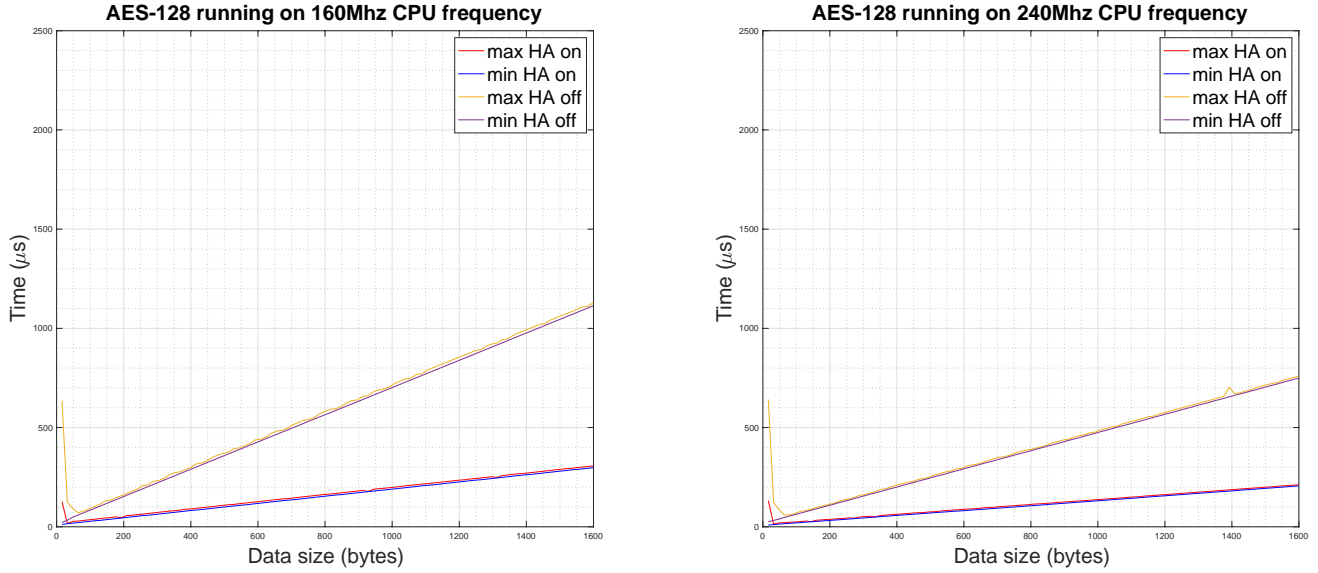


Figure 5.2: Encryption of AES 128-bits in CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

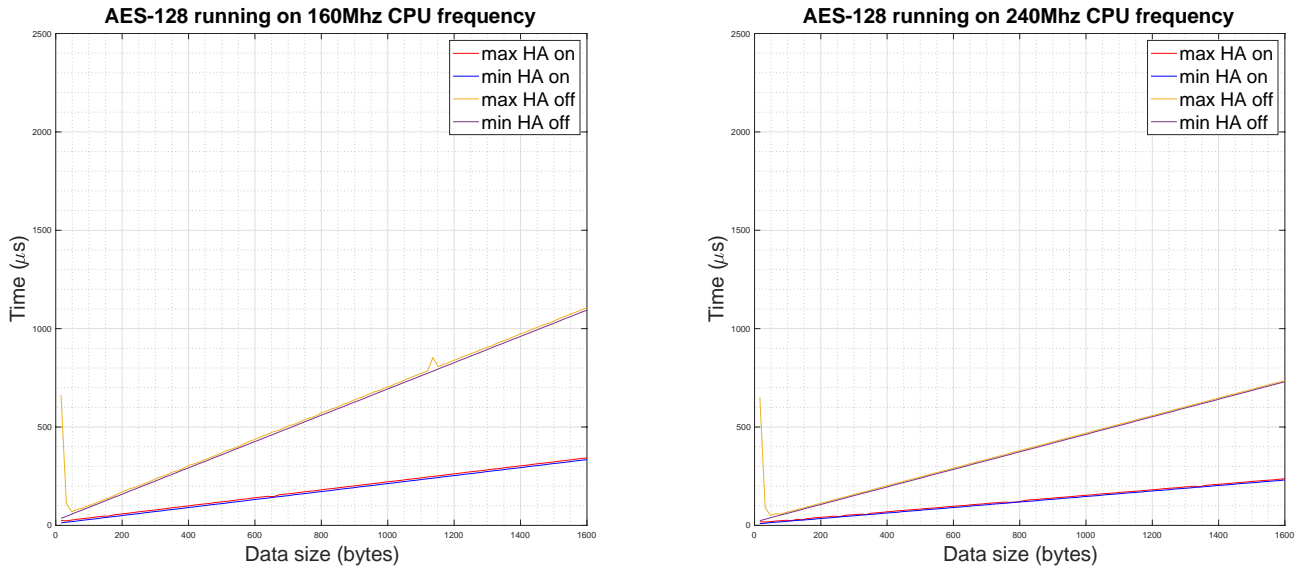


Figure 5.3: Decryption of AES 128-bits in CBC mode with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Table 5-1 shows the throughput (byte per microsecond) values for AES 128-bits with CBC mode running on 160MHz. As expected, the performance of both encryption and decryption are faster due to the hardware acceleration for the AES algorithm. The same measurement process is done with a CPU clock frequency of 240MHz as well, and the results are shown in Table 5-2.

Table 5-1: Performance of AES 128-bits CBC mode with CPU clock frequency 160MHz

AES 128 CBC mode (byte/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	4.9147	4.4122	2.3168
Hardware acceleration off	1.3736	1.3672	0.6841
Throughput increase	257.8%	222.7%	238.6%

Table 5-2: Performance of AES 128-bits CBC mode with CPU clock frequency 240MHz

AES 128 CBC mode (byte/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	7.0794	6.3617	3.3571
Hardware acceleration off	1.9929	2.0452	1.0099
Throughput increase	255.2%	211.1%	232.4%

Figure 5.4 shows the bar diagram of the throughput of AES 128-bits encryption and decryption running on different CPU clock frequencies. As shown in Figure 5.4, the hardware acceleration increased AES-128's throughput significantly, the hardware acceleration made the encryption 257.8% faster and the decryption 222.7% faster with 160MHz CPU clock frequency. For 240MHz, the throughput of the encryption is increased by 255.2%, and decryption is increased by 211.1% using the hardware acceleration feature.

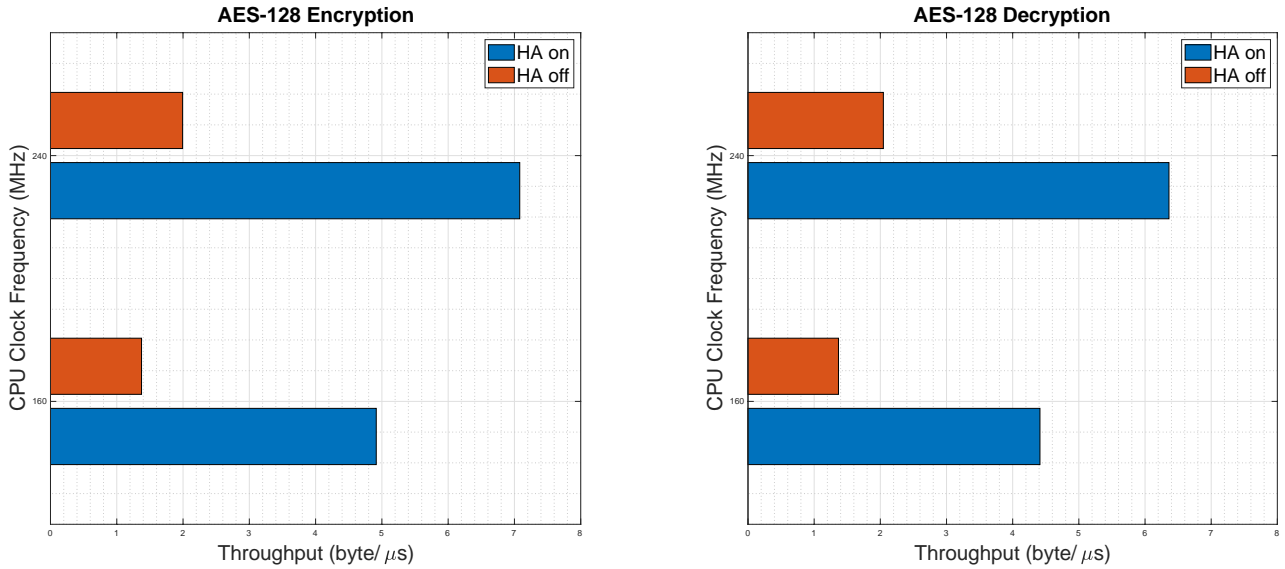


Figure 5.4: The throughput of the AES-128, the left diagram shows the throughput of AES-128 encryption, the right diagram shows the throughput of AES-128 decryption.

5.1.3 SHA algorithm

SHA-2's performance graph is shown in Figure 5.5. The legend shows what value each color corresponds to. The stepping effect of the curves is likely due to how SHA processes the hash. A step will form for each additional 64 bytes of data. In [46], the author does a similar experiment where the execution time of SHA-256 and SHA-512 are measured, and the scatter plots obtained after the measurement shows the same stepping effect. Once again, the hardware acceleration from ESP32 made SHA-2's performance much faster. The larger the data needs to be hashed, the more it can reflect the effect of hardware acceleration. It can be seen from Figure 5.5 that when hashing data with the size of 200 bytes, enabling hardware acceleration reduces the execution time by about 44%. If the data size is 1200 bytes, the execution time reduction is far more than 44%, and the reduction is almost 67% of the original execution time. With higher CPU clock frequency, the graph has similar curvature as the 160MHz one, but the overall performance is faster for all configurations.

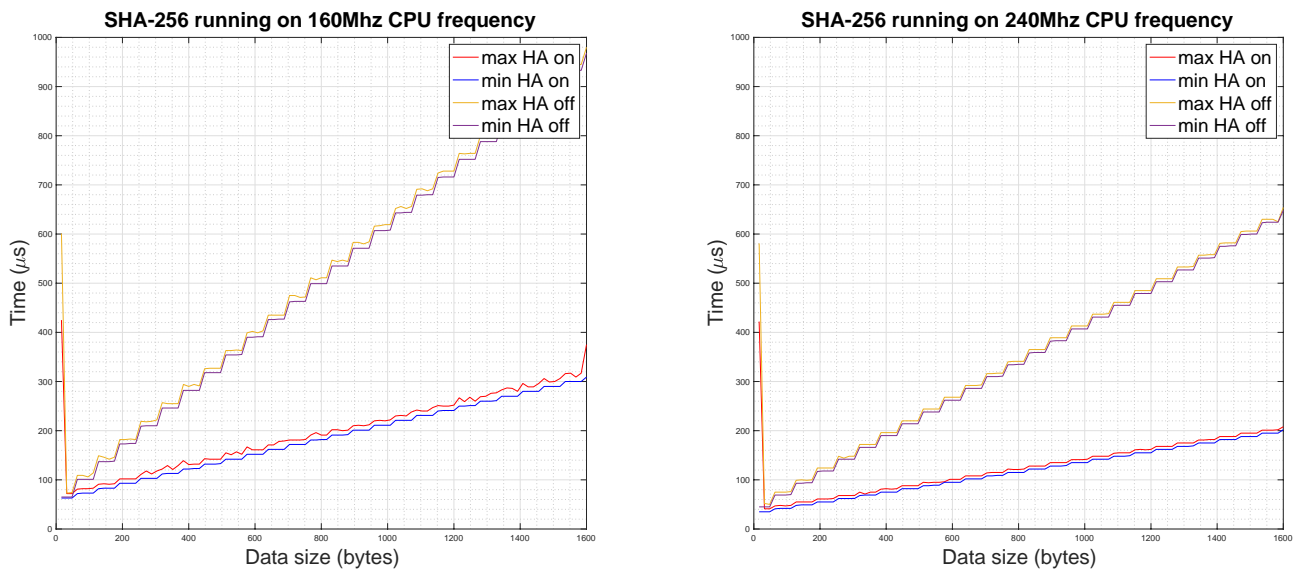


Figure 5.5: Performance of SHA-2 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Table 5-3 shows the performance of SHA-256 in byte per microsecond. The performance is nearly three times faster with hardware acceleration compares to without. With 240MHz CPU frequency, the SHA-256 algorithm's performance is even faster compared with 160MHz, and the throughput values are shown in Table 5-4.

Table 5-3: Performance of SHA-2 with CPU clock frequency 160MHz

SHA-256 (byte/us)	Hashing
Hardware Acceleration on	3.9300
Hardware Acceleration off	1.4818
Throughput increase	165.2%

Table 5-4: Performance of SHA-2 with CPU clock frequency 240MHz

SHA-256 (byte/us)	Hashing
Hardware Acceleration on	6.2060
Hardware Acceleration off	2.2060
Throughput increase	181.3%

The throughput of the SHA-256 algorithm is shown in Figure 5.6. For 160MHz, the hardware acceleration does a 165.2% increase of the hashing speed. For 240MHz, the increase of the hashing speed is 181.3%.

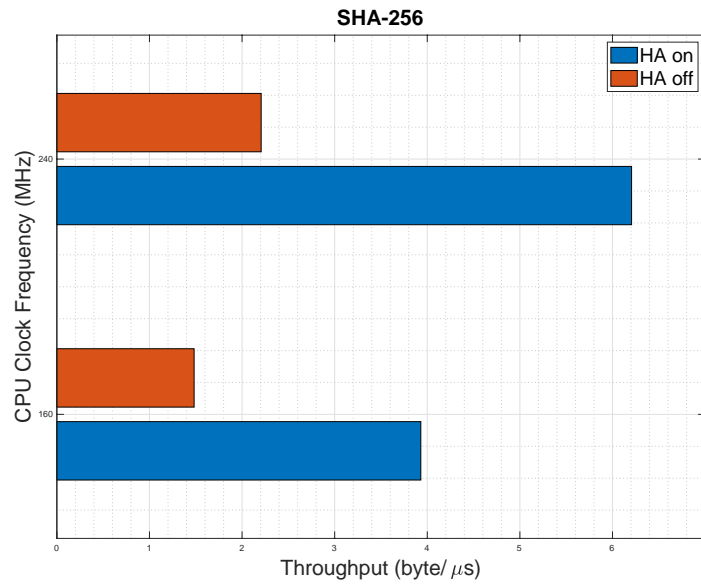


Figure 5.6: The throughput of SHA-256 performing under different CPU frequencies.

5.1.4 RSA Algorithm

Different key length is measured for the RSA algorithm, and the result is presented by graphs and tables. Figure 5.7 shows the RSA with key length 1024 bits with hardware acceleration turned on/off running under different CPU clock frequencies. The graphs show the execution time of RSA 1024 when encrypting and later on decrypting a message of pseudo-random generated text with different sizes. Just like with the AES graphs, the peak at the beginning should be ignored. The size of the data merely affected the execution time of the RSA 1024. Without hardware acceleration, the execution time stayed at 2.25×10^5 microseconds and 1.5×10^5 microseconds for 160MHz and 240MHz, respectively. When the hardware acceleration is turned on, the execution time is much faster. When running under 160MHz, the execution time dropped down to just under the 1×10^5 microsecond line. For 240MHz, the time is even shorter at 0.65×10^5 microseconds.

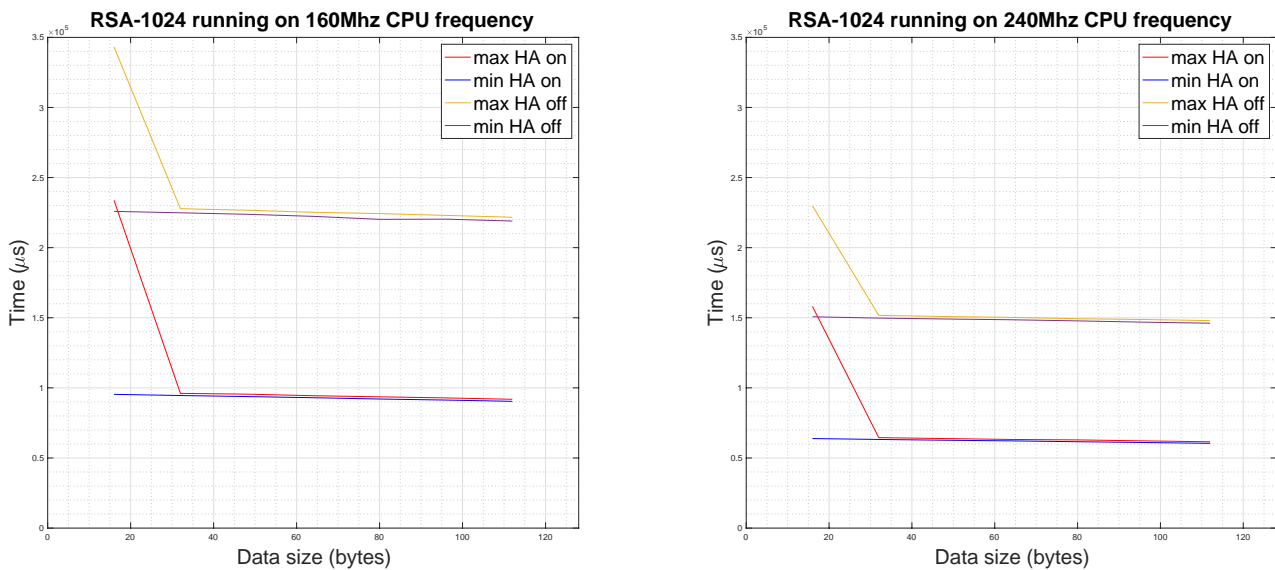


Figure 5.7: Performance of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Figure 5.8 shows the encryption time, and Figure 5.9 shows the decryption time of RSA 1024. Here is where the interesting thing happens. The graphs have shown that the execution time of RSA 1024 when encrypting data message is slower when the hardware acceleration is turned on. Also, it shows with increased data size, the time needed for encrypting the data is less. However, for the decryption, everything is normal. The hardware acceleration does make the decryption speed faster. It should be noted here that the scale between the execution time of encryption and the execution time of decryption in Figure 5.8 and Figure 5.9 is 1:10. The choice of the scale is to better show the difference between the encryption with and without hardware acceleration. It can be seen that the execution time required for encryption is much shorter than that for decryption.

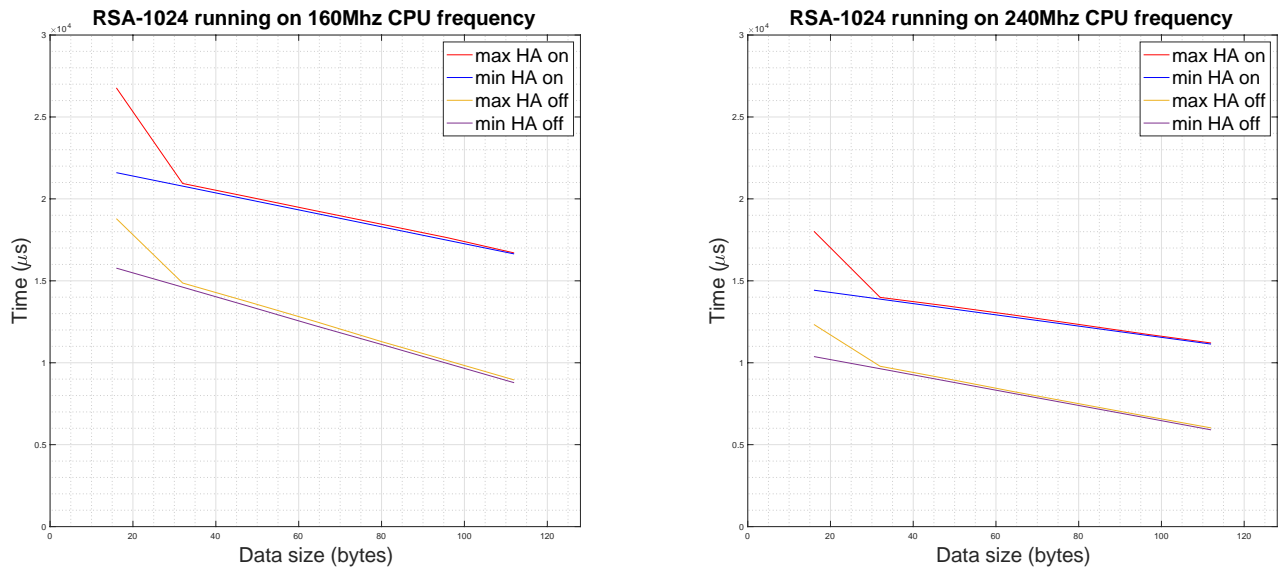


Figure 5.8: Encryption of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

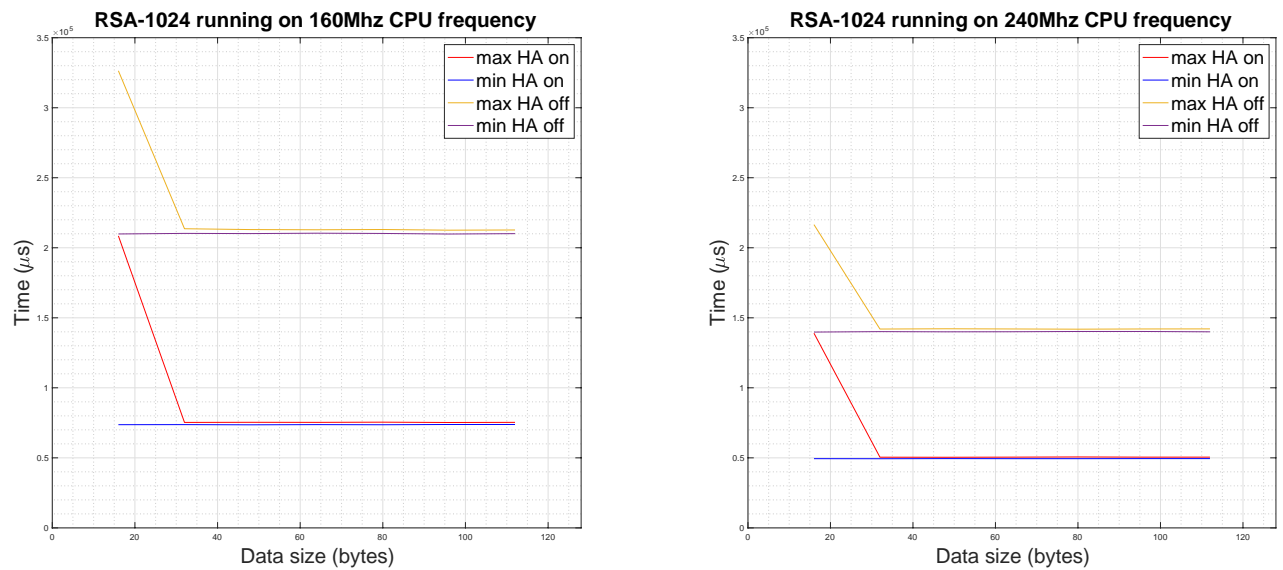


Figure 5.9: Decryption of RSA 1024 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Table 5-5 shows the throughput for the key length of 1024 bits and the difference between using the hardware acceleration and not using the hardware acceleration. The same measurement is done for 240MHz CPU frequency, and the performance for RSA 1024 is shown in Table 5-6.

Table 5-5: Performance of RSA 1024 with CPU clock frequency 160MHz

RSA 1024 (bytes/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	0.0035	8.5925e-04	6.8964e-04
Hardware acceleration off	0.0059	3.0254e-04	2.8720e-04
Throughput increase	-40.7%	184%	140.1%

Table 5-6: Performance of RSA 1024 with CPU clock frequency 240MHz

RSA 1024 (bytes/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	0.0053	0.0013	0.0010
Hardware acceleration off	0.0089	4.5327e-04	4.3047e-04
Throughput increase	-40.4%	186.8%	132.3%

In Figure 5.10, the throughput for RSA 1024 performing on different CPU frequencies is shown. The result is very interesting because the encryption speed is much higher when the hardware acceleration feature is turned off. For 160MHz, the encryption speed has a decrease of 40.7%, and for decryption, the speed has an increase of 184.0%. When running under 240MHz CPU clock frequency, the encryption speed is 40.4% slower when using the hardware acceleration, and the decryption is 186.8% faster.

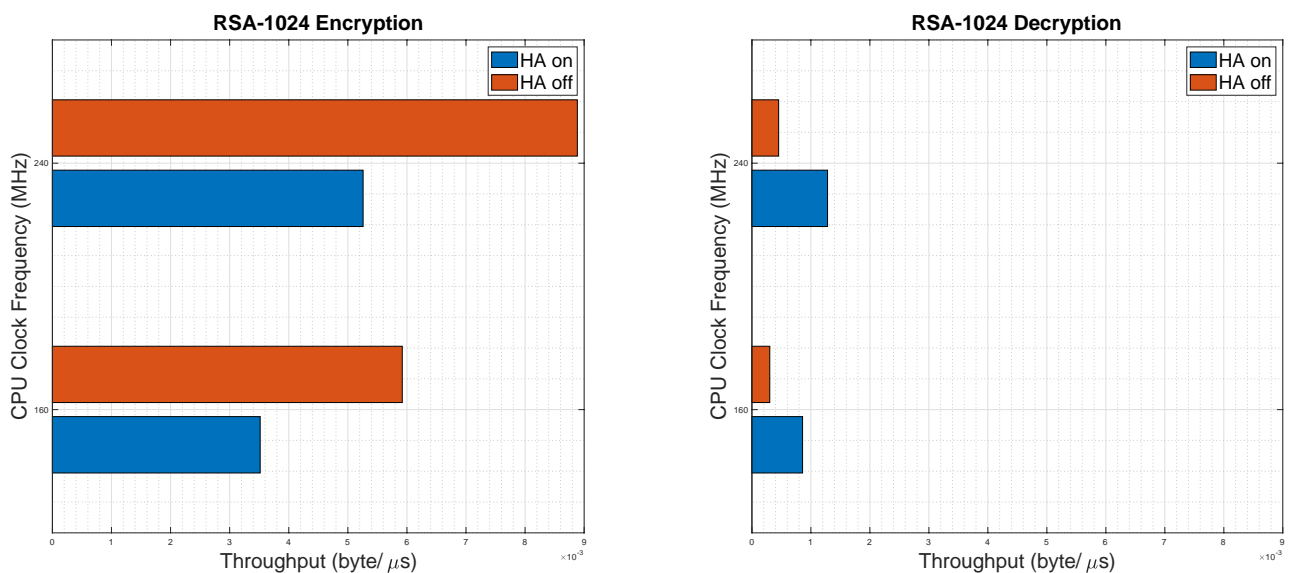


Figure 5.10: The throughput of RSA-1024 performing under different CPU clock frequencies, the left diagram shows the throughput of the encryption, the right diagram shows the throughput the decryption.

Figure 5.11 is presenting the performance of RSA with a key length of 2048 bits. Similar to RSA 1024, the execution time is not affected by the size of the data. The only difference with RSA 2048 besides the longer key size is the execution time is much longer. Compared with RSA 1024, the execution time is nearly six times slower for all configurations.

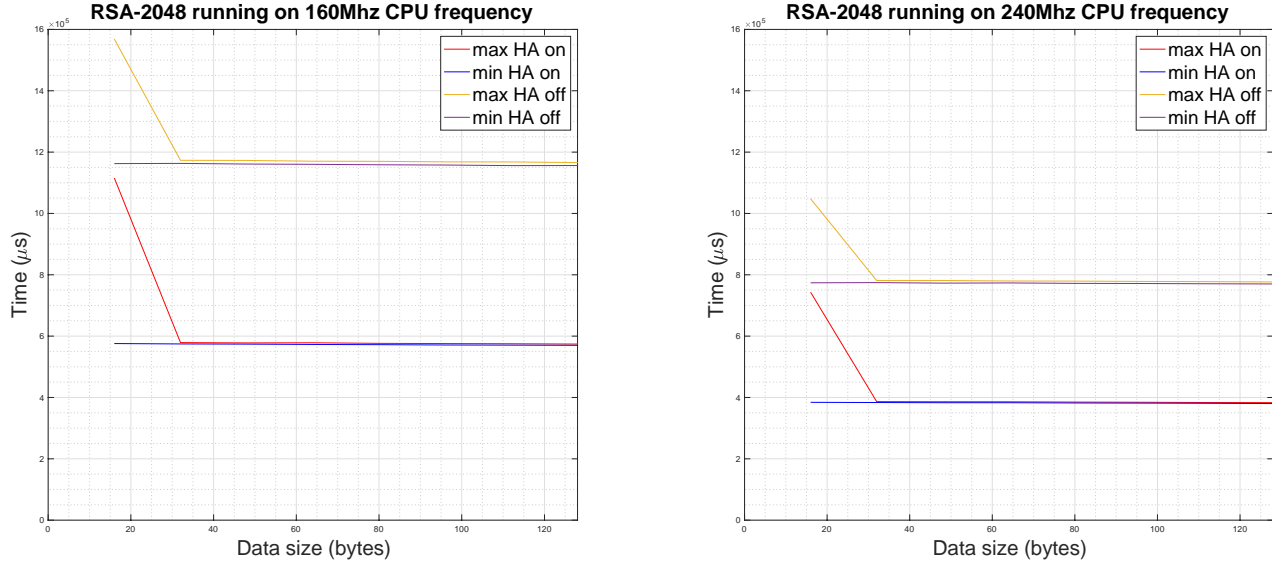


Figure 5.11: Performance of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Figure 5.12 and Figure 5.13 shows the encryption time and decryption time of RSA 2048 separately. As shown in the figures, the RSA algorithms are much slower compares to the previously measured algorithms. Just like what happened with RSA 1024, the same observation can be made here as well. The hardware acceleration made the RSA encryption much slower. Compared to not using hardware acceleration, the execution time is about three times slower for 160MHz CPU clock frequency and about two times slower for 240MHz. The decryption is not affected in any way, just like the decryption of RSA 1024.

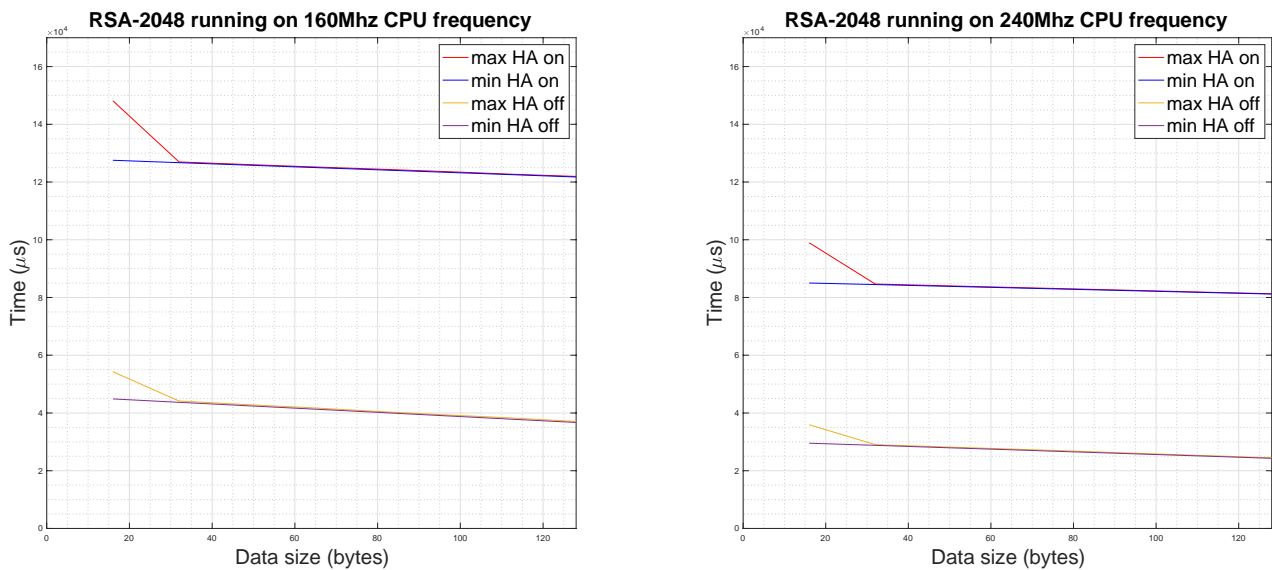


Figure 5.12: Encryption of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

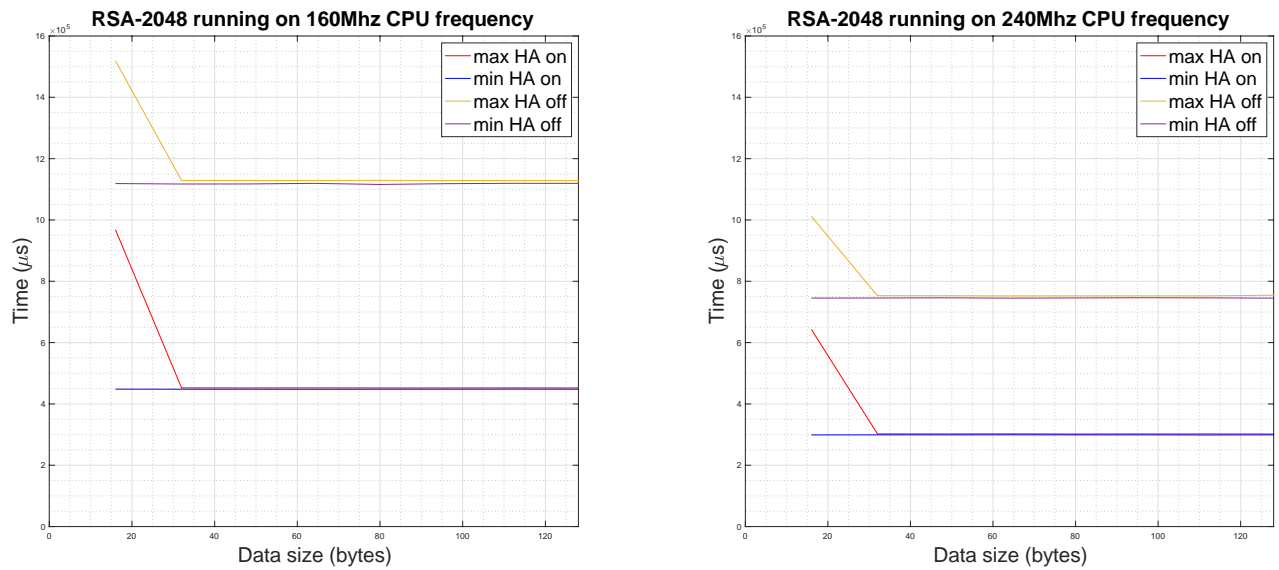


Figure 5.13: Decryption of RSA 2048 with/without hardware acceleration with CPU frequency 160MHz and 240MHz.

Table 5-7 shows the throughput for the key length of 2048 bits and the difference between using the hardware acceleration and not using the hardware acceleration. The same measurement is done for 240MHz CPU frequency, and the performance for RSA 2048 is shown in Table 5-8.

Table 5-7: Performance of RSA 2048 with CPU clock frequency 160MHz

RSA 2048 (bytes/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	0.0011	2.8433e-04	2.2457e-04
Hardware acceleration off	0.0038	1.1386e-04	1.1051e-04
Throughput increase	-71.1%	149.7%	103.2%

Table 5-8: Performance of RSA 2048 with CPU clock frequency 240MHz

RSA 2048 (bytes/us)	Encryption only	Decryption only	Encryption & Decryption
Hardware Acceleration on	0.0016	4.2586e-04	3.3643e-04
Hardware acceleration off	0.0057	1.7079e-04	1.6579e-04
Throughput increase	-71.9%	149.3%	102.9%

The throughput of performing on different CPU frequencies was done with RSA 2048, and the bar diagram is shown in Figure 5.14. Performing under 160MHz, the increase of the throughputs is -71.1% and 149.7% for encryption and decryption, respectively. For 240MHz, the encryption throughput is -71.9%, and the decryption is 149.3% faster when the hardware acceleration is used.

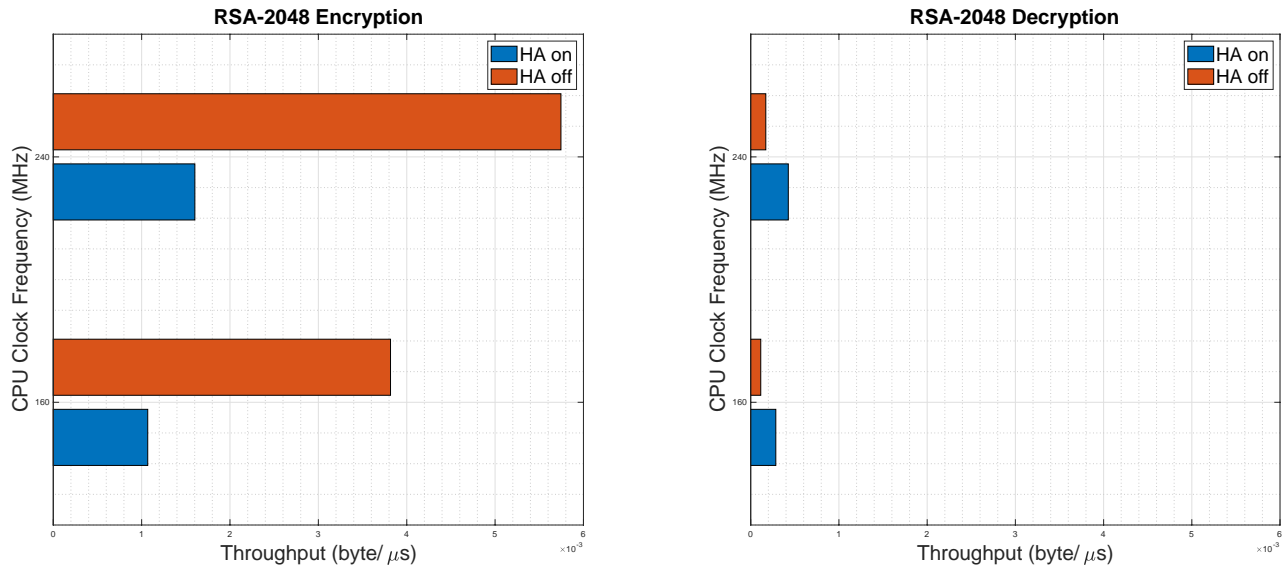


Figure 5.14: The throughput of RSA-2048 performing under different CPU clock frequencies, the left diagram shows the throughput of the encryption, the right diagram shows the throughput the decryption.

5.2 The Multi-Precision Integer Function

The hardware acceleration supported by ESP32 includes AES, SHA, RSA, ECC, and RNG. However, when entering the Developer Framework Configuration, only the hardware acceleration options for AES and SHA are available, and the third option is labeled as "Enable hardware MPI (bignum) acceleration". This MPI hardware acceleration is responsible for accelerating algorithms like RSA and ECC that need to compute large numbers. But this hardware acceleration only accelerates the MPI part of the RSA algorithm. The implementation of the RSA algorithm in the mbedtls library utilizes the following function shown in Algorithm 5–1 to perform the calculation " $a^e \bmod n$ ". There are two different libraries for calculating " $a^e \bmod n$ ", one is with hardware acceleration and the other is without, the following code snippet is the hardware accelerated version.

Algorithm 5–1: The hardware accelerated version of *mbedtls_mpi_exp_mod()*.

```
int mbedtls_mpi_exp_mod( mbedtls_mpi* Z, const mbedtls_mpi* X, const mbedtls_mpi*
Y, const mbedtls_mpi* M, mbedtls_mpi* _Rinv )
{
    int ret = 0;
    size_t x_words = word_length(X);
    ...

    size_t hw_words = hardware_words(MAX(m_words, MAX(x_words, y_words)));

    mbedtls_mpi Rinv_new; /* used if _Rinv == NULL */
    mbedtls_mpi *Rinv;    /* points to _Rinv (if not NULL) otherwise &Rinv_new */
    mbedtls_mpi_uint Mprime;

    ...
    ...
    ...

    Mprime = modular_inverse(M);

    esp_mpi_acquire_hardware();

    /* "mode" register loaded with number of 512-bit blocks, minus 1 */
    DPORT_REG_WRITE(RSA_MODEXP_MODE_REG, (hw_words / 16) - 1);

    /* Load M, X, Rinv, M-prime (M-prime is mod 2^32) */
    mpi_to_mem_block(RSA_MEM_X_BLOCK_BASE, X, hw_words);
    ...
    ...

    start_op(RSA_START_MODEXP_REG);

    /* X ^ Y may actually be shorter than M, but unlikely when used for crypto */
    if ((ret = mbedtls_mpi_grow(Z, m_words)) != 0) {
        esp_mpi_release_hardware();
        goto cleanup;
    }

    wait_op_complete(RSA_START_MODEXP_REG);

    mem_block_to_mpi(Z, RSA_MEM_Z_BLOCK_BASE, m_words);
    esp_mpi_release_hardware();

    ...
}
```

“ $a^e \bmod n$ ” is the main function used in RSA for encryption or decryption, where a and e will be the main variables. Here a can be interpreted as the input (a message to be encrypted or the ciphered text), while e is the randomly selected prime exponential. There is a more detailed explanation of this function and the theory of the RSA algorithm in Chapter 2, subsection 2.2.1.

The code snippet was extracted and tested in different settings and relevant data was collected. The same execution time measurement method as described in section 4.1.2 was used to measure the execution time of this function. In order to bring the collected data closer to the results of the realistic RSA algorithm, the values used in this execution time measurement are similar to those used for RSA encryption/decryption. In this case, a website was used to generating random primes with n -bits [47]. RSA-1024 was chosen and 512 bits primes were generated since RSA-1024 uses 512bits primes. Table 5-9 shows all the random generated values. A more detailed version of all the values can be found in appendix C.

Table 5-9: Values of random generated 512-bit primes and other parameters

Prime p	8056700366.....064671 (154 digits)
Prime q	1216410521.....653633 (154 digits)
$n = p * q$	9800255094.....099743 (308 digits)
$\phi = (p-1) * (q-1)$	9800255094.....381440 (308 digits)
e	65537
$d = e^{-1} \bmod n$	5481156449.....685953 (308 digits)
a	5
Ciphered a	4866274806.....581381 (308 digits)

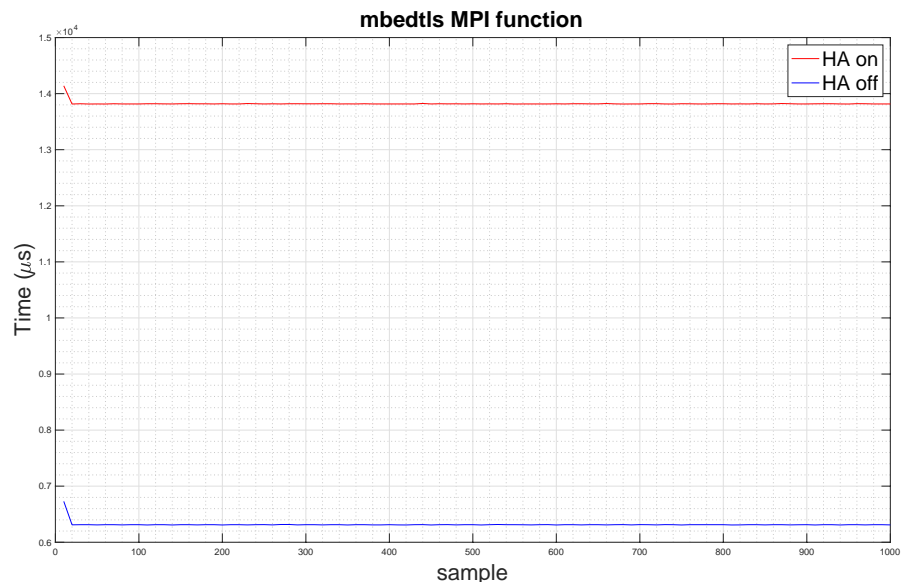


Figure 5.15: Execution time of mbedtls_mpi_exp_mod() with $a = 5$ and $e = 65537$.

Figure 5.15 shows the execution time of the function `mbedtls_mpi_exp_mod()` with small a and e values. In this case, $a = 5$ and $e = 65537$ are used to simulate RSA encryption. 100 samples were collected, and they show that the execution time is significantly longer with hardware acceleration than without. This shows that when computing smaller values hardware acceleration has a disadvantage compared to software. The reason for this may be that to initialize the hardware acceleration requires a certain time and when the number is relatively small the software is then faster. The execution speed of this function with hardware acceleration is 1.44 times slower than when running it in software.

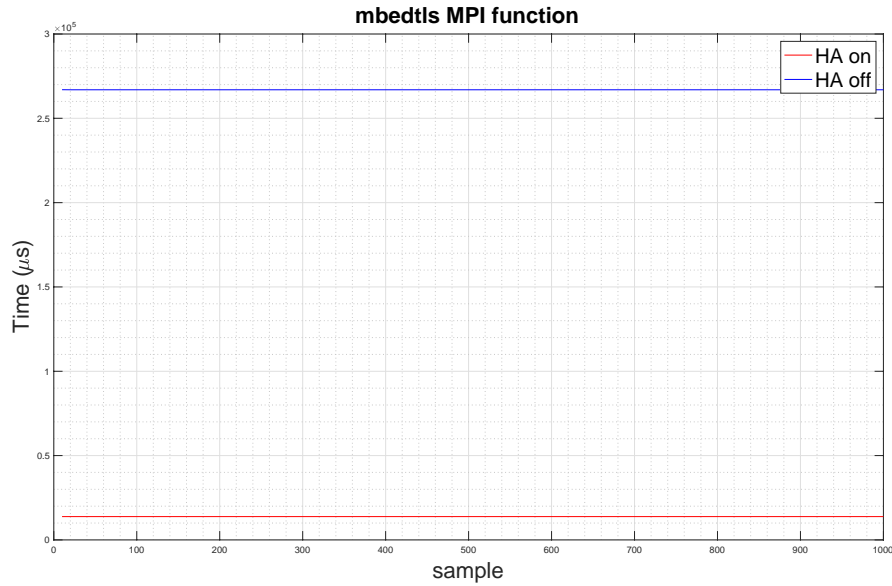


Figure 5.16: Execution time of `mbedtls_mpi_exp_mod()` with ciphered $a = 4866274806.....581381$ (308 digits) and $d = 5481156449.....685953$ (308 digits).

The result shown in Figure 5.16 is using larger values. The values used are ciphered a and d to simulating RSA decryption. And when the values become larger, that's when hardware acceleration becomes beneficial. As shown in Figure 5.16, using hardware acceleration results in a faster execution time than not using it, which is 12.84 times faster than software.

After that the value a is changed to a randomly generated value of 500 digits. The value e is a gradual value that goes from a 10 digits value to a 1000 digits value, increasing by 10 digits per step.

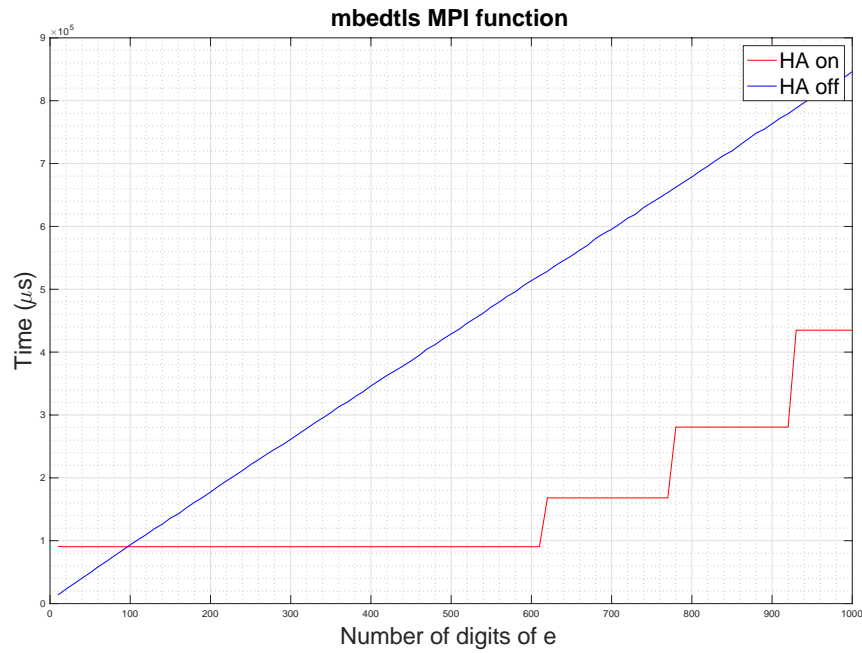


Figure 5.17: Execution time of `mbedtls_mpi_exp_mod()` with a large `a` and a varying `e`.

Figure 5.17 shows that hardware acceleration is slower than software when the value `e` is small, and the advantage of hardware acceleration can be seen when the value to be calculated becomes larger. In this case, when `e` is a number with 100 digits or more, the software becomes much slower compared with the hardware-accelerated one, as shown in Figure 5.17. The graph also shows that the execution time of the “HA on” has a stair-like curve. The reason for that is when the number is too large the hardware acceleration has to break it up into smaller groups, which takes more time to execute and forms a jump in the curve[48].

5.3 Discussion

As expected, the hardware acceleration feature effectively speeds up the cryptographic algorithms in most of the cases evaluated in this thesis. But for the RSA algorithm, the hardware acceleration had the opposite effect when doing encryption. The reason for this is that the hardware-accelerated version of the `mbedtls_mpi_exp_mod()` function is not as fast as the software when calculating small values, which is why the execution time of RSA encryption is slower than the software when hardware acceleration is enabled, because the numbers used in RSA encryption are relatively small.

After the evaluation of all selected cryptographic algorithms, it shows that the hardware acceleration on the ESP32 improved the speed for both encryption and decryption greatly for most algorithms. For AES, the overall improvement is 260% for encryption speed and 220% for decryption speed. For SHA, the speed is up by 165% and 181% for different CPU clock frequencies. For the RSA algorithm, the hardware acceleration did make the overall performance faster, but the RSA encryption speed is much slower. The encryption speed is down by 40% for RSA 1024 and 70% for RSA 2048. The decryption is not affected; the speed-up caused by hardware acceleration is 185% and 149% for RSA 1024 and RSA 2048, respectively for both CPU frequencies.

With a longer key length on the RSA algorithm, the security level is higher, but the execution times for both encryption and decryption are much slower. The throughput is lower with RSA 2048 compared with RSA 1024. The longer execution time also comes with higher energy consumption. This needs to be taken into consideration when designing a secure and sustainable IoT system.

5.4 Summary

This chapter presented the results of all three cryptographic algorithms, AES-128, SHA-256, and RSA 1024/2048. For most of the cryptographic algorithms, the hardware acceleration played an important role and effectively accelerated the speed of those algorithms. Except for RSA encryption, the hardware acceleration does not have a good effect on the algorithm, which dramatically reduces the throughput of RSA encryption.

Table 5-10 presents a summary of throughput increase for all three cryptographic algorithms when the hardware acceleration is turned on. The minus symbol for some of the data means it is a decrease in the throughput instead of an increase.

Table 5-10: A summary of the throughput increased with hardware acceleration for all three cryptographic algorithms.

Throughput increased (%)	Encryption 160 MHz	Decryption 160 MHz	Hash 160 MHz	Encryption 240 MHz	Decryption 240 MHz	Hash 240 MHz
AES-128	257.8%	222.7%	-	255.2%	211.1%	-
RSA 1024	-40.7%	184%	-	-40.4%	186.8%	-
RSA 2048	-71.1%	149.7%	-	-71.9%	149.3%	-
SHA-256	-	-	165.2%	-	-	181.3%

6 Case Study: Secure Communication with Databases

This chapter is about the case study where the ESP32 WROVER-KIT periodically sends non-encrypted or encrypted data messages to a remote database using Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS), which uses RSA algorithm. This scenario is performed under different configurations of the ESP32 WROVER-KIT, and the execution times of both protocols when performing data transmission are measured and evaluated. Section 6.1 describes the setup for this case study. How the measurement is conducted, and the measurement result is shown in section 6.2 and section 6.3, respectively. The discussion is in section 6.4.

6.1 The Setup

This section describes the system architecture used in the case study. The general use case is sending data messages to a remote database. The protocols HTTP and HTTPS are used, and the data message is sent to the database using a POST request. But sending data to a remote database using HTTP is not secure, the information can be hijacked by adversaries. Many IoT devices are collecting and transmitting private and confidential information, data security is extremely important in those cases. HTTPS can provide the security needed for data transmission between IoT devices and remote databases. Therefore, it is important to evaluate the effect on execution time when using HTTPS instead of HTTP.

A more detailed description of the setup follows here. The ESP32 WROVER-KIT is connected to the InfluxDB database. The database is created on the MacBook Pro (which is connected to the same Wi-Fi network as ESP32 WROVER-KIT) by InfluxDB. InfluxDB is an open-source time-series database (TSDB) developed by InfluxData [49]. It is designed to be used as a backing store for any use case involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics [50]. The ESP32 WROVER-KIT will perform post-request to the database with HTTP or HTTPS. This setup for both HTTP and HTTPS is presented in Figure 6.1.

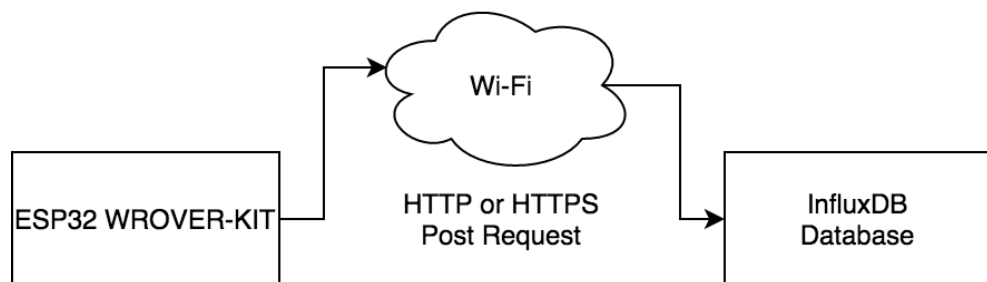


Figure 6.1: The base setup for communication between ESP32 WROVER-KIT and InfluxDB database.

The ESP32 WROVER-KIT is connected to the Wi-Fi for communication with the database. It will execute either HTTP or HTTPS POST-request to the InfluxDB database. The version of the HTTP/HTTPS used in here is HTTP/1.1 which includes more strict requirements than HTTP/1.0 in order to ensure reliable implementation of its features [51].

A Transport Layer Security (TLS) certificate is needed to enable HTTPS. The type of TLS certificate used in this project is a self-signed certificate generated on the MacBook Pro. A self-signed certificate is not signed by a Certificate Authority (CA), it will only provide cryptographic security to HTTPS request, unlike a CA-signed certificate, it does not allow the client to verify the

identity of the database. But in this case, it is sufficient enough to be used for the measurements. The self-signed certificate is generated using the following command shown in Algorithm 6–1. Besides the self-signed certificate, the command also outputs the private key file for the RSA algorithm. This self-signed certificate is based on the RSA algorithm, and the key length is 2048 bits. The self-signed certificate and the private key can be found in the appendix B.

Both HTTP and HTTPS are going to send post requests to the database. These post requests will write a message, which is “MT, one=1 two=0” into the database. The MT is the tag name, and “one” is the name for field_1 with a value 1, “two” is the name for field_2 with a value 0.

Algorithm 6–1: Command for generate the self-signed certificate and the private key.

```
sudo openssl req -x509 -nodes -newkey rsa:2048 -keyout /etc/ssl/influxdb-
selfsigned.key -out /etc/ssl/influxdb-selfsigned.crt
```

6.1.1 HTTP

Hypertext transfer protocol (HTTP) is a request response protocol to communication between client and server. Figure 6.2 shows how the communicate between ESP32 WROVER-KIT and the database is done using HTTP. The ESP32 client initiates a TCP connection with the database. After the connection is established, the ESP32 client will then send an HTTP post request to the database. The database will then process the received post request and send back a response to the ESP32 client. After the ESP32 client has processed the response, the TCP connection will be disconnected.

In order to send a post request to the database with EPS32 WROVER-KIT, the example code of HTTP request [52] is studied thoroughly, and a modified version of it is implemented similar to the previous measurements of cryptographic algorithms.

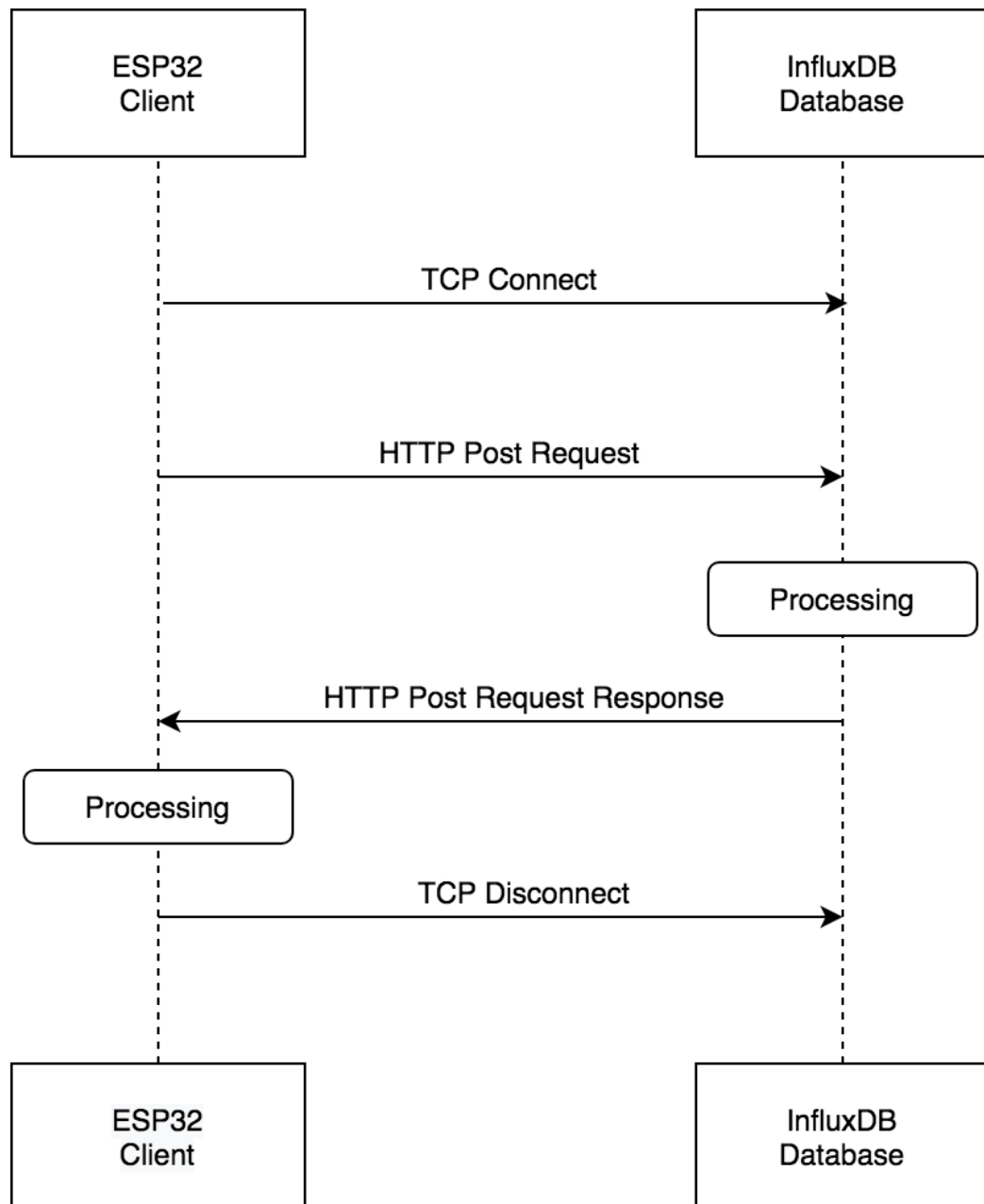


Figure 6.2: Diagram of HTTP connection between EPS32 WROVER-KIT and the database.

6.1.2 HTTPS

HTTPS is HTTP paired with transport layer security (TLS), also known as secure socket layer (SSL) to have the ability to encrypt the transmitted data between client and server. The S in HTTPS is short for secure. In this case, the client is the ESP32 WROVER-KIT, and the server is the InfluxDB database. Figure 6.3 shows the HTTPS connection, similar to HTTP but now it will have an extra phase, TLS handshake. The client and the server will exchange encryption key or cipher during the TLS handshake phase.

A more detailed diagram of how the TLS connection process is shown in Figure 6.4. The first step in the TLS connection process is the TLS handshake. The client sends out a **ClientHello**

message to the server. The server answers with **ServerHello** message back to the client. Then the server sends its certificate information and its public key, after this information is sent, the server will send **ServerHelloDone** message to indicate it has finished sending the initiation information. The client verifies the certificate information and then generates a pre-master secret based on the server's public key and the client's generated private key, then sends it to the server in the **ClientKeyExchnage** message. After receiving the pre-master secret from the client, a master secret will be generated by the server. The master secret will then be sent back to the client in the **ClientCipherSpec** message. Once this is done, a Finish message will be sent to inform the client that the upcoming communications between the client and the server will be encrypted. After this step, the EPS32 can perform post requests to the InfluxDB database the same way as in the HTTP part because the TLS process requires two extra roundtrips to establish the connection, compared to just one roundtrip for the TCP connection in HTTP. So, the time it takes for data transmission with HTTPS will be much longer compares to HTTP.

For the HTTPS implementation part, the setup and the post request message are the same as in HTTP. The example code: `https_mbedtls` [53] is studied and modified to suit this case study. The differences are now the database is using a self-signed certificate to secure the connection with the ESP32 WROVER-KIT, and the post request messages sent to the database are now encrypted using the self-signed certificate. Since the database is created using InfluxDB. The guide for enabling HTTPS encrypts communication between clients (EPS32 WROVER-KIT) and the InfluxDB database provided by InfluxData is followed [54]. Usually, the certificate will be checked on the client-side to ensure the server name is matched with the user request. But since a self-signed certificate is used in this case study, this procedure will not be possible and is then purposely disregarded. In real-world applications, it is suggested to use a CA-certificate instead of a self-signed certificate for better authenticity and security of the data.

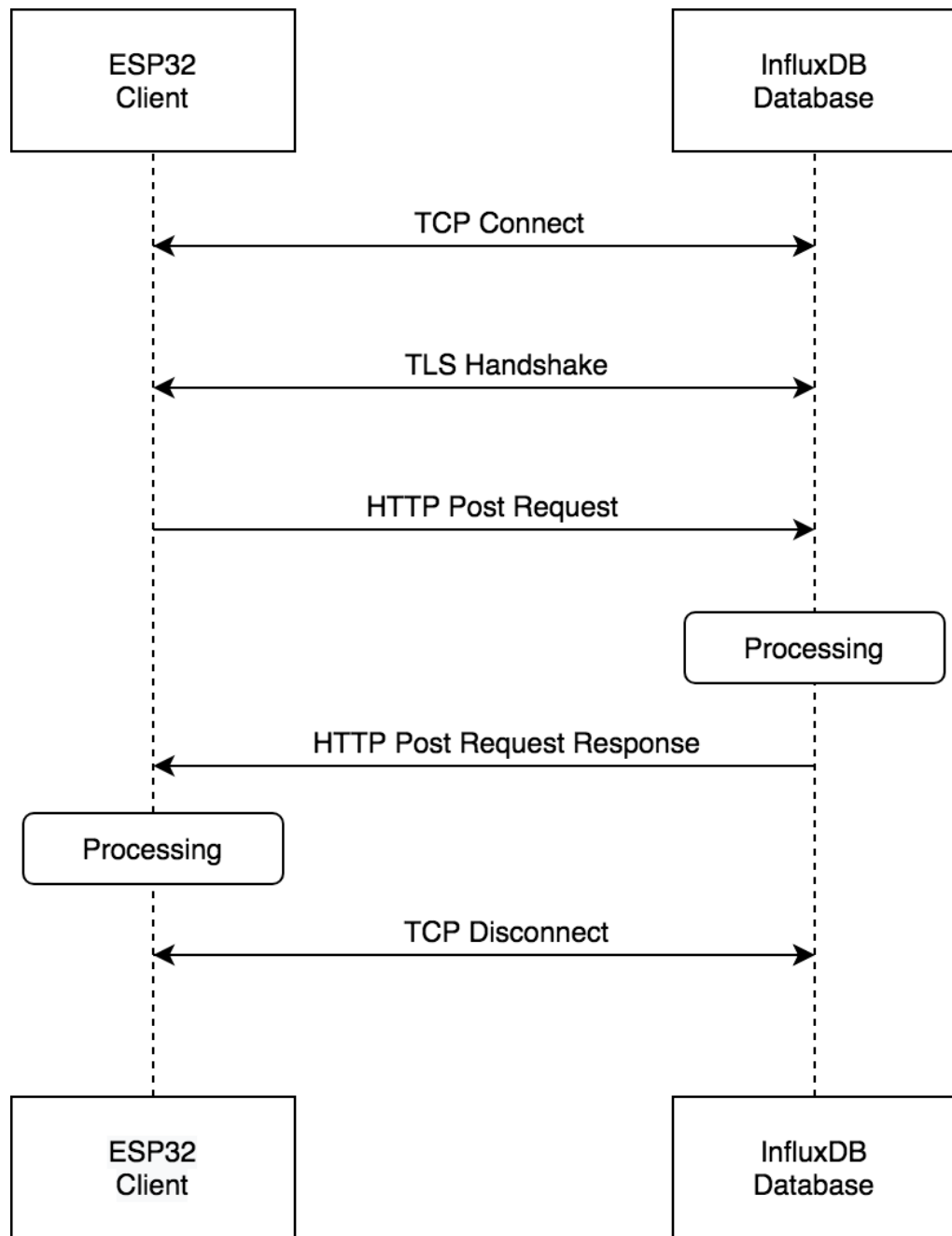


Figure 6.3: Diagram of HTTPS connection between EPS32 WROVER-KIT and the database.

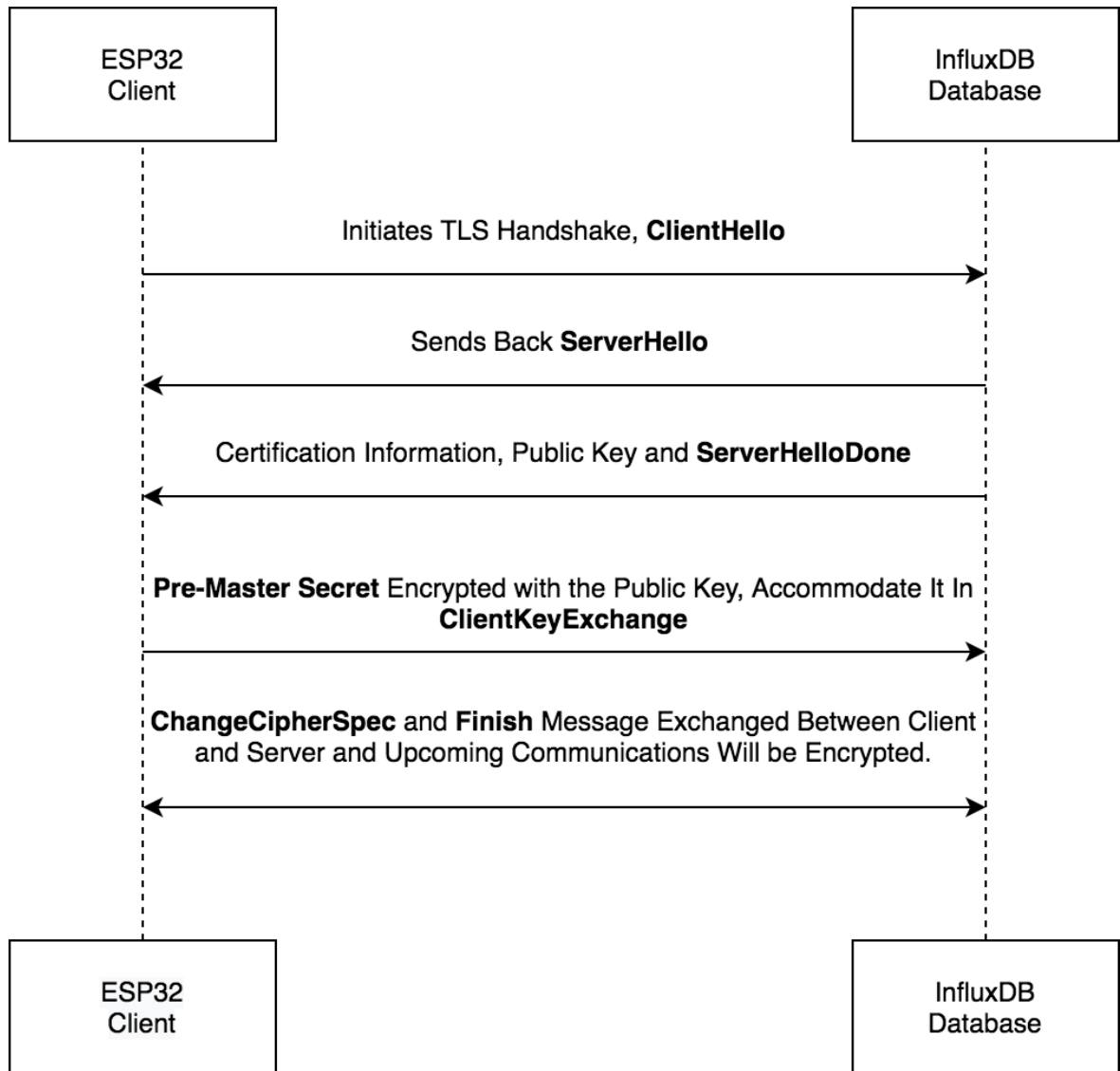


Figure 6.4: The TLS connection process.

6.2 The Measurements and Data Collection

After the setup, the transmission time for both HTTP and HTTPS is measured using the same *gettimeofday ()* function used in the previous chapter. The transmission time is the time taken for the ESP32 WROVER-KIT to establish the connection with the database and then make a post request to the database. For a representable average transmission time, 1000 data points were collected for each of the different configurations. With different CPU clock frequencies, in this case, 160MHz and 240MHz. Also, with hardware acceleration turned on and off. The collected data are then organized in MATLAB for evaluation and generating graphs.

6.3 Measurements Result

Figure 6.5 shows the transmission time for EPS32 WROVER-KIT sending data to the InfluxDB database using HTTP and HTTPS, respectively, in the form of a box plot. The same task is

performed with different CPU clock frequencies and with cryptographic hardware acceleration switched on/off. The average transmission time values for all different configurations are presented in Table 6-1.

The red crosses in Figure 6.5 are the outliers. An outlier is the data point, which differs significantly from other observations. Figure 6.5 shows that there are many outliers, especially for the HTTPS measurements. Several of the outliers differentiate a lot from other measured data points. A box plot shows all the outliers and gives a more accurate picture of how the measurements are. Therefore, a box plot was chosen to present the collected data.

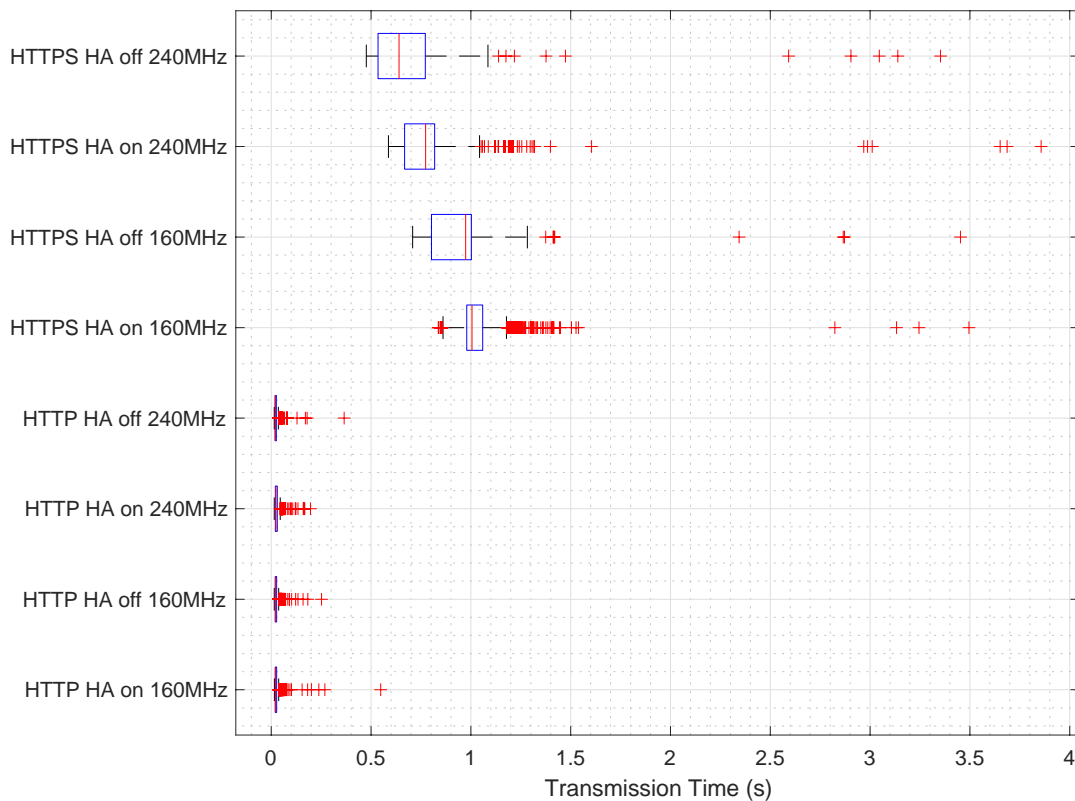


Figure 6.5: Transmission time for HTTP and HTTPS in different CPU frequencies and with/ without hardware acceleration.

Table 6-1: The average transmission time of ESP32 sending data to the database in different configuration.

Average transmission time (s)	HTTP 160MHz	HTTP 240MHz	HTTPS 160MHz	HTTPS 240MHz
Hardware acceleration on	0.0270	0.0283	1.0373	0.7890
Hardware acceleration off	0.0261	0.0248	0.9361	0.6665

The result shows that for both CPU frequencies 160MHz and 240MHz, when the hardware acceleration feature is activated, the average transmission time for HTTP becomes slower, though the differences are small (about 1 millisecond). But in some cases, the transmission times are much longer as what is shown in the box plot, most outliers are in the 0.25 seconds range and for HTTP 160MHz with hardware acceleration, the outlier has even past the 0.5 seconds mark which is almost 17 times longer than the average transmission time for the same case. For HTTPS, even it is using the RSA algorithm for data encryption and the hardware acceleration should make it faster when encrypting the data message, the result still shows that it is slower with hardware acceleration enabled, and the differences are more noticeable (about 0.1 seconds). The outliers are ranging from 1 second up to almost 4 seconds for all the different HTTPS configuration. Additionally, one clear observation from the box plot is that the transmission time for HTTPS is much longer compared to the transmission time for HTTP. It is about 36 times longer with a CPU frequency of 160MHz and about 27 times longer with 240MHz CPU frequency.

6.4 Discussion

HTTP can be used for transmitting data messages between IoT devices and remote databases. But HTTP lacks the security aspect for protecting the transmitted data messages against third parties. HTTPS addresses those kinds of problems by encrypting the data message with cryptographic algorithms such as RSA. However, it comes with a cost of slower execution time, more energy consumption, and the need for more powerful hardware. When it comes to IoT devices, areas like energy consumption and execution time become more significant since many IoT devices are resource-constrained. They do not have powerful CPU to calculate complex algorithms, nor do they have sufficient energy sources to maintain operation for a long time. For a resource-constrained IoT device like ESP32 WROVER-KIT, the result indicates that enabling the hardware acceleration for cryptographic algorithms will make the transmission time longer when transmitting data from ESP32 WROVER-KIT to the database using HTTP and HTTPS. With 160 MHz, the transmission time is 36 times longer, and with 240 MHz, the transmission time is 27 times longer. Even though HTTPS uses the RSA algorithm to encrypt the transmitted data message, the hardware acceleration feature does not accelerate the encryption of the data. But this agrees with the findings in chapter 5; when it comes to RSA encryption, the hardware acceleration should be turned off for faster encryption. The result from this case study further proves that it is true that for ESP32 WROVER-KIT, when performing RSA encryption, the hardware acceleration should be disabled. This result should be taken into account when designing and developing systems that execute RSA encryption on ESP32 WROVER-KIT. Also, as shown in the result section, HTTP is significantly faster than HTTPS when transmitting the same data message. It is suggested that if the data security is not essential, and the speed is the priority when transmitting data, HTTP should be chosen over HTTPS.

7 Conclusions and Future work

This chapter explains the conclusions obtained throughout the thesis, followed by suggestions for possible future works. Section 7.1 describes the conclusions. Section 7.2 possible future works are suggested.

7.1 Conclusions

In this thesis, an evaluation of selected cryptographic algorithms on ESP32 is presented. Three different types of cryptographic algorithms are studied. A symmetric key algorithm, AES 128. A hash algorithm, SHA-256. Also, A asymmetric algorithm with different key sizes, RSA 1024 and RSA 2048. The performance of those algorithms is then measured with the implemented measurement program. The implemented measurement program generates pseudo-random data messages for those algorithms to encrypt and measures the execution time. The measurements are taken in the form of execution time for encrypting/ decrypting or hashing the data message with various sizes. The same setting is also performed under a faster CPU clock frequency (240MHz). Using MATLAB, the throughput of each cryptographic algorithm is calculated and presented as a bar graph. Other results are presented in the form of graphs and tables for each algorithm. These results are then evaluated and discussed. The results obtained can aid the design and development of a secure IoT system incorporating devices build with ESP32. It helps the developers to decide the trade-off between performance and security level. The evaluation and results from this study shows that IoT systems that execute cryptographic algorithms supported by hardware acceleration will perform task generally more efficiently. With higher CPU clock frequency, the performance of all selected algorithms increased linearly.

An interesting finding from this project is that RSA 1024/2048 encryption running on ESP32 WROVER-KIT V4.1 is faster without using the hardware acceleration feature for both CPU frequencies (160MHz and 240MHz).

In the case study, the communications between the ESP32 WROVER-KIT V4.1 and a remote database over HTTP/HTTPS are studied. The remote database was set up with the help of InfluxDB on the computer, the ESP32 WROVER-KIT was connected to the same Wi-Fi network as the remote database. The execution time for the POST request using different protocols (HTTP/HTTPS) was measured. A boxplot showing the transmission time for all configuration is generated using the measured data. The result agrees that HTTPS is much slower compares to HTTP. It is suggested that if the data security is not essential, and the speed is the priority when transmitting data, HTTP should be chosen over HTTPS. In addition, because HTTPS is using the RSA algorithm to ensure the security of the connection between ESP32 WROVER-KIT and the remote database. When transmitting data messages, the speed is slower with the cryptographic hardware acceleration feature enabled. This result further approves that RSA encryption is indeed slower with cryptographic hardware acceleration performing on the ESP32 WROVER-KIT.

The goals of this thesis were met. A measurement program is implemented for each chosen cryptographic algorithms. The execution times of encryption, decryption for each algorithm were collected and analyzed. The performances of those algorithms were then evaluated. Overall, the SHA-256 is the fastest of them all. RSA 1024 and RSA 2048 are the slowest algorithms, plus they can only encrypt an extremely limited amount of data unless the data is divided into smaller pieces. AES 128 is more balanced; it can encrypt larger data and still have a relatively fast execution time. The result of the thesis shows that the RSA algorithm is not made for encrypting plain text files. Both their encryption and decryption time are the slowest of all selected algorithms and the size of the data that can be processed with this algorithm is limited. Therefore, the RSA algorithm should not be used for encrypting data. However, it can be used for encrypting secret keys for other

cryptographic algorithms such as AES. Then, using the AES algorithm to encrypt and decrypt the data. This way, the system is more secure, since the AES secret key can be securely and easily shared with all IoT devices. Plus, the encryption/ decryption time will be faster because the AES algorithm is better at encrypting and decrypting data compares to the RSA algorithm.

7.2 Future Work

- ESP32's hardware acceleration also supports elliptic-curve cryptography. This algorithm is not measured due to the time limitation. It can be interesting for future work to measure the execution time of this algorithm. It could be interesting to see how well elliptic-curve cryptography performs on the ESP32 device since the elliptic-curve cryptography with a smaller key size could provide the same level of security offered by the RSA algorithm with a large key.
- More Measurement of the variance of supported algorithms. Things like different key length or operation modes. To further evaluate different algorithms and see which algorithm and setting are most suitable for the task to be done on the ESP32 based IoT devices. This also eases the difficulty for designers when choosing the appropriate cryptographic algorithm for future secure IoT systems.
- For resource-constrained IoT devices, power consumption is an important issue. The future work can be power or energy consumption measurement for those algorithms. Since it is valuable info when designing a sustainable and long-lasting IoT system. This also aids in forming a more comprehensive comparison guide for secure IoT system development.

References

- [1] M. Somani, "Connect the Dots: IoT Security Risks in an Increasingly Connected World," *Security Intelligence*, May 11, 2018. securityintelligence.com/connect-the-dots-iot-security-risks-in-an-increasingly-connected-world (accessed Feb. 19, 2020).
- [2] Y. Harbi, Z. Aliouat, S. Harous, A. Bentaleb, and A. Refoufi, "A Review of Security in Internet of Things," vol. 108, no. 1, pp. 325–344, Sep. 2019, doi: 10.1007/s11277-019-06405-y.
- [3] "ESP32 Overview | Espressif Systems." <https://www.espressif.com/en/products/hardware/esp32/overview> (accessed Feb. 19, 2020).
- [4] S. Robles, *The RSA Cryptosystem*. Citeseer, 2006.
- [5] Q. H. Dang, "Secure Hash Standard," National Institute of Standards and Technology, NIST FIPS 180-4, Jul. 2015. doi: 10.6028/NIST.FIPS.180-4.
- [6] D. Blazhevski, A. Bozhinovski, B. Stojchevska, and V. Pachovski, "Modes of Operation of the AES Algorithm," p. 5, 2013.
- [7] A. Håkansson, "Portal of Research Methods and Methodologies for Research Projects and Degree Projects," in *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July, 2013*, pp. 67–73.
- [8] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC press, 2014.
- [9] "Japan Captures TOP500 Crown with Arm-Powered Supercomputer | TOP500." <https://www.top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer/> (accessed Jul. 11, 2020).
- [10] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.
- [11] J. Schwartz, "U.S. Selects a New Encryption Technique," *The New York Times*, Oct. 03, 2000. Accessed: Feb. 21, 2020. [Online]. Available: <https://www.nytimes.com/2000/10/03/business/technology-us-selects-a-new-encryption-technique.html>
- [12] C. Paar, J. Pelzl, and B. Preneel, *Understanding Cryptography: A Textbook for Students and Practitioners*, 2nd corrected printing. Berlin: Springer, 2010.
- [13] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.
- [14] "Dr Clifford Cocks CB | Graduation | University of Bristol." <http://www.bristol.ac.uk/graduation/honorary-degrees/hondego8/cocks.html> (accessed Feb. 22, 2020).
- [15] L. Goubin, *Cryptographic Hardware and Embedded Systems-CHES 2006*, vol. 4249. Yokohama, Japan: Springer Science & Business Media, 2006.
- [16] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, "A Critical Analysis on the Security Concerns of Internet of Things (IoT)," 2015, doi: 10.5120/19547-1280.
- [17] A. Mosenia and N. K. Jha, "A Comprehensive Study of Security of Internet-of-Things," *IEEE Trans. Emerg. Top. Comput.*, vol. 5, no. 4, pp. 586–602, Oct. 2017, doi: 10.1109/TETC.2016.2606384.
- [18] N. Sklavos and I. D. Zaharakis, "Cryptography and Security in Internet of Things (IoTs): Models, Schemes, and Implementations," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Nov. 2016, pp. 1–2. doi: 10.1109/NTMS.2016.7792443.
- [19] S. Vyakaranal and S. Kengond, *Performance Analysis of Symmetric Key Cryptographic Algorithms*. New York: Ieee, 2018, pp. 411–415.
- [20] Md. E. Haque, S. Zobaed, M. U. Islam, and F. M. Areef, "Performance Analysis of Cryptographic Algorithms for Selecting Better Utilization on Resource Constraint Devices," in *2018 21st International Conference of Computer and Information Technology (ICCIT)*, Dec. 2018, pp. 1–6. doi: 10.1109/ICCITECHN.2018.8631957.
- [21] N. Gura, A. Patel, A. W. H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8bit CPUs," 2004, doi: 10.1007/978-3-540-28632-5_9.
- [22] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "A Practical Performance Comparison of ECC and RSA for Resource-Constrained IoT Devices," in *2018 Global Internet of Things Summit (GIoTS)*, Jun. 2018, pp. 1–6. doi: 10.1109/GIOTS.2018.8534575.
- [23] N. Khan, N. Sakib, I. Jerin, S. Quader, and A. Chakrabarty, "Performance Analysis of Security Algorithms for IoT Devices," in *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, Dec. 2017, pp. 130–133. doi: 10.1109/R10-HTC.2017.8288923.
- [24] M. El-Haii, M. Chamoun, A. Fadlallah, and A. Serhrouchni, "Analysis of Cryptographic Algorithms on IoT Hardware Platforms," in *2018 2nd Cyber Security in Networking Conference (CSNet)*, Oct. 2018, pp. 1–5. doi: 10.1109/CSNET.2018.8602942.

- [25] O. Barybin, E. Zaitseva, and V. Brazhnyi, "Testing the Security ESP32 Internet of Things Devices," in *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S T)*, Oct. 2019, pp. 143–146. doi: 10.1109/PICST47496.2019.9061269.
- [26] M. S. Henriques and N. K. Vernekar, "Using Symmetric and Asymmetric Cryptography to Secure Communication Between Devices in IoT," in *2017 International Conference on IoT and Application (ICIOT)*, May 2017, pp. 1–4. doi: 10.1109/ICIOTA.2017.8073643.
- [27] K. Quist-Aphetsi and M. C. Xenya, "Node to Node Secure Data Communication for IoT Devices Using Diffie-Hellman, AES, and MD5 Cryptographic Schemes," in *2019 International Conference on Cyber Security and Internet of Things (ICSIoT)*, May 2019, pp. 88–92. doi: 10.1109/ICSIoT47925.2019.00022.
- [28] I. A. Landge and H. Satopay, "Secured IoT Through Hashing Using MD5," in *2018 Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, Feb. 2018, pp. 1–5. doi: 10.1109/AEEICB.2018.8481007.
- [29] P. Singh and K. Deshpande, "Performance Evaluation of Cryptographic Ciphers on IoT Devices," *ArXiv181202220 Cs*, Dec. 2018, Accessed: Sep. 07, 2020. [Online]. Available: <http://arxiv.org/abs/1812.02220>
- [30] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, "Security and Performance in IoT: A Balancing Act," *IEEE Access*, vol. 8, pp. 121969–121986, 2020, doi: 10.1109/ACCESS.2020.3007536.
- [31] "Espressif Systems - Wi-Fi and Bluetooth Chipsets and Solutions." <https://www.espressif.com/> (accessed May 23, 2020).
- [32] "Overview | Espressif Systems." <https://www.espressif.com/en/products/hardware/esp-wrover-kit/overview> (accessed Mar. 08, 2020).
- [33] "Esp32-wrover_Datasheet_En.pdf." Accessed: May 04, 2020. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf
- [34] "ESP-WROVER-KIT V4.1 Getting Started Guide — ESP-IDF Programming Guide v3.3." <https://docs.espressif.com/projects/esp-idf/en/v3.3/get-started/get-started-wrover-kit.html> (accessed Feb. 19, 2020).
- [35] "The Internet of Things with ESP32." <http://esp32.net/> (accessed Feb. 23, 2020).
- [36] "Get Started — ESP-IDF Programming Guide v3.3." <https://docs.espressif.com/projects/esp-idf/en/v3.3/get-started/index.html> (accessed Feb. 19, 2020).
- [37] T. S. Kuhn, *The Structure of Scientific Revolutions*. University of Chicago press, 2012.
- [38] A. Abdul Rehman and K. Alharthi, "An Introduction to Research Paradigms," vol. 3, 2016.
- [39] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," National Institute of Standards and Technology, NIST Special Publication (SP) 800-90A Rev. 1, Jun. 2015. doi: <https://doi.org/10.6028/NIST.SP.800-90A1>.
- [40] "ESP-IDF SDK v3.3," *GitHub*. <https://github.com/espressif/esp-idf> (accessed Jul. 08, 2020).
- [41] J. Dudovskiy, "The Ultimate Guide to Writing a Dissertation in Business Studies: A Step-by-step Assistance," *Pittsburgh USA*, 2016.
- [42] "The Open Group Base Specifications Issue 7, 2018 edition." <https://pubs.opengroup.org/onlinepubs/9699919799/> (accessed Mar. 01, 2020).
- [43] N. J. Salkind and T. Rainwater, *Exploring Research*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.
- [44] J. Aspnes, "Notes on Data Structures and Programming Techniques (CPSC 223, Spring 2018)." <http://www.cs.yale.edu/homes/aspnes/classes/223/notes.html#randomization> (accessed Oct. 04, 2020).
- [45] I. Ristic, *Openssl Cookbook: A Guide to the Most Frequently Used Openssl Features and Commands*. Feisty Duck, 2013.
- [46] A. Toponce, "Aaron Toponce : Do Not Use sha256crypt / sha512crypt - They're Dangerous." <https://pthree.org/2018/05/23/do-not-use-sha256crypt-sha512crypt-theyre-dangerous/> (accessed Sep. 19, 2020).
- [47] W. J. Buchanan, "Generating random prime with n bits." <https://asecuritysite.com/encryption/getprimen> (accessed Aug. 30, 2021).
- [48] "bignum.h File Reference - API Documentation - mbed TLS (Previously PolarSSL)." https://tls.mbed.org/api/bignum_8h.html#a55433a16c951178e2b98a01c6386239e (accessed Jul. 11, 2021).
- [49] "InfluxDB: Purpose-Built Open Source Time Series Database," *InfluxData*. <https://www.influxdata.com/> (accessed Apr. 29, 2020).
- [50] "InfluxDB 1.8 | InfluxData Documentation." <https://docs.influxdata.com/> (accessed Apr. 29, 2020).
- [51] "HTTP/1.1: Introduction." <https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html> (accessed May 10, 2020).
- [52] "ESP HTTP Client Example," *GitHub*. <https://github.com/espressif/esp-idf> (accessed May 03, 2020).
- [53] "HTTPS GET Example using Plain MbedTLS Sockets," *GitHub*. <https://github.com/espressif/esp-idf> (accessed May 03, 2020).

- [54] “Enable HTTPS with InfluxDB | InfluxData Documentation.” <https://docs.influxdata.com/> (accessed Apr. 29, 2020).

Appendix A: RSA Key Pairs

RSA 1024 key pair used in this thesis.

-----BEGIN RSA PRIVATE KEY-----

```
MIICXgIBAAKBgQChsxxIRSQHlDm+W3XJ5CCD+aDJbhC4yiSN5ptQkTch8zO7VS3O
mrOIvbc4AX799P/eWuCSNrFVlpfxh7Usb9I3959ZS/HKcoT5nsbUp9Y79CSDY3/e
lx7O/a/4P8DilsVg19aBwyZW+I65278LK9edZsCscQbgNvITLN1dXlUoVQIDAQAB
AoGBAIQpPZuAD2O2bb+9RzedIkp9GPOdxJh/aCRdC5UkKhIW+boRslPvh+sWBoJ
c/8jiSH3CwfWdfirXkMGFTldp+XTfEVNFCLRG4OqM2lEV3QGotEUDpBhSB/M1z+R
BMneLNFzXANSKeiMr5PLlNGpQPdYIvdVICzv4x/CXwosyQ3hAkEA1zn1A7OGsQci
FI9EasmGobMmjcy99s9hH1QLyEN8uzhwptvNXDpa5Eu47TDKGcfjU+PdSd4P6F5/
H/q49XAh7QJBAMBVNwuGkf7f4mZTGCqAtBZx6uEXg3bKZNORO/LzQ5htd/9TWMdD
8q4phypl/OifVxSuk83L61XE2WUNv5SfrwkCQQCzAoyMWIxl2YULZkdLKL6Sve
7YzqNFA2i2QFXERpP6dKoTS8SfkBOw4Dkfn2eW/VfYe3gZA/mCEqCxI9add9AkA7
tnk3h9kt8157/FJlZ74Bte6BYeZ77AxvrwiYvjP/UAVQbRUEELF4pwYDANULd+T2
U9kprKts5QfFmwYe45DZAkEAj7hwdwH5EqUicjes96IzvdYVusWYvRqk5m51C14d
hnNcAFP16AcCzCoK2Vz1S8clBaPAoCtoy+ot8cFNorKcOw==
```

-----END RSA PRIVATE KEY-----

RSA 2048 key pair used in this thesis.

-----BEGIN RSA PRIVATE KEY-----

```

MIIEowIBAAKCAQEA+c1gPn5D3NttDAUqGUh9hFnqcjn7L4xPgNK1ofrcYfopwZ7
Ysc99NToCa+iUdwBuraDNCX4K5840oW2cQuSoJtklMmdY8GmYtfBiomp7uqx5los
SxJGe+is5uMqAQjcumulI8W8cZtZlTkmOZ/369uT5cUUKiFRSVTCGyLwbU56svbD
N7datApcHV/cwG+mmsq14G3KxbbJO82tJjuFUdecBKxVk75T88SqY5MIOMowlb2E
p32wzfS2GaZUNtlVzjm2+2SeZr/l/Vk/wZ+hzJXqLZoTWT/bwLMsMS6tdgOl9qQ
bh6CQomJPdbliZL7IYC2L6IjjYWjIbPoRtvsuQIDAQABAoIBAGFLARNrLEjwWGmr
xkIQvxbfvmgkog59oK9uLnmxDu7Rbq2uoDhCEJUmlvRf56vNnsBdAhzygjGA9nc
w9iqttGGZIAS6iGMoepJGyugCTdNEzE8ueI9mOyU/GRqah6CAcTejk+OnVO9THeV
9OOSLiep02UMDs77MXYKUWbp5SjuJiVx4AozBU4mr1yanQKIbwRl+GVS3AwX/cdn
CYDHqTpsZEKQ/g65X3HXhFn+nVha5ZUfnGm8Jeo5M8I3K39CRFv2XI5I2K+2b5g1
uYOAxVW4RH4XRLXvajCQjSHxgg9qH3lt+HijmZFJCOG5WCp7U3bKewoMVfirRhvZ
LAFwL8ECgYEA5ck8EkldqPJybxFYAdNVQjNLgDyqVlzItU5Isl6h/tSGq4mYqSm
8545bm60oSVHp6ntQksEuYBr5ju+FXIJ5Yco1gwXhCfEYsE8pR8KFNoKvjHKumfC
RsYseq3Dn/Bq/yelGQSqg2P7biAcs7AauNANPqwo8w3WfpQ/RFJbH8sCgYEAzOH9
wG+RdUMZigZnl5wj1OSQ1d3NmrFckx6sg+ze7BMkhossR4VsVFHujPyGktptmt3dp
WgmYoP6gUFBrcCvqShLrbjkOHnDqHDLyRNPcNu/BGbaCUQNdbTYqkW6izYDvfYO4
WMQXmOKX+9lPcfgKhrPjyKvZyvK9LMT8FYXozQsCgYAFSKpT9VHtB54wBaMTb6uf
ORS8Xyi/kaSf5XQp454osFPD/ossq4KXSYUij/MhXXoXUuX8xoLVjSluAuOtFJE7
sO87GM4VoOB4MN9XIK6XqjmMBBMYVDh4bigprgM4kouykFPQjnXpTYxDjQN78km7
BDZSsz5zsFd5fzMLjrEoiHwKBgEA7/jWOBRmxjtlCRso2rtk5S9txbdu5DaAmdsnp
wr5bBhGUoq5Nu3f4K96D8aErDhcpPgN3jL2o9wgXHmfo4aCVq/BjvPR4TD3JDoHa
9mjW5ECugs7pcgmHQPnVr38klHFSkcJqwEczi5jvOcQukwZGcf1hntRNJFhb7fjF
hcmfAoGBALQ8PIeKvPhVs+zgxq5QVI6sE2Br5TBNRicQDybOTBo8JF7oCO1Kaky9
lj9B9Gyw+uChvifTMGbCAGxOYsML8MBMajShUE21JiKAGMPOxmYtRmD2yHJlWXi
oyoWdmhw5AYjeZnr9mB3uT2MFyCiEoM3zlVHwRoNMaoock4PmCwzu

```

-----END RSA PRIVATE KEY-----

Appendix B: Self-signed Certificate and the Private Key

The self-signed certificate used in the case study.

-----BEGIN CERTIFICATE-----

```
MIIDazCCAlOgAwIBAgIUIfT64QsJRc7iZguh+1Ect1WXj+swDQYJKoZIhvcNAQEL
BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM
GEludGVybmVoIFdpZGdpdHMgUHR5IEExoZDAeFwoyMDAzMTgxOTEyMjRaFwoyMDAo
MTcxOTEyMjRaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVNoYXRIMSEw
HwYDVQQKBDBhJbnRlcmlldCBXaWRnaXRzIFBoeSBMdGQwggEiMAoGCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQQDOQXP8LTw6Q4FUF62FqIi2fZ/Vnb29adeDOcCjyjoX
Y/+343HcnJ3dBdovjyFfZjdJVNnu1LaL7jdbUXKgjz43joTCl6/44UigQKeE7dd
JUNiUXJthLqRi/Mnp+RjtNLwVo9eXjXCm8I7YsnBQUmvJDRXgTB2D7uxGsInOpIe
o9rFjvcMUcKucrXSwFGB95JENiBLOUB1kjPLGAwtri2ENq77ot/adboFOT4KLIG
BgPMpHNm7FjEj5aar6iM+k+HoiOqtfZrKM4m4mkt4q9QzEP4oDdzWAM6Tomocbs
ucA5W8HMYTWTUfVKAkvbaITh8rCmO74rD5TNwnpO8x7FagMBAAGjUzBRMB0GA1Ud
DgQWBbTWjyCd89aLjthzZtbHK11f3yKANDafBgNVHSMEGDAWgBTWjyCd89aLjthz
ZtbHK11f3yKANDAPBgNVHRMBAf8EBTADAQH/MAoGCSqGSIb3DQEBBCwUAA4IBAQA3
IKOBB/gyEg/3wKX+a/6M82Rel3vJLRCDR/XuXa1dJNKz/hIKqzUB8Yo4PIPySN9Y
PsRobyQqBdAisPmgv/oQt7h/qQJJtpqPrLR54AxAOlj6UIoCAF+4Eeo1HcyzfXph
IgbBgq3OO+Bm4xO/eeqTNnhGpGTJ+sH25uX2MNgTDxqi8m3coWOxAQEbIPmXg3eP
LzYUQrsigOoke2wIkx+d6qmTgc+dWoN6rj9gJm+Ymqpy2SMMZxy6IEbogczGsJJ
CkXFnvysFkikrdhI99Ehuvh9pY6swUCm3NOSTnERaWobsVe9tc/4NMbq6oe5/SQO
W6kjQXqSDcPEZAvAELJF
```

-----END CERTIFICATE-----

The private key of the self-signed certificate.

-----BEGIN PRIVATE KEY-----

```
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBAgEAAoIBAQQDOQXP8LTw6Q4FU
F62FqIi2fZ/Vnb29adeDOFCjyjoX/+343HcnJ3dBDtovjyFfZjdJVNnu1LaL7jd
bUXKgjz43joTCl6/44UigQKeE7ddJUNiUxJthLqRi/Mnp+RjtNLwVogeXjXCm8I7
YsnBQUmvJDRXgTB2D7uxGsInOpIeogrFjvcMUcKucrXSwFGB95JENiBLOUB1kjPL
GAwtri2ENq77ot/adboFOT4KLIGBgPMpHNm7FjEj5aar6iM+k+HoiOqtfZSRKM4
m4mkt4q9QzEP4oDdzWAM6TomocbsucA5W8HMYTWTUfVKAkvbaITh8rCmO74rD5TN
wnpO8x7FagMBAAECggEAfx+WZiJXUa3ToonouE2U3a11+wp7LKhonTvz1m/K4Ys5
zFQSFKAxKE74ek7aLIip0/uyeWfafAIX1doIvvUgsV8Ak6C7n1eS8h5Dx7xtMqrJ
2qP6JAg/mX+BgbqYKOAfDv5S5R35opV/1Gxh34vm51zQKfLFNK8jxtewPM97ygem
CMnvdyuvbFAfRhvvzWSO8l9qgi/FGdS4slgWS7UmPnOcdae8/gtGo6KSXWrg3Egs
XsI9lHfGgSk2PpI5O9qtxa+rZx/LWsJEr8oLNvt+4Lz2i6JPNXcfqFZlQ6kzz9sl
nnIwF6Vs9yOOazUv2sDnrfDwGLCQqvy81ryyvDmDQKKBgQDz2Rdoh3YBvQ7Eu9lE
n+31m+f9AvqW6g8ES8+VSDUO55hzA/g7weFvKvR6yMQXs269v7jkjQfHUTpRuTnC
m+NqWcd3aFTW44/SZuRa+EmubJMICc982HmQX9uTS42rV9cABunBCK56WXEioKjf
DtxhhaPVoo6EnjYReXNOLaLE9QKKBgQDYiMZRGcZ4RKbtiFPdB+8rNJUNoLUBKQmz
VzLCjrnL4o3CH6Qepm23lSHyogLkhIDCBFXB2kvUoar4vZYE4c9Jni9fxuDLGCuM
3zxTSvTn58l4qo4Q36Qhjd8TzaO8nDzezFbfw9JXD5vKttPrl5DERQPgCk3C7gHQ
3JgffkFQkQKBgQDhLoRvslCmsFl4EmVE+o/7tXsZ6mtRfFpbxEqxeBoPZo4hALR3
MscIKwzUapyzqm8E6Yor+pUJWpi5qYyOi/kKWfhci8t1GnbRfVU6MPmSCxAlpo6l
4x731Z3nOylb5uVUUA4DximiMQDKftjEDGw13vz5yCftTlFt8E/kDT64FQKBgEz/
FcYV54UMpG8Ccw+9qoYqhsW+2R44gYKMFzinl9mwUg4dQjbD52/IT/IQS4cGtuJG
uFfeT+h/ORSKfbZWtWbwINhl17lgNtw45TKZDMoJ1tJZk3rci3iDAjgAf6CpzdL
vfiDoV/j5PoIWQMFOio1eJNKFqyiBXNHfbB1LLSBAoGBAJEZVPCLjFkSqINFMtoy
woWrnotGo9YQJ6WjAZ+Oi4XqaZjvOqrJ9o7VvqB3vL6g2ZishyMsK49K//iZPyrF
8H3ffZtAugddr8XVE1/WEXppPEzY8rec+kgx74zHLhRoXiMFhU8DvWMJEIY6L//J
MWuCWtTPqZ6enlvXxkZgGfre
```

-----END PRIVATE KEY-----

Appendix C: Random Generated Primes

No of bits (n) in prime is 512

Random n-bit Prime (p):

8056700366977028148324436773563845582872903913142936199972829572691740209289945987
233443500931531472239379998959962541289556293680672863892581252180064671

Random n-bit Prime (q):

121641052142803257472529813661872719599165795994277263129360619125896817653320288607
66057911865654227471156636227049514883083367872471559762139859738653633

$N=p*q=$

980025509438394820680242513342397298931866952449591847835956797548469764708086260
97648112478156155885510196006456218205127510001651613168430092466526160431940045567
83455499125707263099424998764944645747851426665445327914006944247970654113751360527
6433120364450424770516536365602448709519683637315328409099743

$\text{PHI}(p-1)*(q-1) =$

980025509438394820680242513342397298931866952449591847835956797548469764708086260
976481124781561558855101960064562182051275100016516131684300924665261604117192399865
77201095679654491243132444859962944907851753745561793858647467857731693138012192479
247420653913789583504480192962787156375259982594216490381440

$e = 65537$

$d =$

54811564494796715988241160083696584517217222752166470223811221840092788433419585592
6147659303040379911727836858667046415115698246874014537686590669095302459855810071
8679110977069526713193732752333920964725877262904652321734080423054606528038663507
8418831448625296296653389947554175510471623348673396878685953

$a = 5$

Ciphered $a =$

4866274806114620445903248848404555163907422350888306318787214119209863453915727735
348908034859176084266546409945964050332408611885031128878775980723158012670738579
82170455134203036875789142761099509117540796145065200313445499410040931616375677448
01327110318690697933984218072509449694773107816332071725581381

TRITA-EECS-EX-2022:41