Teesside University

Performance Evaluation of Cryptography Algorithms in a Virtualized Environment

Toluwani Adefisoye
Student ID: B1107803
toluenenate@gmail.com

Supervisor: Dr Paolo Modesti

CHAPTER 1

Introduction

1.1 Overview

Wireless cryptographic protocols place a high priority on data security, and a cryptographic method is essential to maintaining network security. In recent years, "Lightweight Cryptography (LWC)" has become a ground-breaking technique. In situations with limited resources, such as RFID, sensor networks, healthcare, IoT, cyber-physical systems, distributed control systems, indicators, meters, custom controls, smart energy systems, etc., a lightweight cryptographic scheme is ideally suited for use (Ibrahim and Agbinya, 2021).

IoT devices have a number of limitations, including a lack of resources, such as low processor speed, memory, and battery life. It is therefore essential to pay great attention to these gadgets, especially when it comes to data processing. All wireless network nodes exchange a lot of data, which introduces additional hazards and difficulties, such as the need for limited resources like electricity and money. Despite this, security only receives a small part of the available resources.

A vast network of internet-connected gadgets that capture and share tremendous amounts of data is known as the Internet of Things (IoT) (Bandyopadhyay and Sen, 2011). The enormous number of linked devices has led to the IoT becoming an essential aspect of the lives of billions of people globally.

By connecting intelligent "things" in the real world with cloud-based IT systems, the Internet of Things (IoT) has the potential to lead to the next wave of innovation. To be successful, the IoT must secure data and privacy, and this will raise new problems for cryptographic security, identity management, and credentialing (Li et al., 2015). However, there have been a number of recent security incidents involving IoT that have had catastrophic effects when used with open and interconnected devices, such as smart city infrastructure (such as smart transportation, traffic lights, and meters) or industrial IoT devices like programmable logic controllers (PLCs), robots, and machines (Sundmaeker et al., 2010). Hence, while selecting and deploying IoT-related devices, security must come first.

Lightweight messaging protocols that can function with the constrained hardware and resources found on IoT devices are often preferred by IoT applications. To address various application-specific difficulties in IoT networks, numerous application-layer communication protocols have been created. Maximum throughput, minimal energy usage, and low latency are the goals of these communication protocols. The amount of data exchanged and the security measures employed might have a considerable influence on system performance because IoT applications contain several devices. Consequently, choosing the right protocol for a particular application is essential. The implementation of a low-cost (lightweight) communication protocol that takes into account the communication constraints of IoT components is one of the key strategies for reducing energy consumption and latency while maximizing throughput.

There is a growing need for secure communication due to the rise in linked computer devices and the frequency of cyberattacks. Nevertheless, because sensor networks often use devices with low-performance hardware, traditional cryptosystems are frequently inappropriate for usage in these applications. The NIST has started the Lightweight Cryptography Project to make it easier to create, assess, and standardize suitable lightweight cryptography algorithms (Madushan et al., 2022). The ultimate goal is to standardize lightweight cryptosystems that provide authorized encryption with accompanying data, as well as lightweight hash functions. Around 200 different AEAD encryption implementations have been submitted to NIST, which has published 56 method proposals.

It has become more crucial for IoT devices to adopt encryption in settings with different limits since it is an efficient method of thwarting cyber threats. Traditional cryptography, which is appropriate for servers, PCs, tablets, and smartphones, and lightweight cryptography, which is appropriate for embedded systems, RFID, and sensor networks, are the two types of cryptography that are frequently employed.

Research and development have gone into a technology called "lightweight cryptography" that makes it possible to encrypt data securely even on low-powered devices (Khanam et al., 2020). However, this technology has a number of difficulties, including key management, the trade-off between security and performance, and the development of cryptographic algorithms that are appropriate for devices with limited resources. Lightweight cryptography has a major issue in key management since sophisticated cryptographic key exchange protocols cannot be implemented on devices with little processing power. It is extremely difficult to strike a balance between security and performance since cryptographic algorithms that are geared toward performance may not offer the required level of security. It is difficult to balance security and performance when creating lightweight cryptographic algorithms that are also optimized for devices with minimal resources.

Concurrently, improvements in algorithm performance are leading to developments in cryptography systems. The Advanced Encryption Standard (AES), Rivest Shamir Adleman (RSA), and Data Encryption Standard are examples of these methods (DES). In lightweight cryptography, the idea of AnBx, often known as Alice and Bob, is frequently used to explain the configuration of cryptographic algorithms on hardware with little resources. Its goal is to cut down on both code size and computation time. Alice and Bob are the two people that AnBx represents. They want to converse safely across an untrusted channel. To accomplish their goal, they trade a public pilot and calculate the opposite channel's impulse response (CIR), frequency response (CFR), and received signal strength (RSS) (Zhang et al., 2016).

The AnBx compiler, which uses Alice and Bob notation, is used in this research effort to configure security protocols for lightweight cryptography. The performance of the LWC security and protocols is then assessed based on resource consumption, including execution time, consumption time, and memory use. The AnBx notation is then translated into Java code. Following that, data analysis and visualization tools are used to validate the evaluation's findings.

1.2 Problem Statement

The use of lightweight cryptography (LWC) to encrypt communication between IoT devices is becoming more and more common. To maximize resource usage, it is essential to integrate LWC into older systems; nevertheless, setting up highly secure LWC cryptography offers a number of difficulties. The needs of various devices must be met, and it is important to choose the right encryption schemes, key sizes, key lengths, key pair generation algorithms, and safe random number generation algorithms.

1.3 Aim

To assess the resource usage of the protocols and create lightweight security protocols using AnBx notation.

1.4 Objectives

The followings are the objectives of this study;

- To set up the AnBx compiler extension on the Java compiler environment
- To implement LWC security protocols.
- To identify configuration parameters to evaluate resource utilization based on the cypher, key size, and secure random generator.
- To validate the experiment by comparing it with past related works.

1.5 **Justification**

The need to identify the best combination of configuration parameters for security protocols is important to optimize the resource utilization of cryptography algorithms.

Exploration of AnBx notation for configuration security protocols to aid the design and implementation of the cryptography process abstraction.

CHAPTER 2

Literature Review

2.1 Background

In the last ten years, there has been considerable advancement in the creation of resource-constrained IoT devices and lightweight cryptography. The main goal is to develop and use straightforward cryptographic algorithms that can provide adequate levels of security. IoT application implementation can be split into two categories: software and hardware. Speed, space, and energy usage are crucial factors in hardware implementation. The required memory size (ROM and RAM) of the embedded software must be considered when implementing software. Consequently, it is crucial to take these limitations into account when selecting a security strategy for devices with limited resources.

2.2 AnBx Abstraction

The AnBx abstraction is a useful approach for configuring lightweight cryptographic algorithms with minimal resource requirements while maintaining security. AnBx has been widely adopted in various applications and has also been used in the development of standards. However, it is important to ensure that the authentication and authorization processes in the system are carefully designed to avoid potential flaws.

There are two persons involved in the EAX mode of operation: Alice, the sender, and the recipient (Bob). The sender encrypts the message's plaintext, creates a tag, and sends it to the recipient along with the ciphertext. To access the original communication, the receiver then validates the tag and decrypts the ciphertext. A lightweight block cipher is used to carry out the encryption process in EAX, while a message authentication code is used to carry out the authentication process (MAC). According to Agrawal et al., 2020, the MAC is intended to provide strong security and thwart assaults, whereas the lightweight block cipher is created to be quick and effective in terms of code size. On the other hand, the OCB mode of operation uses a single algorithm to streamline implementation and cut down on code size for both encryption and authentication procedures. High-level security is guaranteed by OCB while the algorithm's computational and memory requirements are kept to a minimum.

Widespread use of the AnBx notation as a standard abstraction for implementing lightweight cryptographic algorithms has occurred, particularly in Internet of Things (IoT) devices, mobile devices, and smart cards (Noura et al., 2022). Standards like IEEE 802.15.4, which outlines the communication protocol for low-rate wireless personal area networks, include AnBx integrated into their architecture (LR-WPANs).

2.3 Lightweight Cryptography Evaluation

In order to evaluate the performance of lightweight cryptographic algorithms on hardware platforms like the Raspberry Pi and Arduino (El-Hajj et al., 2023). They selected 122 block ciphers and assessed their performance for different cipher and key sizes during the encryption and decryption procedures in terms of speed, cost, and energy consumption. The most effective ciphers in terms of speed and ROM consumption, according to the researchers, were LEA-128-128, COMET-64 CHAM-64-128, Hight-64-128, Speck-48-72, Speck-64-128, and XTEA-64-128.

Podimatas and Limniotis (2022) notably focused on the Saturnin family of ciphers when examining the performance of lightweight block ciphers in resource-constrained settings. They used an experimental approach to assess Saturnin's performance when used in a specific context with limited resources and to compare it to the Advanced Encryption Standard (AES). The results showed that Saturnin, which is based on the AES design and can be up to twice as fast as AES in these constrained situations, can achieve considerable performance benefits.

In order to give a general overview of the present state of lightweight cryptography algorithms utilized in IoT contexts, Regla and Festijo (2022) undertook a survey of the literature. They gathered pertinent information on several algorithms, including their level of security, performance in terms of encryption and decryption, execution time, memory consumption, clock speed, latency, and frequency. Comparative tables with the obtained data were displayed for additional analysis, review, and assessment. The study indicated prospective research paths for more exploration and offered insightful information about how lightweight cryptography algorithms work in IoT devices and surroundings.

El Hadj Youssef et al. (2020) conducted a thorough literature review for their study in order to identify the different lightweight cryptography algorithms used in IoT environments and to collect pertinent information, such as security level, encryption and decryption speed, execution time, memory usage, clock speed, latency, and frequency. The acquired data was arranged by the authors into tables for additional analysis, appraisal, and assessment. The study identified potential routes for future research and offered insightful information about the performance of lightweight cryptography algorithms utilized in IoT devices and surroundings.

Kim et al. (2019) looked into the HIGHT block cipher's resource-constrained capabilities for IoT platforms, including Application-Specific Integrated Circuits and 8-bit and 32-bit ARM Cortex-M3 processors. For these platforms, they provide optimized hardware and software implementations.

Singh et al. (2017) investigated several aspects of lightweight cryptography (LWC) and proposed a hybrid LWC method that may be used with Internet of Things (IoT) gadgets. Based on the device's memory storage, processing capability, and power requirements, they suggested selecting a certain LWC algorithm. The time frame prior to 2016 was the main topic of the article.

2.3 Cipher Algorithms

2.3.1 ADVANCED ENCRYPTION STANDARD (AES)

AES was developed by the National Institute of Standards and Technology (NIST) in 2001 to replace the Data Encryption Standard (DES). AES was created with the goal of having stronger mathematical characteristics and being simple to implement in hardware and software. It works with keys that are 128 bits, 192 bits, and 256 bits in size. The key size for AES encryption that is currently most frequently used is 128 bits.

A lightweight encryption technique created exclusively for the protection of voice communication sent over wireless networks is presented by (Hazzaa et al., 2021). The method is meant to retain a high level of security while using less power and fewer resources to execute encryption (confidentiality). The suggested approach makes use of methods that are comparable to those used by the AES algorithm.

The number of rounds in the AES algorithm, N, which is a type of block cipher, varies depending on the size of the key. There are 10 rounds if the key length is 128 bits, 12 rounds if the key length is 192 bits, and 14 rounds if the key length is 256 bits. A block of data with a size of 128 bits is split into four blocks of equal size in the AES method, and each cycle of the algorithm processes this state, which is arranged in a 4x4 matrix (Abdullah, 2017).

Each round of the AES algorithm consists of four operations: substituting bytes, shifting rows, mixing columns, and adding round keys. With only three operations—substitute-bytes inverse, shift-rows inverse, and add-round-key—the final round is a little different. In this final phase, the inverse operation of the mix-columns procedure is used instead. Decryption uses the inverse operations of the four operations (substitute-bytes inverse, shift-rows inverse, mix-columns inverse, and add-round-key) to recover the plaintext (El-Hajj et al., 2023). Encryption starts with add-round-key, which XORs the plaintext with a key.

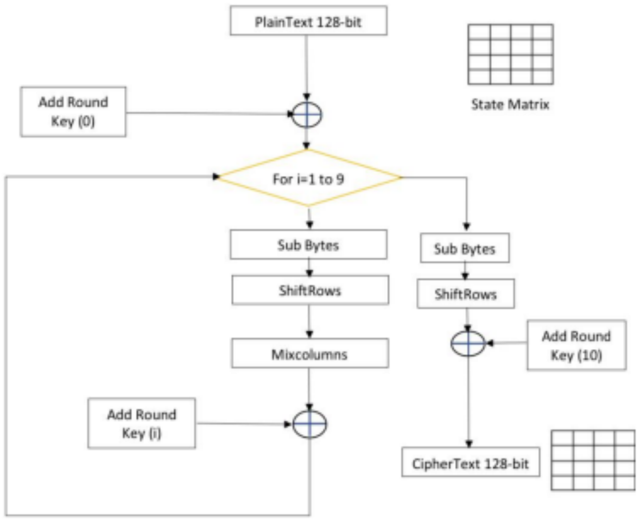

Figure 2.1: AES

A modified AES version that substitutes a permutation step for the mix columns overhead from data was proposed by Kawle et al. (2014). This simple encryption method is ideal for large plaintext and offers faster encryption for multimedia data security.

Sari et al. (2019) introduced an advanced approach to enhance the security and message capacity in image steganography, by utilizing AES and Huffman coding to decrease the total number of bits in the message and maximize the capacity. The method proposed in the paper efficiently and securely embeds a message image into a cover image through discrete wavelet transform (DWT), while reducing the usage of bits.

A modified AES method was presented by (Kumar et al., 2019) with the goal of encrypting voice signals sent through peer-to-peer networks. The only difference between this modified algorithm and the standard AES algorithm is the removal of the mix columns step. On the Artix-7 and Kintex-7 FPGAs, two separate FPGAs, the proposed algorithm was tested and put into practice. Based on the algorithm's execution time, power consumption, and area utilization, the authors assessed the algorithm's performance. The outcomes showed that the modified AES algorithm was able to retain low power and area use while achieving high levels of security.

2.3.2 Blowfish Algorithm

Bruce Schneier developed the symmetric-key block cipher method known as Blowfish in 1993. According to Schneier, the algorithm's prominent characteristics include the use of key-dependent substitution boxes and a complicated key scheduling (2005).

With a configurable key length ranging from 32 to 448 bits and a Feistel structure, the Blowfish algorithm is a symmetric block cipher. It has 16 rounds and a key-dependent S-box with a block size of 64 bits. Four S-boxes are utilized in the process, and the same algorithm is also used for decryption. The technique is extremely safe because of the key schedule and key-dependent S-box. Due to its enormous key size and several rounds, Blowfish offers strong defense and is immune to multi-key attacks, making such attacks unlikely.

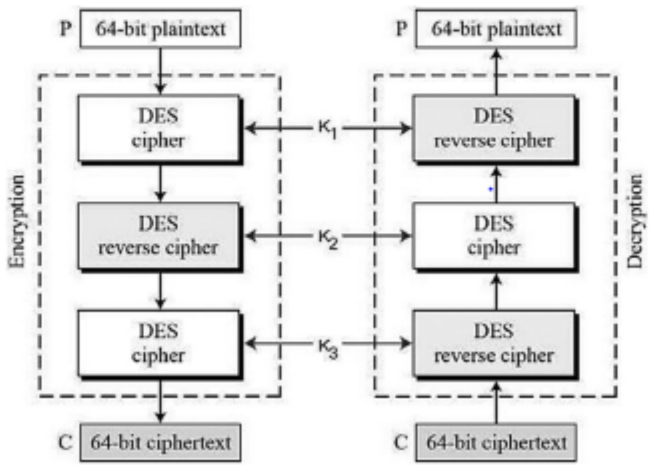

Figure 2.2: Blowfish Algorithm

2.3.3 DES

A sort of block cipher used in cryptography is called Triple DES, commonly referred to as Triple Data Encryption Algorithm. It was first used in 1998, and its name refers to the way it uses to encrypt and decrypt each block of data three times in the order of encryption, decryption, and encryption. The key length for 3DES can be either 112 bits or 168 bits, and the block size is 64 bits. Without having to create a brand-new block cipher algorithm, Triple DES was created to increase the security of the original DES algorithm against brute force and cryptanalytic assaults.

Kumar et al., 2011, compared the vulnerabilities and defenses of symmetric and asymmetric cryptography methods using a theoretical analysis.

IBM created the symmetric encryption algorithm known as DES in 1972, and the National Bureau of Standards (NBS) later approved it as the Federal Information Processing Standard (FIPS) in 1977. It is based on an earlier Horst Feistel invention known as LUCIFER and employs a single key for both encryption and decryption. The Feistel block cipher, which is currently used by the algorithm, necessitates the definition of the round function, key schedule, and other relevant processing steps. The plaintext, broken up into 64-bit blocks, and the secret key (56 bits), are the inputs used in the encryption process. Due to substitutions and permutations, the output is a 64-bit ciphertext that is challenging to crack. DES has some drawbacks, despite benefits such the difficulty of brute-force assaults caused by its 56-bit key size and the algorithm's resistance to uncovering vulnerabilities. One drawback is that by giving a S box two carefully selected inputs, the identical output can be produced. The initial and final permutations are both ambiguous and puzzling. Despite these drawbacks, DES is still a popular encryption scheme.

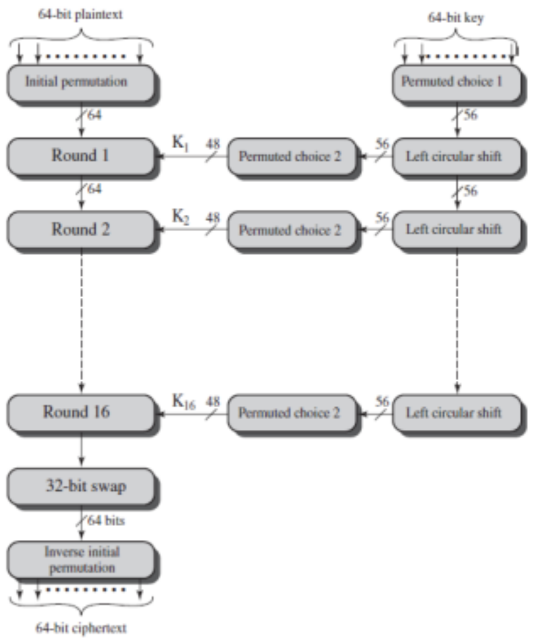

Figure 2.3: DES

2.3.4 RSA

In the late 1970s, cryptographers Ronald Rivest, Adi Shamir, and Leonard Adleman worked on public-key cryptography based on the work of Whitfield Diffie and Martin Hellman, who introduced it in their 1976 paper. In 1977, the group introduced the RSA scheme, which has become the most widely used asymmetric encryption scheme (Paar et al., 2010). The RSA encryption was not meant to replace symmetric block or stream encryption methods. Instead, RSA is slower than commonly used symmetric encryption algorithms like AES, DES, and Triple DES, as it requires significant mathematical computing that can decrease performance and result in slower processing times. RSA is primarily used for securely transferring and exchanging symmetric encryption secret keys, as well as for validating senders and receivers (through digital signatures and non-repudiation). Symmetric encryption is still used to encrypt bulk data.

For Wireless Sensor Networks, Assad et al. (2021) suggested an encryption system based on the chaotic map-based encryption system of the related map lattice (WSN). The method is quick and enforced, and it uses chaotic maps and genetic operations to provide improved light encryption. To counter the shortcomings of the current public key encryption systems, particularly RSA, Astrid and Diego (2021) presented a revolutionary encryption technique that converts numbers. The performance of the BB84 quantum encryption technique was also examined by the authors. To build a more reliable and secure cloud architecture, Usha and Subbulakshmi (2018) presented a novel double-layer encryption cryptographic key.

In order to address the sluggish decryption issue of RSA, Bhattacharjya et al. (2019) conducted research on a messaging system dubbed SHRSA that combines low weight and high efficiency RSA encryption with a four-layer authentication stack. The system incorporates four crucial security mechanisms, including external passwords, digital certificates, and third-party authentication. To improve security between the user and the cloud, Hussain et al. (2018) suggested a key-sharing mechanism utilizing RSA encryption technology to encrypt data in cloud computing.

In order to emphasize the contrasts between conventional and quantum computing, Mavroeidis et al. (2018) discussed quantum computing algorithms that make use of both symmetric and asymmetric encryption techniques. For wireless sensor networks (WSNs), Blasco et al. (2019) suggested a cryptography system that employs symmetric key encryption and the Diffie-Hellman key agreement protocol to exchange secret keys.

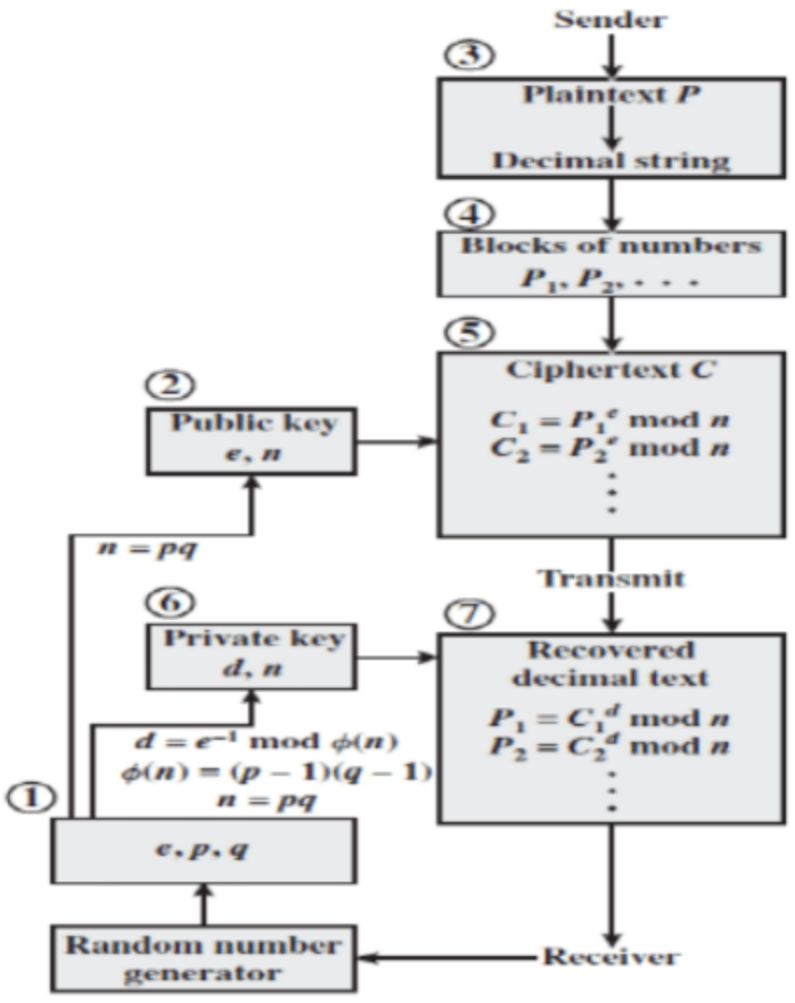

Figure 2.4: RSA Algorithm

2.4 Key Length of Encryption and Decryption Keys

Key length refers to the dimension of a cryptographic key or associated mathematical structure. It plays a significant role in establishing the achievable level of security. Yet, those who are unfamiliar with the topic may find the relationship between key length and security confusing. Not out of mathematical or security considerations, but rather because data is frequently processed and stored in 8-bit (byte), 32-bit (word), and 64-bit (block) chunks, key lengths are frequently based on powers of 2 or tiny multiples of it (Lenstra, 2010).

Newly installed symmetric encryption systems should have key lengths of at least 90 bits, according to Blaze et al. (1996), in order to assure adequate security over the following 20 years, up until 2016. Determining the minimal key length necessary to strike a balance between security and performance requirements is especially important for asymmetric encryption systems and cryptographic hash functions (Lenstra, 2010).

Singhal and Raina (2011) conducted a comparison between the AES and RC4 algorithms by measuring various performance metrics, including encryption throughput, CPU workload, memory utilization, and key size variation. The study found that RC4 was faster and more energy-efficient than AES for encryption and decryption of larger-sized data. Similarly, Soni et al. compared the AES and DES algorithms for image file encryption and decryption using MATLAB and found that AES outperformed DES in terms of encryption and decryption time. Thakur and Kumar compared the performance of different cipher algorithms (AES, DES, Blowfish) for various cipher block modes (ECB, CBC, CFB, OFB) on different file sizes ranging from 3kb to 203kb. The results showed that the Blowfish algorithm performed better for all block cipher modes, and the OFB block mode provided better performance than other block modes.

In order to compare the three encryption methods DES, Triple DES, and Blowfish, Kofahi carried out a study in 2006. The study examined the processing times of each method and discovered that the key generation times for all three were essentially the same. However there was a difference in how long the Processor took to do encryption. According to the findings, Blowfish was the quickest on the SunOS platform, followed by DES and Triple DES. The study also examined how long it took the CPU to generate the secret key, encrypt, and decrypt a 10MB file.

In their study, Naddem and Javed (2005) evaluated the encryption times of four cipher algorithms—AES, DES, 3-DES, and Blowfish—for various file sizes and cipher block modes (ECB and CFB) on two various computers—Pentium-4, 2.4 GHz, and Pentium-II, 266 MHz. According to the results, Triple DES was the slowest, followed by DES and Blowfish. Moreover, the CFB mode required longer than the ECB mode.

Wheeler and Needham (1994) proposed the 128-bit key size and 64 rounds Tiny Encryption Algorithm (TEA). In order to combat the vulnerability of related key attacks and the subpar hash function, the authors later upgraded TEA and introduced XTEA in 1997. The gate equivalent (GE) of XTEA was higher than the required range of 1000 to 3000. It became challenging to decipher encrypted information once the Advanced Encryption Standard (AES) launched a highly secure round key approach in 1998. AES is thought to be extremely safe even though it isn't categorized as a low-weight cryptography (LWC) method.

2.5 Secure Random Number Algorithm/ Generator

A random number generator is a device or software that generates unpredictable numbers within a specified range. RNGs can be classified into two types: Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs). A PRNG is a complex mathematical function that emulates randomness and is designed to be difficult to reverse engineer solely based on its output. The randomness of a PRNG is derived from a random source known as a seed. On the other hand, a TRNG relies on an input source that exhibits random properties, such as atmospheric noise or radioactive decay, to generate values. Although it is practically impossible to prove that an RNG produces genuinely random bits mathematically, rigorous statistical testing can indicate that the generated stream possesses properties consistent with a probability distribution.

IoT devices frequently produce randomness using the LPRNG, however, several experts have suggested substitute methods for gathering entropy or creating random numbers. The Yarrow RNG was first suggested by (Kesley et al., 2001) as a flexible approach that is currently used in iOS and OSX. The Fortuna PRNG, which is now used by FreeBSD, was proposed by McEvoy et al. in 2006 as a cryptographically secure way of generating random numbers. It is unclear how these algorithms might impact system performance in an IoT environment, despite the fact that they could be deployed there.

As Muller mentioned in 2014, other research has indicated that CPU jitter might be exploited as an entropy source for producing random numbers. The latter hasn't been thoroughly tested on low-power devices, while the former is solely suitable to x86 processors. (Francillon and Castelluccia, 2007) proposed a method for wireless sensor nodes that use received bit errors as a source of randomness for sensors. (Lo Re et al., 2011) recommended feeding a TRNG with the physical measurements gathered by massive wireless sensor nodes. In this research, unpredictability derived from inexpensive sensors found in mobile and IoT devices is our main area of focus.

In Mandal et al., (2016), the Warbler PRNG family for low-cost smart devices, including sensor nodes. Its foundation is a nonlinear feedback shift register by Welch-Gong (WG) with modified de Bruijn blocks. Using a 65 nm CMOS technology, two instances of Warbler with various security settings were exhibited. The two suggested scenarios are appropriate for protecting inexpensive smart devices. Nevertheless, issued a warning that the security of Warbler PRNGs may be in jeopardy because to a major differentiating attack against the whole WG family of stream ciphers that shows that nearly every member of this family is susceptible to linear attacks.

CHAPTER 3

Methods

3.1 Overview

This section describes the procedures used for implementing the configuration of lightweight using AnBx in java, compute machine, the selection of libraries for security such as cipher scheme, key length, and key size, resource utilization measurements (RAM usage, Execution time, computation time, encryption and decryption speed latency), data analysis, and visualization.

3.2 Compute Machine

A window server of 16 G RAM with ROM of 256 G was used to install Eclipse Java IDE to configure the protocol.

3.3 Lightweight Security Protocols with AnBx

AnBx is a formal protocol specification language based on the popular (informal) Alice & Bob notation. AnBx conservatively extends the AnB specification language [4] with a richer notion of the communication channel. Security Protocol is a set of procedures for encryption/decryption and verification/authentication of message agents.

In this project, based on the literature on AnBx security protocols, four security protocols were selected which includes **Fresh_From_A**, **Fresh_From_A_Secret,** **From_A_Secret**, and **From_A** protocol.

Table 3.1 describes communication modes in the selected security protocols

η	Mode	Semantics
(_,_)	Plain	Conveys no security guarantee
(A, −)	From A	a public, but authentic exchange which provides the

		receiver with a guarantee on the origin of the message
(@A, −)	fresh from A	(A, −) with the additional guarantee that the message is fresh, and may not be replayed
(−, B)	secret for B	a secret transmission, providing guarantees that only the intended receiver B will be exposed to the message payload: an intruder may become aware of the existence of an output, but not of the message contents
(A, B)	from A, secret for B	combines the guarantees of modes (A, −) and (−, B)
(@A, B)	fresh from A, secret for B	combines the guarantees of modes (@A, −) and (−, B)
$\uparrow \eta_0$	forward	conveys some of the guarantees given by η0 and signals that the message did not originate at the sender's; η0 cannot be a forward mode itself

Encryption algorithms consume a significant amount of computing resources such as CPU time, memory and computation time (Mandal, 2012).

3.4 Security Parameters

Cipher Scheme

A cipher scheme, also known as a cryptographic algorithm or encryption algorithm, is a mathematical function used in cryptography to transform plaintext (unencrypted data) into ciphertext (encrypted data) and vice versa. Cipher schemes typically involve a key or set of keys to control the encryption and decryption processes. The security of a cipher scheme is determined by its ability to resist attacks from unauthorized parties trying to decrypt the ciphertext without the key. In this project, four cipher algorithms were selected which were RSA, DES, AES, and Blowfish

Key length

key length refers to the size, usually measured in bits, of the cryptographic key used in an encryption algorithm. The length of the key determines the number of possible keys and the computational effort required to break the encryption. Longer keys are generally considered more secure, as they provide a greater number of possible keys and require more computing power to be broken. The key length is a critical parameter in determining the strength of an encryption scheme, and it should be chosen carefully to balance security and efficiency. The key lengths selected are 128 and 256 bits

Key Size

the key size refers to the length of the key used in an encryption algorithm. It determines the number of possible keys that can be used to encrypt and decrypt a message and is usually measured in bits. A longer key size generally makes it more difficult to break the encryption and recover the original message without the key. The appropriate key size depends on the specific algorithm and the desired level of security. The key sizes used are 1024 and 2048.

Secure Random Number

The secure random algorithm is a cryptographic algorithm used to generate random numbers that are highly unpredictable and cannot be easily reproduced or guessed. The algorithm uses various sources of entropy, such as keyboard timings, mouse movements, and other environmental factors, to produce truly random numbers. These numbers are commonly used in cryptography for generating encryption keys, session keys, and other security-related purposes. The use of a secure random algorithm is crucial to ensure the security and confidentiality of sensitive information in various applications, such as online banking, e-commerce, and secure communication.

In this project, three secure random algorithms were selected that is SHA1PRNG, PKCS11, and DRBG.

3.5 Resource Utilization Metrics

RAM Usage

The amount of RAM used by a cryptographic algorithm can vary depending on the specific implementation and the size of the input data. In general, cryptographic algorithms that process large amounts of data, such as encryption algorithms for large files, may require more RAM to operate efficiently. Additionally, some algorithms may have specific memory requirements or limitations that must be taken into account when implementing or using them. Proper memory management is important for ensuring the security and performance of cryptographic operations. The Ram usage was obtained by setting up Java Mission Controller (JMC) which is connected to the Eclipse IDE to monitor RAM usage per execution. This is achieved by setting a flight recording on the JMC for each execution.

Execution Time

Execution time is an important aspect to consider in cryptography algorithms as it directly affects the speed and efficiency of the encryption and decryption process. The execution time depends on various factors such as the complexity of the algorithm, the key size, and the size of the input data. Generally, symmetric key algorithms have faster execution times compared to asymmetric key algorithms due to their simpler structure. Additionally, hardware accelerators and parallel processing techniques can be used to improve execution time. However, it is important to balance the need for fast execution time with the need for strong security measures to ensure that data is protected.

Execution Time = End Time - Start Time

Computation Time

Computation time refers to the amount of time it takes to perform a particular operation or computation. In cryptography, computation time is an important factor to consider when evaluating the performance of encryption and decryption algorithms. Generally, faster computation time is desirable, as it allows for faster data processing and reduces the time required to complete cryptographic operations. However, faster computation time may also come at the cost of weaker security, as some faster algorithms may be more vulnerable to attacks. It is important to balance computation time with other factors such as key length, algorithm complexity, and memory usage to ensure the security and efficiency of cryptographic operations.

Computation Time = Number of Operations x Time per Operation

Encryption and Decryption Speed Latency

Encryption and decryption speed latency refers to the time taken to encrypt or decrypt data using cryptographic algorithms. The speed latency is affected by factors such as key size, key length, RAM usage, and computation time. It is important to consider the speed latency when choosing a cryptographic algorithm for a particular application as it can impact the overall performance of the system. Techniques such as parallel processing and hardware acceleration can be used to optimize the speed latency of cryptographic algorithms.

$$\frac{f}{1/Bs}\left(\frac{Cycles}{Block}\right) = f * Bs * \tau$$

τ is the time taken during one ENC or DE

f is the processor frequency in hertz.

Bs is the block size of the algorithm in bytes.

3.6 Experiment

The experiment for this project was designed using a combination of security parameters based on the configuration of the security protocols.

The cipher scheme, key length, key size, and security random number were combined for Fresh_From_A and Fresh_From_A_Secret security protocols. For these protocols, a total of 144 experiments were recorded each.

The cipher scheme, key length, and key size were combined for From_A and From_A_Secret security protocols. For these protocols, total 48 experiments were recorded each.