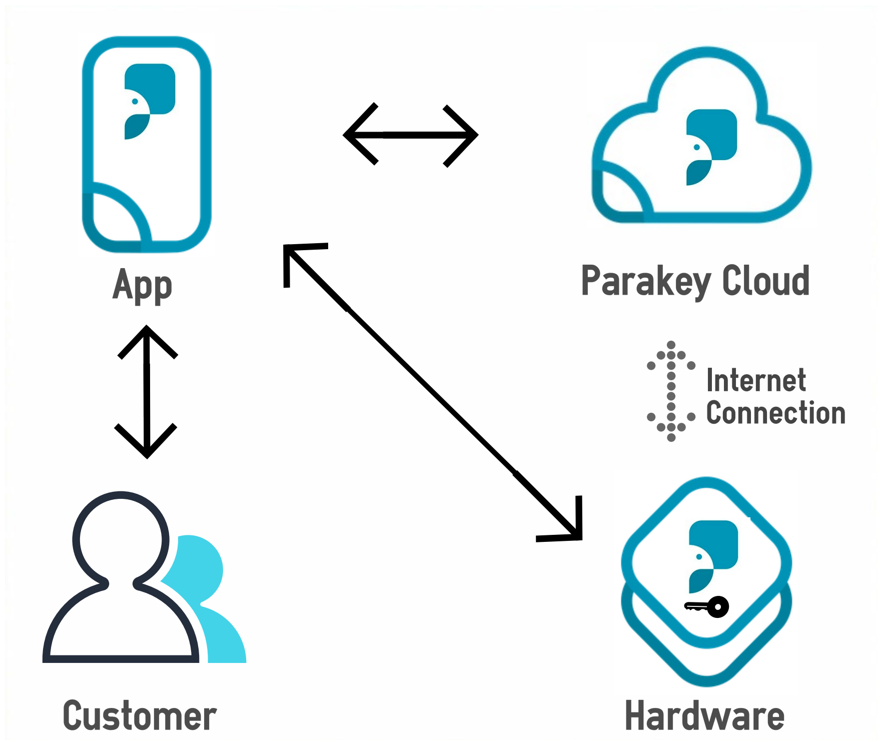
CHALMERS
UNIVERSITY OF TECHNOLOGY



A Search for a Convenient Data Encryption Algorithm

For an Internet of Things Device

Master's thesis in Systems, Control and Mechatronics

OLGEIR GUNNSTEINSSON

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis EX020/2016

A Search for a Convenient Data Encryption Algorithm

For an Internet of Things Device

OLGEIR GUNNSTEINSSON

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2016

A Search for a Convenient Data Encryption Algorithm
For an Internet of Things Device
OLGEIR GUNNSTEINSSON

Supervisors: Jonas Arvidsson and Erik Barrefors, Parakey AB
Advisor: Jessica Chani Cahuana, Chalmers
Examiner: Henk Wymeersch, Chalmers

Master's Thesis EX020/2016
Department of Automatic control, Automation and Mechatronics
Division of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover:
Connections between devices, cloud and customer using the Parakey product.

Typeset in LaTeX
Gothenburg, Sweden 2016

A Search for a Convenient Data Encryption Algorithm
For an Internet of Things Device
OLGEIR GUNNSTEINSSON
Department of Systems and Signals
Chalmers University of Technology

Abstract

Nowdays, the number of IoT (Internet of Things) devices are growing rapidly around the world with wide range of purposes. Many types of these devices collect data and depending on their purpose, some of the data can be highly sensitive for the user. This calls for security on the device which has to secure the collected data and send it to a server through the Internet. Using TLS (Transport Layer Security) is a great way to provide such a security and one part of it is the data encryption which is the main focus of this thesis. The problem is that IoT devices are often made out of microprocessors having a low computational power and performance. The main goal of this thesis is to find the most suitable data encryption algorithm for a small 32-bit CPU hardware with requirements of good speed performance, security and space complexity using a specific encryption library from ARM mbed. The four most popular symmetric encryption algorithms DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard-Rijndael) and Blowfish from the library are going to be compared in order to find the most suitable one for a mid-high end IoT device. Then the most suitable block cipher mode of operation, the ECB (Electronic Codebook), CBC (Cipher Block Chaining), OFB (Output Feedback), CFB (Cipher Feedback) or CTR (Counter) are compared to find the one that fits the task the best. Based on literature studies, it is shown that the AES-128-bits using CTR cipher block model provides the best speed performance and security on the device.

Keywords: IoT, security, data encryption, speed, DES, 3DES, AES, Blowfish, CTR.

Acknowledgements

I would like to thank my supervisors, Jonas Arvidsson, Erik Barrefors and all their coworkers from Parakey AB for all the help and support throughout the thesis. Also I want to thank my adviser Jessica Chani Cahuana and my examiner Henk Wymeersch from Chalmers (Division of Signals and Systems) for guidance and support.

Olgeir Gunnsteinsson, Gothenburg, June 2016

Contents

1
Introduction

In this chapter, the background of the IoT and cryptographic world will be shortly described. Then the benefits and problems of IoT and the root of the problem will be investigated. An introduction of the company Parakey AB will be given out and short discussion regarding OpenSource and speed performance of an IoT in order to inform the reader why that is a topic worth to discuss. Then to help the reader to understand the overall aim of the thesis and what constraints and requirements appear for a encryption algorithm on a IoT device.

1.1 Background

After a fast growing net of connections worldwide the machines and computers have gotten faster and smaller giving the opportunity to connect more devices and machines together to share information and process it. This development has triggered a trend where all things in the world should be connected to the Internet in order to provide a system which could improve the transmission of information, called the Internet of Things.

The concept of IoT was formally proposed around 2005 in a report from International Telecommunication Union (ITU) called "ITU Internet Report 2005: Internet of Things (IoT)"[1]. An IoT device, sometimes called IoT edge node, is an independent hardware that is uniquely identifiable through its embedded computing system and connected to the Internet. For example, it could be a vehicle, phone or a pressure sensor but what they all have in common is that they collect data from sensors and/or used as actuators. The growth of IoT connected devices around the world in 2015 is astonishing and keeps on growing with a forecast of 20.8 billion units by the year 2020[2].

1.2 The Benefits and problems of IoT

Trust is the new currency. Trust will define companies that win and companies that lose.

Gary Kovacs, AVG CEO

There are many benefits of having everything connected together. Who doesn't want to ease their lives with a home automation and security, getting notifications when your health is getting worse or even feed your pet when you are not at home and keep track of it? IoT will have a great impact for instance on health-care, car industry and monitoring business but unfortunately, it has a downside. When a device is connected to the Internet, it opens up connection possibilities to others who are not allowed to connect or interact with the device, so called hackers. Hackers are often compared to pirates of the Internet, trying their best to steal data from servers, devices or get full control of devices which can be devastating for people and companies.

In the beginning of the 21st century, data became more valuable for companies, people and, of course, hackers. Some data is personal but it is not always well protected and can be misused. People often think that the data collected by their phones or IoT devices does not matter. They do not mind if someone sees how many steps they took last month or gets data about their power usage. Hackers do not care about it either. They want to get to the valuable data, your bank account, social security numbers, credit card information and so on. People tend to forget that this information is stored in the company servers or, in worst cases, in the device itself. If the company has a weak or no encryption on the data or does not provide a secure connection from the device to their servers the hackers will eventually find it and abuse it. Then the customers lose their trust which can be dreadful for the company. Unfortunately, many start-up companies skip this part but security must be included from the beginning and should not be an option.

In the OWASP (Open Web Application Security Project)[3] webpage, they made a list of the IoT Attack Surface Areas where they list for instance:

- Device memory
- Device firmware
- Local data storage
- Update mechanism
- Authentication/authorization
- Transport encryption

Only *Transport encryption*, *local data storage* and *update mechanism* will be one of the main security concern discussed in the theory part where the are affected by secure communication. From the same security project, a list of top IoT vulnerabilities were made. There they mentioned the "unencrypted services" as one of the vulnerabilities, which will be the main IoT vulnerability focus in this thesis. Many other critical vulnerabilities must also be considered, for instance, "weak password", "update sent without encryption", "no manual update mechanism" and others that do not fit in the scope of this work.
Implementing security in a normal PC with unlimited power usage, a lot of processing power and CPU speed is normally not a problem. When dealing with IoT devices, it gets a bit more complicated. The IoT devices are typically small low-power devices with low processing and battery power. Therefore, companies face various problems today when implementing a security on IoT devices, for example following problems inspired by Nermin H.[4]:

- Devices are low cost and essentially disposable

- It is difficult to update firmware or apply a patch.
- IoT hardware is and will remain a problem
- The small size and limited processing power of many connected devices could inhibit encryption and other robust security measurement
- It is difficult to compress 25 years of knowledge into tight time-frame.

Also in [5], a list of constraints on an IoT device is shown which include:
- Constraints on the maximum code complexity (ROM/Flash),
- Constraints on the size of states and buffers (RAM),
- Constraints on the amount of computation feasible in a period of time (processing power),
- Constraints on the available power
- Constraints on user interface accessibility in deployment (ability to set keys, update software etc.).

1.3 Cryptography and security between an IoT device and a server

In order to provide security of data transmitted back and forth from an IoT device to a server, the communication has to be encrypted. Cryptography is the practice and study of techniques for secure communication and has been around for a long time. One of the cryptographic protocol is the TLS (Transport layer security) protocol where the data are encrypted with ciphers to protect them from third parties through the Internet. Many ciphers exist today with a range of possibilities and different purposes, for example the symmetric ciphers AES (Advanced encryption algorithm), DES (Data encryption algorithm), 3DES (Triple DES) and Blowfish. Those four encryption algorithms will be compared and the most convenient cipher will be chosen and hopefully implemented in future work.

1.4 Parakey product

Parakey is the next generation product for physical access control and can control access to, e.g., doors and gates which has an electromechanical locking mechanism. The product consists of three parts: a cloud service, a small hardware and a smartphone app.

The cloud service handles and distributes digital keys to users and their smartphones. The hardware acts as a bridge between a smartphone, such as an iPhone, and door lock or existing access control system. The smartphone app gathers the user's digital keys and sends a signal to the hardware which opens the chosen door.

Parakey's hardware is currently using Bluetooth to communicate with smartphones. Parakey AB plans to add Internet connection to the hardware thus making it more flexible to add functions that requires Internet connection, such as remote to open a door.

Parakey is using a STM32F0xx microcontroller in their hardware device, which will be connected to the Internet in augment to their current Offline solution via Bluetooth. The main requirements are that it has to be secure, fast and reliable. Using an STM32F0xx microcontroller sets the boundaries with computing speed and therefore the "size" of the security algorithm and execution speed. Also, Parakey will use the ARM® mbed™ TLS encryption library for implementation. That makes it easier for developers to include cryptographic and TLS capabilities in embedded products to connect securely from a so called perception layer to the application layer.

1.5 Open-Source algorithms and speed performance

The Open-Source community is a great place to learn when starting and building a product or a business. Using an Open-Source library or algorithm for encryption can save both time and money which is of great advantage for startup companies and even for stable corporations. One can have easy access to the algorithm and can be sure that many other people are using it. That means, there is a lot of documentations about it and even its own forum where everybody can share their experience and propose new ways to improve the library.

In most cases when encrypting data, a good speed performance is one of the best advantages after the security. If the encryption was overall a slow procedure, it would be more tempting for people to discard the security over the speed performance which can have a terrible impact on the privacy for people and companies. The speed depends on many things such as the library, implementation, which block cipher mode is used, structure of the encryption algorithm and what language it is written in. One of the worlds most famous cryptographer, Bruce Schneier, once said that if speed and performance is critical, the code for the algorithm itself and the most suitable block cipher mode, can be written in assembly. It is the same when choosing the right key size, increasing the keysize will increase the execution time of the encryption procedure[6]. From the beginning of the cryptography studies, there have and will always be a trade-off between security and performance.

1.6 Overall aim and constraints

The main purpose of this thesis work is to find the most convenient cipher to implement in an IoT device to establish a secure data communication with the server. This should be done by taking into account that the IoT device has narrow limits of available memory, computing power and speed. Also some of the critical security strategies for IoT devices are examined, in order to highlight the main keys in any security solution, and to draw conclusions for a successful encryption algorithm approach between the server and the (IoT) device.

When trying to compare encryption algorithms and operation modes and find which of them would suit a specific hardware the best, one should specify the hardware, platform, library usage, data package sizes and the purpose of the data encryption. With the boundaries and requirements described in sections 1.2 and 1.4, the constraints imposed on the (required) algorithm can be summarized as follows:

- Has to be a Open Source algorithm and included in the ARM mbed TLS library.
- Has to be fast.
- Has to work on 32-bit CPU with low CPU power.
- Has to be able to handle data packages smaller than 10kB.
- Has to have a low space complexity.
- Has to be secure.

With those constrains, only specific algorithms will be examined in this thesis work. The four most known encryption algorithms and Open-Source from the mbed TLS library, called DES, 3DES, AES and Blowfish, are compared in order to find the most suitable encryption algorithm for a specific hardware made by Parakey AB. Also different operation modes of the ciphers are investigated to select the fastest and most secure mode (among them).

1.7 Outline

The rest of this thesis is organized as follows. In Chapter 2, the theoretical background of this work is presented. The architecture of the IoT is briefly described and the cryptography concept is introduced. Moreover, a short introduction to symmetric and asymmetric encryption algorithms is presented, the block cipher modes are explained and few cryptanalysis methods are described. Chapter 3 introduces the comparison factors considered to select the convenient data algorithm and block cipher mode. Thereafter in Chapter 4, the potential algorithms are evaluated and compared by the factors which were introduced in chapter 3. Finally in Chapter 5, conclusions are drawn and recommendations for future work are provided.

1.8 Related work

Even though IoT is a fairly new "thing" in the Internet world, in the past 25 years the IT security controls have evolved and can be just as effective for this but it is difficult to compress those 25 years of knowledge into these devices[7]. Today, no unified classification or standards have been made for the devices in the IoT world. In [8] the performance of the AES, DES and Blowfish encryption algorithms are examined by comparing their performance using a varying key- and block-sizes and also number of round of encryption input file. In order to analyze the performance of every and each algorithm, the performance of execution time, memory requirement and throughput were computed.
Ciphers can be split in asymmetric and symmetric encryption algorithms. Studies have shown that the asymmmetric algorithms tend to be much slower than symmetric encryption methods and an old rule of thumb says that block ciphers can be around 50 times faster than asymmetric encryption[9]. That been said, this thesis

does not focus on asymmetric encryption algorithms for data encryption which are mainly used for key exchange methods.
Various larger companies like Atmel, WindRiver and Intel have conducted researches on how to protect an IoT device in the best way, how to secure communication between IoT devices and published articles and white papers regarding their results. Most of the studies and comparisons on ciphers have been made on different platforms, hardware and libraries which makes it harder to use their results to make any conclusions.

2
Theory

When looking into the problems of having an IoT edge node connected straight to the Internet, there are many things that need to be kept in mind. How the architecture of the IoT is set up, what encryption algorithms should be used and what are the potential threats for the system. The decision on what algorithms and encryption should be used depends on what the device is capable of regarding to processing power, computing speed and purposes. As mention in Section 1.4 the STM32F0XX microcontroller will be the hardware that the chosen encryption algorithm has to be deployed on using the ARM® mbed™ TLS library. In this chapter, the main theory of cryptography and the fundamental aspects of the encryption algorithms will be investigated.

2.1 ARM® mbed™ TLS library and the STM32F0 MCU

Implementing a security library from scratch can be hard, complicated and time consuming. Using a pre-build library can be a great way to get past that and spare some time which is a big deal for startup companies. In that case, the ARM® mbed™ TLS library, a successor of the PolarSSL library, is used in this work. This library can be used as a replacement for the OpenSSL library or other SSL libraries. It includes few of the most-used data encryption algorithms, that is AES, Blowfish, Triple-DES (3DES), DES and more. The library's memory footprint can get as small as 30k and averages below 110k, which is a great benefit for an IoT device with low ROM. The mbed TLS is primarily licensed under the Apache 2.0 license.

Using the ARM® mbed™ TLS library with the STM32F0XX microcontroller is a suitable match because they are made by the same company consequently compatibility errors are less likely to occur and cause security issues. The STM32F0XX microcontroller is an ARM® based 32-bit Cortex®-M0 CPU provided by STMicroelectronics. The chip has relatively high processing power and speed compared to the Arduino platform which is often based on a 8-bit microcontrollers. The STM32F0XX is a part of the STM32 family and is used all around the world in many different products for example phones, cars and measuring devices. It is suitable for cost-sensitive applications and highly competitive in traditional 8-bit and 16-bit chips in power consumption.

2.2 Architecture and layers of IoT

The basic architecture of IoT consists of three-layers, the application layer, network layer and sensing layer. This representation is widely used in the IoT studies to show the framework of a simple IoT architecture[10]. In recent studies, new layers have been added in between the layers, e.g. *middleware layers, transport layers, service layers* and *link layers* [11][12]. In order to get a clearer picture of the IoT set-up, interaction and connections, it is convenient to split the architecture in up to the five layers architecture as shown in figure 2.1 [13],[11],[14] and [12].

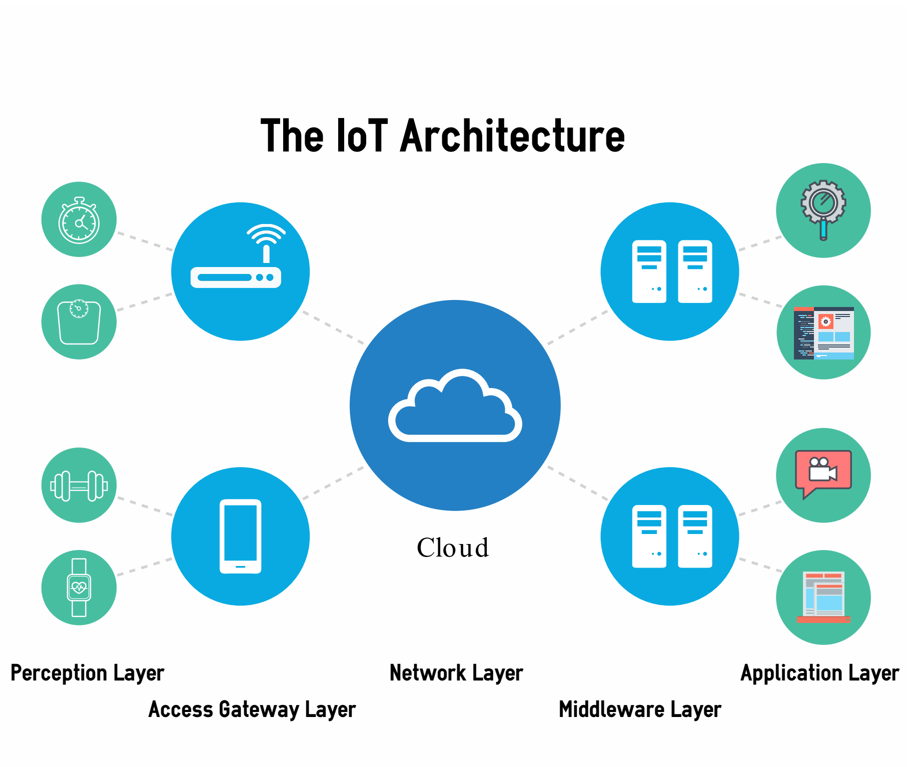


Figure 2.1: Architecture layers of IoT.

When applying full security to an IoT device, the security has to be considered on each layer. In this thesis, part of a security in the *perception layer* is investigated where the data encryption happens on the hardware itself. Although, it is feasible to shortly introduce the other layers in order to have a better and more clear overview of the structure, data flow and connections between devices.

2.2.1 Perception (Edge node) layer

This layer is also called physical layer. Here is the data collected or provided with many different hardware devices such as sensors networks, RFID tags, embedded edge processors and actuators.

2.2.2 Access gateway layer

This layer is the first stage of data handling and is important in order to publish and subscribe the services that are provided by the IoT devices. Also, if required, it performs cross platform communication.

2.2.3 Network layer

The network layer filters the received data from IoT edge nodes and other hardware devices. Then also it provides an access control from the application layer making a two-way transmission.

2.2.4 Middleware layer

The middleware layer acts as an interface between the application layer and perception layer and operates in bi-directional mode. This layer is responsible for device and information management and can take care of many functionalities such as access control, data aggregation and information service.

2.2.5 Application layer

The application layer delivers various application services through the middleware layer to the user in system based on IoT devices. This service can be used for instance in logistics, industries and healthcare.

2.3 Cryptography Algorithms and SSL/TLS

In the Internet Security Glossary RFC2828[15] the cryptography system is defined as "a set of cryptographic algorithms together with the key management processes that support the use of the algorithms in some application context." This definition gives a clear picture of the mechanism that is used to provide secure communication protocols and data encryption algorithms. When sending sensitive data between an IoT device and a server or application, the sender encrypts the unencrypted text, called plaintext, and sends the encrypted text, called ciphertext, to the receiver who (later) decrypts it.
The main requirements to be fulfilled for Parakey hardware device are then the following:

- The device has to be fast and responsive, the time between pressing "open" in the app and the door opening must not fend off customers by being too slow. That means the faster the encryption time of the encryption the better.
- Has to have tight security that involves three fundamental components of security which refer to the acronym "CIA" :
  - ***Confidentiality*** - whether data are stored or transit in a message it should be visible only to an authorized person.
  - ***Integrity*** - When a message is sent, it should not be tampered or changed on the way to its destination.

  - ***Authenticity*** - The sender of the message is who he says he is.
- The encryption algorithm must be well documented, that is, have been researched, attacked and written about. Also the implementation methods and library are more or less Open-Source.
- The encryption algorithm must be has to be suitable for STM32F0 32-bit MCU hardware which the IoT device from Parakey is based on.

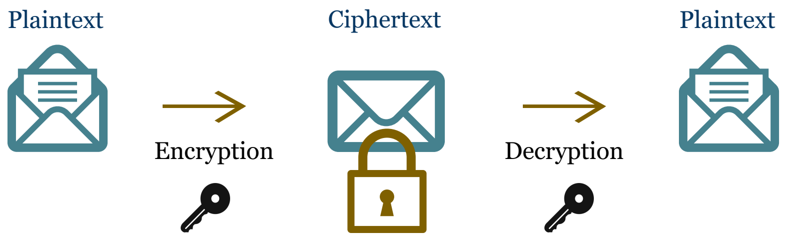


Figure 2.2: Secure communication with Symmetric Key.

2.3.1 Secure Sockets Layer(SSL)/Transport Layer Security(TLS)

In order to fulfill the "CIA" requirements, a cryptographic protocol called Transport Layer Security (TLS), or its predecessor Secure Sockets Layer, is often used. The main purpose of TLS is to provide secure communication between two computer applications that includes privacy, data integrity and also authenticity.
Having a secure TLS connection means that:

- The connection is using symmetric cryptography, which is a requirement, in order to encrypt the transmitted data which provides the privacy of the message. For the thesis, this part of the TLS protocol is the main focus, that is, which symmetric cryptography algorithm is the most suitable to provide privacy or confidentiality of the message.
- Public-key cryptography is used to authenticate the identity of the message sender. This procedure is generally required for the server side but the authentication can be made optional depending on the importance of the message.
- The transmitted message includes an integrity check meaning that the connection is reliable. The integrity check is made by using message authentication code (MAC) using a hash function which prevents alteration of the data and some undetected loss of the message during its transmission caused for example by "man-in-the-middle".

A typical TLS connection process starts with a procedure called "handshake" between the client and server and will be described shortly as following:

1. The client, often a browser, sends a **ClientHello** message to a specific server, including a highest TLS protocol version it supports, cipher settings, a random number and a list of suggested compression methods.
2. The server responds with the message **ServerHello** which contains the chosen protocol version which always has to be the highest version that both the client

and server can provide. The message also contains the servers certificate with the servers public encryption key and other information that the client needs to communicate with the server over TLS.

3. Next, the client checks the certificate from the server and encrypts a random number pre-master-secret used to generate the symmetric key. That will be used for the secure communication in the end, with the servers public key and sends it to the server with the clients certificate. This is called the key exchange.
4. The client sends **ChangeCipherSpec** to the server to indicate that it has now sent the pre-master-secret, generated the symmetric key and changed the cipher spec to using symmetric encryption. Then the client sends **ClientFinished** message, telling the server that the client is done with the key exchange and the handshake and ready to begin secure communications using the generated symmetric key.
5. Now the server sends back **ChangeCipherSpec** to the client indicating the cipher change. Also it sends back **ServerFinished** message which is encrypted with the symmetric key. This message tells the client that a successful handshake with the server has been made, making the secure communication, using the chosen symmetric cipher which could be for example AES, DES, 3DES or Blowfish.

The focus in this thesis will be on the privacy or confidentiality of a message, that is, what is the best and most convenient data encryption algorithm for the Parakey's IoT device using the ARM® mbed™ TLS library in order to have a secure communication between a IoT device and a server.

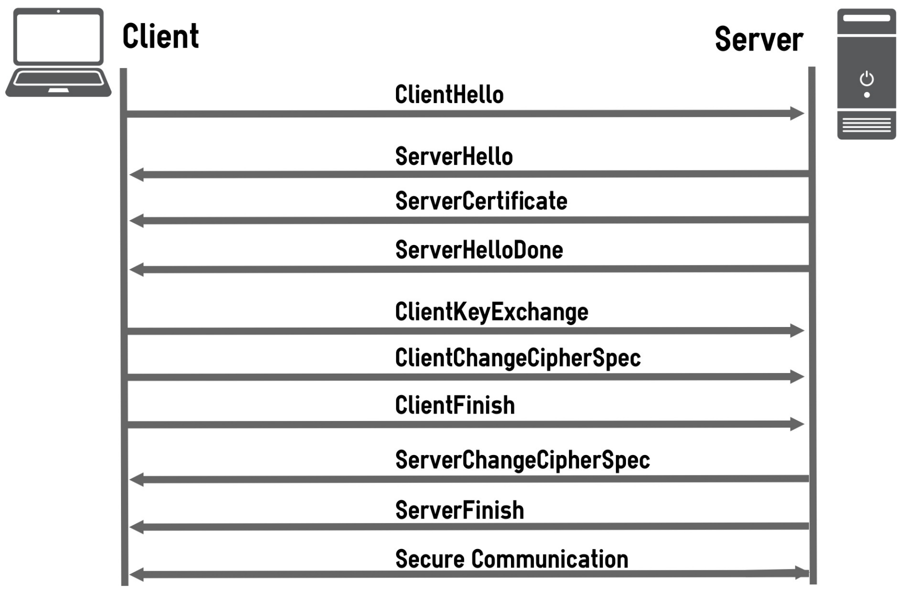


Figure 2.3: TLS handshake procedure.

2.3.2 Symmetric encryption

In cryptography, there are two main techniques to encrypt/decrypt data and other information. They are called symmetric and asymmetric cryptosystems. For asymmetric encryption the keys are distinct where the encryption key is public and accessible by everyone and can be disseminated widely but the decryption key to decrypt messages is kept secret, often in a server [16]. This method is called Public-key cryptography and as mention in the Introduction Chapter, it will not be investigated further as a candidate for a suitable data encryption algorithm. In figure 2.4 the classification of encryption algorithms can be examined.

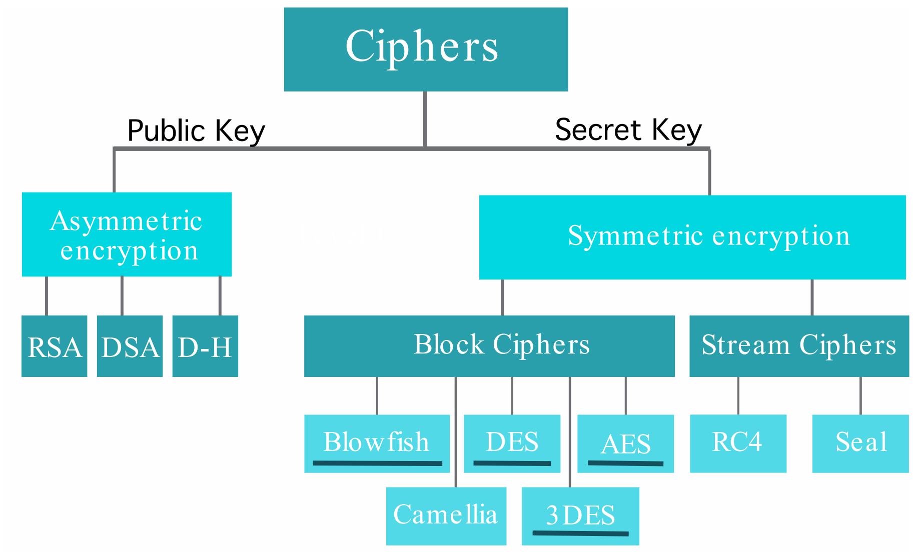


Figure 2.4: Classification of encryption algorithms. The underlined algorithms indicates those that will be examined in the thesis.

The symmetric encryption is told to be the most known encryption technique and the oldest one, also called secret key encryption. The method is based on the idea that the same secret key is used for both the sender and reciever to encrypt or decrypt, i.e., the encryption key is equal to the decryption key[16]. As mentioned in section 2.3.1, the secret key is pre-agreed by the sender and receiver in the "handshake" process.
Symmetric encryption algorithm can be divided in two types of ciphers, a block cipher and stream cipher.

- **Block cipher:** Block ciphers operate on so called blocks which are fixed-length groups of bits. Each block is usually 64, 128 or 256-bits in length. Having a 128-bit block cipher will be able to process 128-bits of plaintext and give out 128-bits ciphertext. If the plaintext is shorter than 128-bit scheme called padding will modify it in order to get the length of it equal to a multiple of the given block size. Typical block ciphers are the DES, Blowfish and AES algorithms [17] [9].
- **Stream cipher:** In stream ciphers the encryption is done on each bit of a plaintext at a time using a random digit stream of a pseudorandom bits (keystream)[17]. In order to have the stream cipher implementation secure,

the generator for the pseudorandom should be unpredictable. For example, the symmetric key algorithm RC4 is a popular stream cipher[18].

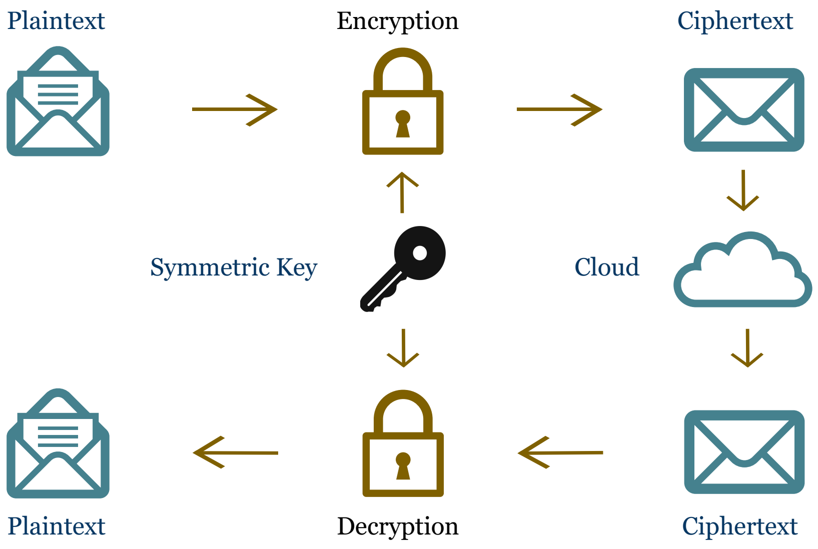


Figure 2.5: Secure communication with Symmetric Key.

The function of the four symmetric algorithms that will be evaluated in chapter four are described in following subsections:

2.3.2.1 Data encryption standard

Data encryption standard (DES) is a block cipher which was the predominant symmetric key algorithm for data encryption until its successor AES was introduced. DES was created at IBM in the early 1970s based on a design of Horst Feistel and is the most studied cipher in the world. It has 64 bits block size and key size of 56 bits plus 8 parity bits. For encrypting DES uses 16 rounds of Feistel structure, as can be seen in figure 2.6, where it takes a 64-bits plaintext in the first round and splits it in two 32-bits blocks called R_0 and L_0. Then it puts R_0 into a Feistel function and XORs the output of it to L_0 which provides R_1 while L_1 is made straight out of R_0. Then this procedure is repeated 16 times, giving out a 64-bits ciphertext.

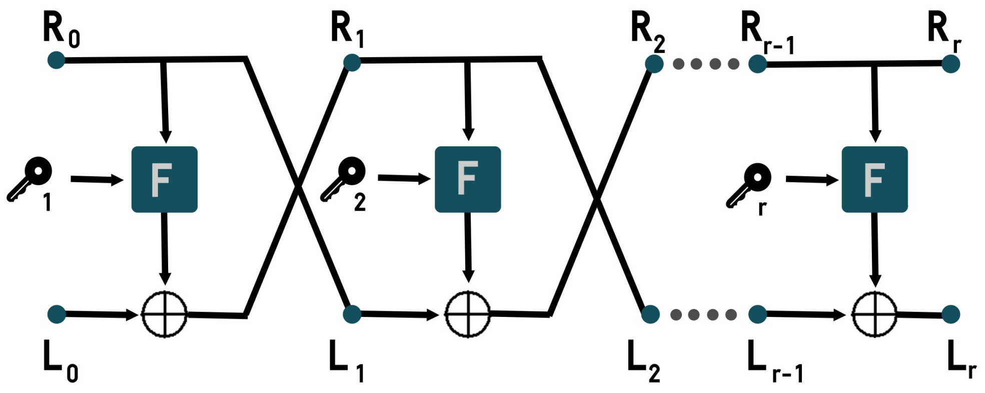


Figure 2.6: Feistel structure.

The Feistel function is often called the "heart" of the cipher where the actual encryption happens. The input is a 32-bits of text which is expanded to 48-bits and XOR-ed with 48-bits sub-key. The product is then divided into eight parts including 6-bits each which are imported to a S-box (a special made look up table), giving out 4-bits. All the bits are then combined making a 32-bits text where permutation is made before it goes out as an output.

2.3.2.2 Triple-DES (3DES)

The Triple DES is the DES cipher algorithm but applied three times to each data block. Therefore, it has the same block size of 64-bits but the number of rounds are 48 using the Feistel network structure. When the DES was made, the key size of 56 bits where enough security but with faster computer processors and power the key size had to be increased to ensure the security and in the year of 1998 the 3DES algorithm was published. The key size of the 3DES depends on which keying option is used. The key sizes are then 168, 112 or 56 bits and the keying option 1, 2, 3 respectively. The four common proposals for 3DES are showed in table 2.1 and described in figure 2.7.

	Shorthand	Description
2-key ENC-DEC-ENC	EDE_2	$c = ENC_{k_1}(DEC_{k_2}(ENC_{k_1}(m)))$
2-key ENC-ENC-ENC	EDE_2	$c = ENC_{k_1}(ENC_{k_2}(ENC_{k_1}(m)))$
3-key ENC-DEC-ENC	EDE_3	$c = ENC_{k_3}(DEC_{k_2}(ENC_{k_1}(m)))$
3-key ENC-ENC-ENC	EDE_3	$c = ENC_{k_3}(ENC_{k_2}(ENC_{k_1}(m)))$

Table 2.1: Table of the common proposals for 3DES encryption, inspired by [9].

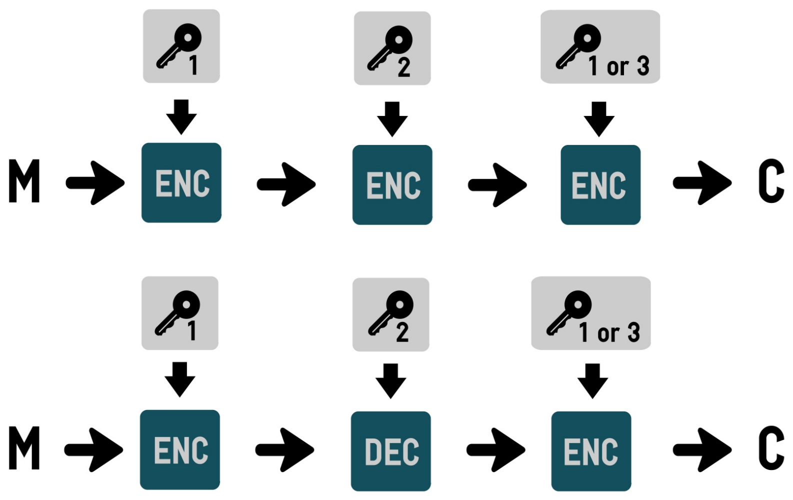


Figure 2.7: Figure of the common proposals for 3DES encryption, inspired by [9].

One of the most common proposal deployed is the EDE_2 which might be a surprise but the fact is that it allows backward compatibility with single encryption[9]. Although the NIST standard specifies using the form three-key EDE (EDE_3). One should notice the when IBM designed the DES, it was designed for efficient hardware implementation but unfortunately it is really slow in software which leads to the fact that the 3DES is about three times slower in software.

2.3.2.3 Advanced Encryption Standard (Rijndael)

Advanced Encryption Standard (AES) is the successor to DES and is based on the Rijndael (Dutch pronunciation is similar to Rain-Doll) cipher. Rijndael was introduced in 2000 and was chosen by NIST and other cryptographers in 2001 to become the AES cipher. It is a symmetric key block cipher. AES uses a 128-bits block size for encryption and the key size can differ from 128,192 or 256-bit size using 10,12 or 14 cycles of repetition (rounds) respectively to the key size.

In each round, the AES uses the four following operations[9]:

- **SubBytes:** Each byte of the array is transformed using a nonlinear substitution box called the AES S-Box. The S-Box in the AES has been carefully constructed and the cipher uses only one S-Box throughout the encryption.
- **ShiftRows:** Is a transposition step which ensures that the last three rows of the array are shifted by a different number of byte positions.
- **MixColumns:** Mixes each column in the array to create even more diffusion.
- **Addkey:** Using bitwise XOR, each byte of the array is mixed with a byte of a sub-key material, also called round-key. The sub-key is made by "key expansion" and is derived from the main cipher key using a Rijndael key-schedule.

The overall structure of the cipher begins with a key expansion before going in to the pre-whitening which only includes *AddKey*. Then the rounds performing *SubBytes*, *ShiftRows*, *MixColumns* and *AddKey* will loop for $n-1$ rounds where n is the numbers of rounds. The n-th round or final round does not include *MixColumns* and produce the output ciphertext. This high-level description of the algorithm, see figure 2.8, can give a better overview how the cipher works.

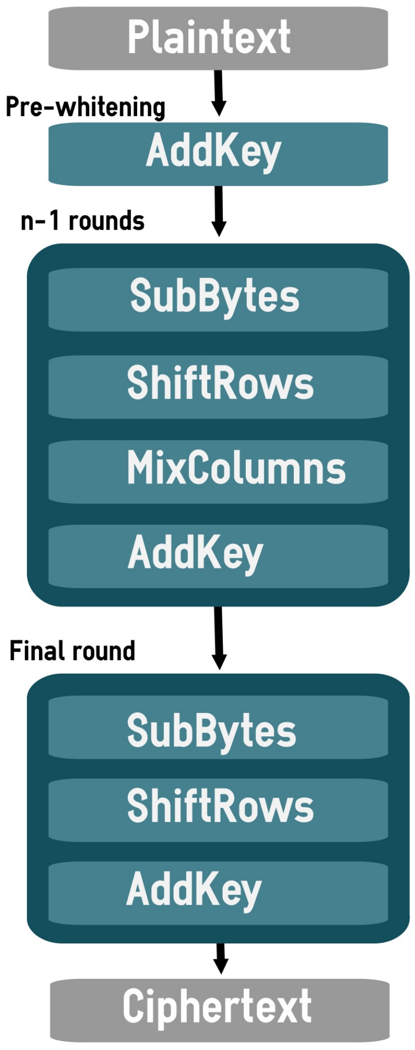


Figure 2.8: AES high-level structure.

When using longer keys, it becomes much harder to break the cipher and in addition but can slow down the performance of the cipher. For a comparison, AES has double the block size of DES and having a much bigger block size makes the cipher more resilient against information leak which can occur by using repetitive blocks.[26] Many CPUs and MCUs include hardware support for AES which can boost up the speed performance and make it a really fast cipher.

2.3.2.4 Blowfish

Introduced in 1993, the blowfish algorithm was designed to replace DES and is known for its overall effectiveness, speed and being one of the flexible encryption methods available [19]. The Blowfish encryption algorithm has the default key length of 128 bits and operates on 64-bit bit blocks of plaintext. It has a variable key lengths which can range from 32 up to 448 bits but key size is never used below the default 128-bit key size. It uses a modified Feistel structure where both sides are modified in each round, which is different from the DES Feistel structure where only other one is modified each round. First, the sub-keys are generated using Blowfish's key schedule which initializes so called P-array and S-boxes. In order to prevent that the initialization values that are used will not contain any obvious patterns, they are derived from the hexadecimal digits of pi. After the conversion of a key of at most 448 bits, the sub-keys arrays, noted as K, will be totaling 4168 bytes but the method will not be described in details in this thesis. A plaintext is split into two 32-bits L_0 and R_0, then in each round (Blowfish uses 16 rounds) four actions are performed:

- The left half (L_0 in the first round) is XORed with the first sub-key.
- The output is used as an input for the Blowfish F-function.
- The output of the F-function is XORed with the right half (R_0 in the first round).
- Lastly the XORed bytes are swapped and a new round begins.

After the last round, both left and right half outputs are swapped back and XORed with the last sub-keys K_{17} and K_{18}. A better explanation of the Blowfish's Feistel structure can be seen in figure 2.9.

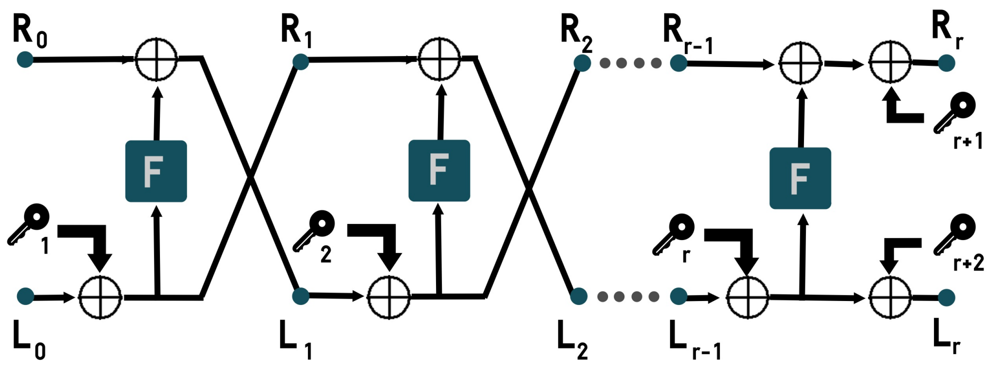


Figure 2.9: Blowfish's Feistel structure.

The F-function takes 32-bits as an input, splits them into four 8-bits blocks and feeds each block into a 8-to-32 S-Box. Then the outputs from S-Box one and two are added together using addition mod 2^{32} and the output from that XORed with S-Box three. Lastly the output from earlier XOR is added together using addition mod 2^{32} again and the product is the output from the F-function as can be seen from figure 2.10.

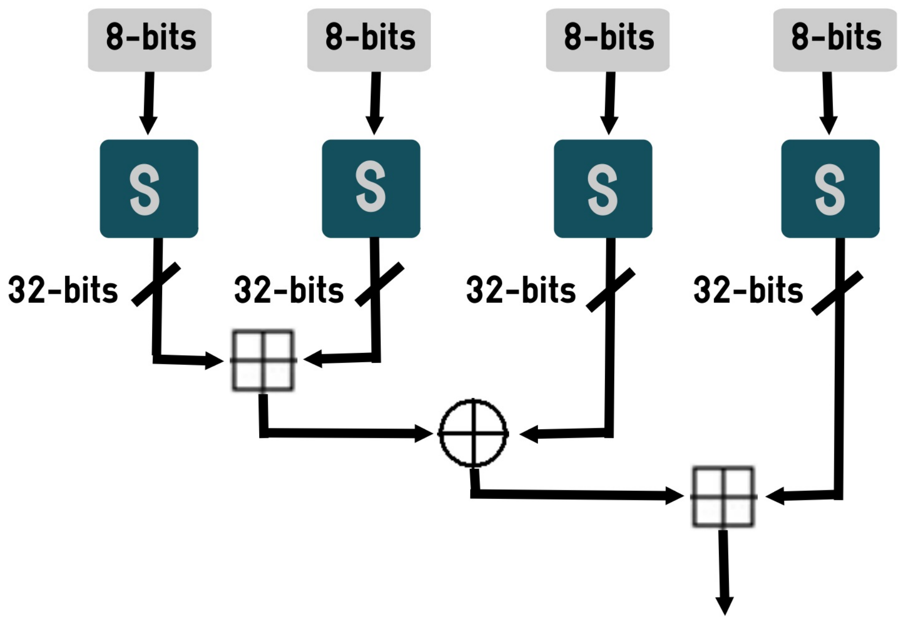


Figure 2.10: Blowfish's F-function.

The Blowfish algorithm can be optimized in hardware applications though it's mostly used in software applications.

2.3.3 Block cipher modes

A block cipher which has two or more equal plaintext blocks to process, will produce the same ciphertext blocks for each of plaintext blocks. In that case, ciphertext might reveal patterns in the plaintext which is a potential weakness for the cipher. In order to counter this, the block ciphers have different modes of operation that can by used. The National Institute of Standards and Technology (NIST) have five recommended modes called Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR).
When the modes are described, following characteristics are investigated in order to have a better understanding of the modes:

- *Padding schemes:* The block ciphers use fixed size blocks but messages are not always in the same size as the block size. Padding is needed to fill up to the block and can be done by several padding schemes for example adding extra null bytes to the plaintext. Padding can produce problems were the original length of a plaintext can be recovered, using for instance side-channel-attack or padding-oracle-attack.

- *Hides repeating plaintext blocks:* When encrypting a plaintext, it will produce a ciphertext. If two identical plaintext are encrypted with the same key, it could produce the same ciphertext twice and can causes security issues.
- *Proof of security:* Has the operation mode been proofed as secure? Are there any knowable cracks in the method itself and what kind of security services can it provide, that is, confidentiality, integrity and/or authentication.
- *Speed and efficiency:*
  - Parallelization is examined, that is, the feature to process different blocks at the same time which could increase speed of the encryption of a plaintext.The capability for parallel processing is also called random read access.
  - Is there any capability for preprocessing available for the certain method which could also be used in some environments to increase speed performance. In the preprocess the mode can compute the "pad" Y generated from the encryption block in "spare cycles" even before one knows the plaintext P. When P becomes known, it need only be XOR'ed with the already-computed pad Y.[20]

2.3.3.1 Electronic Codebook (ECB)

ECB is the simplest and native mode of the block cipher modes where only one message block is encrypted completely independently of the encryptions of other blocks. The data is divided into plaintext blocks where the block size depends on the encryption algorithm and each block is encrypted separately from the others. For ECB encryption and same for decryption, the multiple forward and inverse cipher functions can be processed in parallel. That feature makes ECB on of the fastest and easiest cipher block mode to implement and is one of the most common mode of DES algorithm.

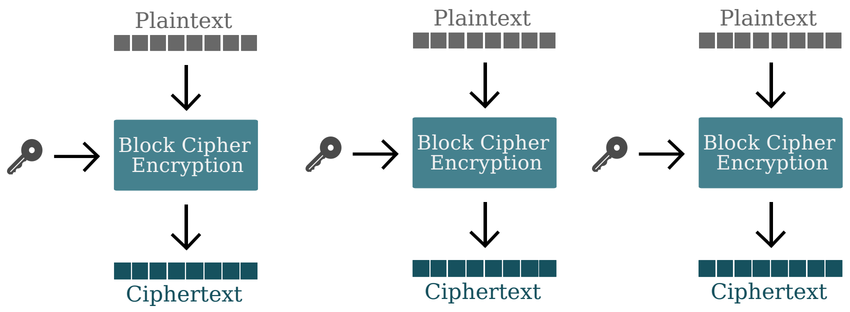


Figure 2.11: Electronic Codebook (ECB).

The ECB block cipher mode require their input to be an exact multiple of the block size at each time. For instance, if the input is 14-bits and the block is only 8-bit a padding is needed to fill up the last 2 bits in the second block. This mode has some fundamental and widely known problems. First among these is when the plaintext blocks being encrypted are identical the encryption will produce identical ciphertext using the same key. As stated before, for ECB, each block is encrypted

independently, giving it the feature that encryption and decryption is parallelizable procedure. For the same reason, ECB has random read access which is a good factor of getting better speed performance. In speed test done by Wei[21], the speed performance of key setup and IV(initialization vector) for AES in ECB mode is the best compered to the other modes, as can be seen in table A.1 in Appendix 1. That is related to the fact that ECB does not use IV. For messages longer than one block or keys are reused for more than a single block message, the ECB block cipher mode is not recommended were it does not hide data patterns. Hence, if plaintexts are completely uniformly random and no padding is needed, the ECB will be able to provide confidentiality.[20]
Specifically, when a block cipher can not hide repeating plaintext blocks, it can allow an attacker to:

- distinguish whether two ECB encrypted messages are identical;
- distinguish whether two ECB encrypted messages share a common prefix;
- distinguish if and where a single ECB encrypted message contains repetitive data, such as long runs of spaces or null bytes, repeated header fields or coincidentally repeated phrases in text.

2.3.3.2 Cipher Block Chaining (CBC)

Cipher block chaining is an IV-based encryption scheme. The IV (Initialization vector) is a fixed-size input to a CBC mode. How the ciphertext is generated is described in the following equation and an explanation figure of the mode is shown in figure 2.12. There the plaintext m_1 is first XORed to the IV before it is encrypted with the key producing the ciphertext c_1. Thereafter the ciphertext c_1 produced by the first round of CBC is XORed with the next plaintext m_2 block and so on, thus making all the blocks dependent on all the previous blocks [9].

$$c_1 = ENC_k(m_1 \oplus IV) \quad and \quad c_i = ENC_k(m_i \oplus c_{i-1}) \quad for \quad 2 \leq i \leq n,$$

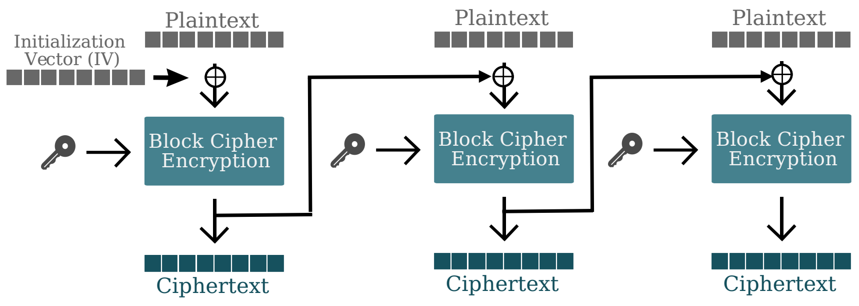


Figure 2.12: Cipher Block Chaining (CBC).

As can be seen in figure 2.12, the CBC is chaining dependent were a ciphertext block in the chain is dependent on the input plaintext blocks and all preceding plaintext blocks. The CBC only allows random access to ciphertext, not plaintext, giving it the capability to have only the decryption parallelizable. The CBC uses IV for the first block but the IV does not need be set secret, but its integrity should be protected. Changing IV or the first plaintext block results in different ciphertext in other words, same plaintext will not produce identical ciphertexts using different IV. CBC ciphers can be attacked with a so called Lucky Thirteen attack if the library is not written carefully to eliminate timing side channels[22]. Although, CBC is said being secure and providing confidentiality if the user employs a random IV but if one merely uses a nonce (number only used once) IV, the mode can not been considered secure[20]. Unfortunately, preprocessing is not doable where every encryption process depends on the one before.

2.3.3.3 Cipher Feedback (CFB)

The CFB is primarily a mode to derive some characteristics of a stream cipher from a block cipher. In particular, the decryption using CFB is almost identical to CBC encryption but only performed in reverse. In CFB mode, in each call to the block cipher a t bits are encrypted where $1 \leq t \leq b$ and b is the b-bit block size. For simplification, MSB_t is the most significant bit and LSB_t will be the least significant bit of t respectively of a block. Assuming that each ciphertext and message block is t bits of length the encryption of a block m_i is given by following equation:

$$c_i = m_i \oplus MSB_t(ENC_k(x_i)) \quad and \quad x_{i+1} = LSB_{b-t}(x_i)||c_i \quad for \quad 1 \leq i \leq n, \tag{2.1}$$

where x_1 is a chosen initial value IV.[9]

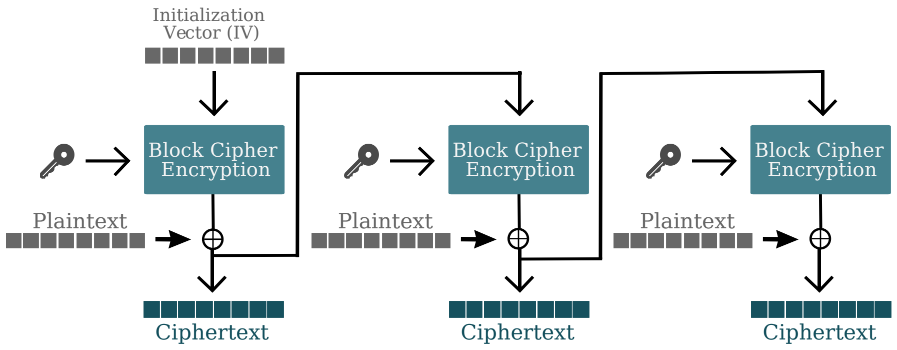


Figure 2.13: Cipher Feedback (CFB).

CFB mode of operation of a block cipher effectively turns the block cipher into a stream cipher. The mode is similar to CBC, that is, does not have identical messages and the chaining is the same. CFB allows random access to ciphertext which only gives the capability for parallel processing in decryption. It uses IV which does not need to be secret were it is XOR'ed with the plaintext. Preprocessing is not an option for CFB were the plaintext has to be XORed with the keystream before it is used as an input for the next message block.

2.3.3.4 Output Feedback (OFB)

The OFB is similar to the CFB mode. In order to get the ciphertext, it generates keystream blocks which are XORed with the plaintext blocks. But unlike the CFB mode, the encryption in OFB does not depend on previous ciphertexts. Thus, each OFB encryption operation is dependent on all the previous ones which leads to the fact that it can not be performed in parallel. For each iteration of the encryption function, the OFB mode provides s bits of keystream for all the block messages except the last one where $s = b$. Depending on the size of the input block, $s \leq b$ bits might be used in the last block of message[9]. In following equation, the OFB encryption of block m_i can be seen as:

$$c_i = m_i \oplus MSB_s(ENC_k(x_i)) \quad and \quad x_{i+1} = ENC_k(x_i)||c_i \quad for \quad 1 \leq i \leq n, \tag{2.2}$$

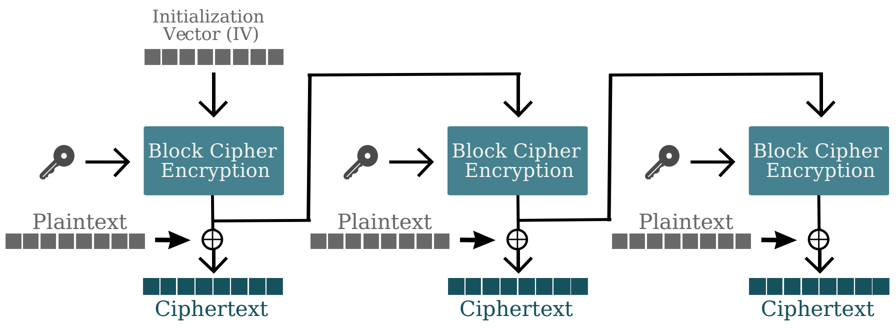


Figure 2.14: Output Feedback (OFB).

This mode also turns the block cipher into a stream cipher. It is similar to CBC and CFB where it does not produce identical ciphertexts from identical plaintexts due to the chaining properties and therefore it hides repeating plaintext blocks. For OFB, there is no possibility for parallel processing nor random access for either encryption or decryption. Instead it has a great advantage that preprocessing is possible where it keeps encrypting and decrypting previous output blocks. IV is used for the first encryption block but does need not be secret, but should be changed if a previously used key is to be used again. OFB does only provide a confidentiality and is considered a secure mode.

2.3.3.5 Counter (CTR)

In the same manner as CFB and OFB, the Counter mode makes stream cipher out of a block cipher. By encrypting a successive values of a so called "counter", the next keystream block can be generated. This "counter" can be a function which produces a sequence of a number and is guaranteed not to repeat for a long time. The counter is combined with a nonce (arbitrary number that may only be used once) and encrypted with a key. Then the keystream is XORed with the message to produce the ciphertext. In its most general form, the encryption of block m_i is given by

$$c_i = m_i \oplus MSB_s(ENC_k(x_i)) \quad and \quad x_{i+1} = Increment(x_i) \quad for \quad 1 \leq i \leq n, \tag{2.3}$$

The CTR does not depend on the values of previous encryption operations like OFB which gives it a parallelizability which makes it faster than other confidentiality modes and in some settings it can be made much faster. [9]

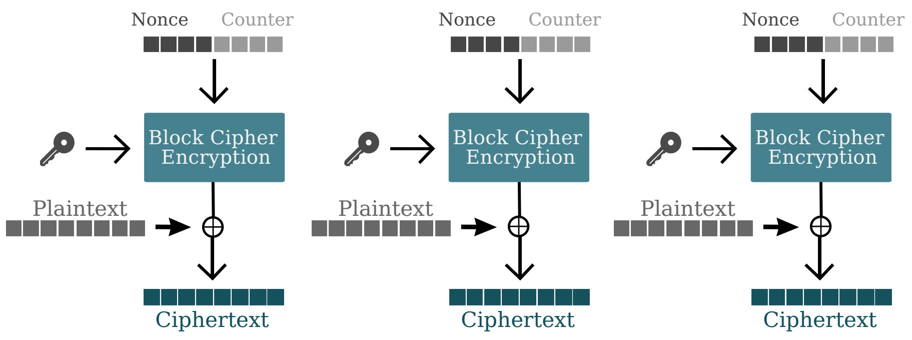


Figure 2.15: Counter (CTR).

The mode is the simplest and most elegant of the confidentiality-only schemes that have been examined above, ECB, CBC, OFB and CFB. It is not using the chaining method like CBC, CFB and OFB, instead similar to ECB mode, each encryption block is independent from another. That gives CTR the possibilities to preprocess the encrypting/decrypting counter and IV where encrypted counter is sufficient to encrypt or decrypt. Also it allows random access and makes both encryption and decryption to be parallelizable. Unlike ECB it can hide repeating plaintext blocks by changing nonce results in different ciphertext but the nonce should be random and should be changed if a previously used key is to be used again. Given that CTR is a pseudorandom permutation, which is the standard cryptographic assumption about a blockcipher's security, proves that the security of CTR-mode encryption is enough.[20]

2.4 Cryptanalysis

Cryptanalysis deals with the attacks on cryptosystems and is used to gain access to an encrypted message without even not knowing the cryptographic key[16]. In this thesis section, some of the most popular symmetric key algorithms attacks, that concerns the investigated algorithms in this thesis, will be introduced.

2.4.1 Brute force attack

The brute force attacks also known as exhaustive search is a cryptonanalytic attack which can be used on any encrypted data. In a brute force attack, the attacker tries to decrypt a ciphertext by trying all different key combinations from the key space. Although it is consider the weakest attack, it can be very effective for cryptosystems with two key sizes, using the Meet-in-the-middle attack.[16].

2.4.2 Meet-in-the-middle attack

Meet-in-the-middle is a known attack that can reduce the number of rounds using the brute force attack on order to decrypt a message that has been encrypted by more than one key.
The attacker applies brute force attack to both the ciphertext and plaintext of the cipher. Then encrypts the plaintext and simultaneously decrypts a ciphertext with various keys to achieve an intermediate ciphertext. If there is a match in a block of intermediate ciphertext and plaintext, it is highly probable that the key used to encrypt the plaintext and the key used to decrypt the ciphertext are the two encryption keys used for the block cipher.

2.4.3 Side-channel attack

In cryptanalysis, side-channel attacks do not target weaknesses in the encryption algorithms themselves. The side-channel attack exploits weaknesses in the implementation of the algorithm in software and retrieve information from the encryption device, in this case the Parakey IoT device, which is neither a plaintext or the ciphertext. That is why software developers must be aware of the potential threat of a side-channel attack form the beginning[24].

2.4.4 Differential cryptanalysis

Differential cryptanalysis was invented in 1990 by the Israeli researchers Adi Shamir and Eli Biham. Differential cryptanalysis is suited to the block ciphers with a weak round function (Feistel-network ciphers) like DES for instance. The main function of the method is to study how differences in plaintext will make changes in the ciphertext in block ciphers. It tries to discover where the cipher displays a non-random behavior and from that use that weakness to retrieve the secret key. Differential cryptanalysis method is fairly complicated and will not be studied into details in this thesis.

2.4.5 Padding-oracle-attack

Padding-oracle-attack is an attack which is often aimed to the specific block cipher mode CBC but can be used on other ciphers which needs padding, for instance RSA cipher. It is hard to describe this attack in few words but the main idea is to use an oracle, usually a server, which you send a ciphertext to and ask if it has a proper padding. Depending on the answer you get back, error or success, you can modify the IV in order to get the original plaintext. This attack requires at least two-block ciphertext, one IV and one block of ciphertext. The method will not be described in details but the main observation for the padding-oracle-attack is that if the attacker modifies the ith byte of the IV, it will cause a predictable change in the ith byte in the encoded data.

3 Method

The main goal is to investigate and figure out which encryption algorithm and block cipher mode is the best for a hardware device made by Parakey. In this chapter, results and conclusions from other studies are examined in order to find the encryption algorithm that fits the best to Parakey's microcontroller. The examined aspects will be introduced and explained in order to help the reader to understand how each encryption algorithm will be evaluated and what operation mode will be selected.

3.1 Examined aspects

The encryption algorithms are examined with respect to security, execution and throughput speed and space complexity:

- ***Security:*** When security in encryption algorithms is discussed, the security of a cipher is often measured in *bits of security.* Having an encryption algorithm with 256-bits of security indicates that the cipher uses an encryption key which is 256-bits long. According to RFC 7525, The Recommendations for Secure Use of Transport Layer Security (TLS), says that cipher suites that offers less than 112 bits of security **must not** negotiate with the implementation. Then, cipher suites that use algorithms offering less than 128 bits of security **should not** negotiate with the implementation [25]. This is critical for algorithms in order to be called a secure encryption algorithm for TLS 1.2 and will be examined in this section.
  - *Possible key generation:* What is the lowest and highest key bits security that the algorithm can provide. As mention above, an encryption algorithm has to obey the RFC 7525 recommendations in order to be called a secure algorithm.
  - *Time taken for a brute force attack:* How long time does it take to find the key searching all possible combinations using a fast guessing tool as brute-force-attack. For example, if an algorithm offers a more than 128-bits of security it will take a computer with a guessing rate of 10^{12} keys/sec[26], millions of years to find all possible keys and is considered secure with today's technology.
  - *Most effective attack:* What kind of an attack is the most effective for attackers to use? Here the brute-force-attack will not be taken into account where it has the same impact on all the algorithms. Often an algorithm

    has a certain "weak spot" which can allow a hacker to use a certain attack made for a special task. In most cases, the weak spot does not exist in the algorithm itself, rather it lies in its implementation of it. Most often the attack does not break the code, it makes it weaker, resulting a theoretically possibility to break the code but is unfeasible with current technology.

  - *Has it been cracked?:* Has there been any successful attack on the algorithm? If that is the case, the algorithm is considered as not secured.

- ***Execution and throughput speed:***
  - *Encryption speed:* How relatively fast is the encryption execution compared to the other ones. Speed often called throughput is displayed in data length per second, i.e., Megabyte/seconds = Throughput. The comparison of encryption speed of different algorithms has to be done on the same platform, hardware and computer language. Comparing AES in C and Blowfish in Java or computers with different CPU would not make sense at all. Unfortunately, an algorithm speed test was not doable for this thesis on specific hardware using 32-bit MCU, instead, results from other research papers on a speed comparison between the algorithms are examined and educated conclusion from them will be made. Different researches will be examined with respect to encryption speed.
  - *Key setup and IV:* How relatively fast is the key setup and IV initialization compared to the other ones.
- ***Space complexity:*** When a plaintext gets encrypted, its size can scale up depending on the encryption algorithm. In this section, each encryption algorithm will be examined and compared how the plaintext scale after being encrypted using the specific algorithm.
- ***Choosing the right key size:*** Some algorithms tend to have slower key setup encryption for larger key size while other have constant speeds and key setup for all keys. Even in some cases, algorithms have slower key setup for smaller keys but that is really rare.[6] Also the key size can affect the speed performance of the encryption procedure itself where more computation is involved. As stated before, the security level of an algorithm depends directly of the key size, lower key size, lower security but better speed performance.

3.2 Comparison of Block Cipher Mode of Operation

In Chapter 4, the following aspects of the block cipher modes, described in Section 2.3.3 will be compared in the Summary. In order to find the most suitable block cipher operation mode it should:

- ***Not need padding scheme.***
- ***Be able to hide repeating plaintext blocks.***

- ***Be proven as a secure mode and provide at least confidentiality.***
- ***Be able to provide parallelization for encryption and/or decryption.***
- ***Be able to provide random access for writing and/or reading data.***
- ***Have the capability for preprocessing.***

  Having the ability to provide parallelization, random access and the capability for preprocessing, will increase the speed of the operation mode of the cipher. The security will increase if the operation mode is able to hide repeating plaintext blocks, does not rely on a padding scheme and has no history of being cracked.

4 Performance results

4.1 Security

In this section, the security aspects will be evaluated on each encryption algorithm, as described in Chapter 3.

4.1.1 DES

DES does not fulfill the RFC requirement [25] with only 56-bits keys giving a 2^{56} possible keys. This relatively small key size makes it vulnerable to attacks and has been broken many times for instance with SciEngines RIVYERA FPGA cluster computer, having utilized 128 Spartan-3 5000 FPGAs[27]. Using a brute force attack, with a guessing rate of one thousand billions keys per second 10^{12} keys/sec[26], it would take less than day to find the key. Therefore DES is not considered to be a secure algorithm and has been removed from TLS 1.2.

4.1.2 3DES

The 3DES algorithm uses either two or three effective DES keys which are 56-bit keys which provide security from 56-bits to 168-bits depending on the key option. The 3DES has three key options:

- **Key option 1:** For key option 1, all three keys are independent from each other giving the strongest security having a key of $3 \cdot 56 = 168$-bits.
- **Key option 2:** In key option 2, two of the three keys are independent, giving this option a security of $2 \cdot 56 = 112$-bits. This size of key is a few bits lower than recommended minimum of 128 bits[25].
- **Key option 3:** Here, all keys are identical which is equivalent to DES, giving only a 56-bit key of security. This option is, as mentioned above, no longer recommended by the National Institute of Standards and Technology (NIST).

Using the lowest possible security key (excluding key option 3) gives 2^{112} possible keys which would take $1.6 \cdot 10^{14}$[26] years to check all the keys using the guessing rate 10^{12} keys/sec. That can be judged as fairly safe algorithm but is not considered highly secure by NIST. Even the key length of 3DES is 168 bits, the security strength can be reduced to only 112 bits when the most standard technique to attack 3DES is used, called the Meet-In-The-Middle attack which requires 2^{112} encryption steps [28]. Up until today it has never been cracked and it is even currently used in German passports and banking systems.

4.1.3 Blowfish

The variable length key for Blowfish can range from 32 up to 448 bits, having a 128-bits as a default, using a 64-bit block cipher. The least security that can be given is having 2^{32} possible keys up to the highest security of 2^{448} possible keys with a brute-force time ranging from few minutes up to 10^{115} years. The best known public cryptanalysis is Truncated-differential-cryptanalysis or weak-keys attacks but no effective cryptanalysis of it has been found to date[29]. Blowfish has not been cracked yet and is stated as a secure encryption algorithm today.

4.1.4 AES

Where AES has the key sizes of 128-bits, 192-bits and 256-bits, it is able to produce between 2^{128} and 2^{256} possible keys. Using the least amount security of AES, that is 128-bit key size, it would take 10^{19} years to find the right key using the guessing rate of 10^{12} keys/sec. The AES has not been cracked and brute force attack which is the only effective attack known against it. Although, the best public cryptanalysis is side-channel attack are made to exploit weaknesses in the cryptographic algorithm implementation and thus are not strictly related to this thesis in that context. As for today, no known practical attack has broken the AES algorithm which indicates that AES is secure.

4.2 Execution and throughput speed

When comparing speed performance results from many different researches, one should keep in mind that the CPU power, clock rate and memory of a hardware used for the tests may not be always the same. Not even the implementation may be the same. Consequently, the speed rate must be compared instead of the actual execution time it takes to encrypt a message.

4.2.1 Encryption speed

In the research made by Aamer Nadeem and Muhammad Younus Javed[30], performance comparison is made between DES, 3DES, Blowfish and AES. As a basis for the time measurements, a Pentium-II 266 MHz machine and a Pentium-4, 2.4 GHz machine, both running Microsoft Windows XP operating system, where used. Both of the tests where made in ECB mode which is the most straight-forward way of processing message blocks. The platform used was Java platform (JDK 1.4) and was chosen by the authors due to benefits reasons that are irrelevant to this thesis. However, choosing the Java platform to implement the ciphers had the drawback that it slows the speed of the encryption which could be have been a concern where the performance of the algorithms were tested but as stated before, using the same language (Java) and platform, the effect of inefficiency was balanced out. The main purpose of this comparison in the research was "to compare the relative performance of various popular algorithms" [30] only measuring encryption time and without initialization and key-setup but with arbitrarily key sizes.

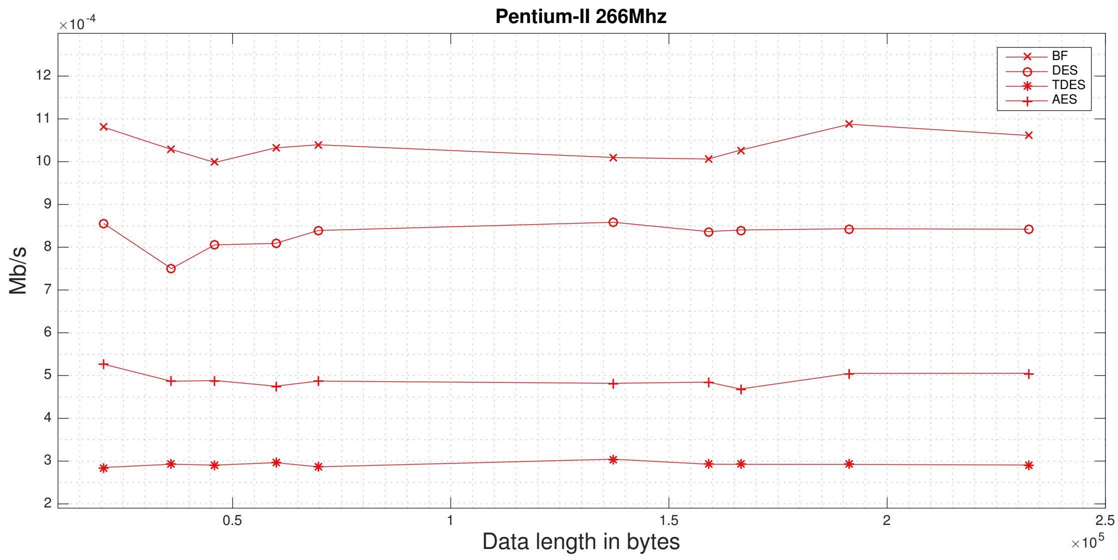


Figure 4.1: Comparative execution throughput of secret key algorithms in ECB mode on a Pentium-II 266-MHz machine[30].

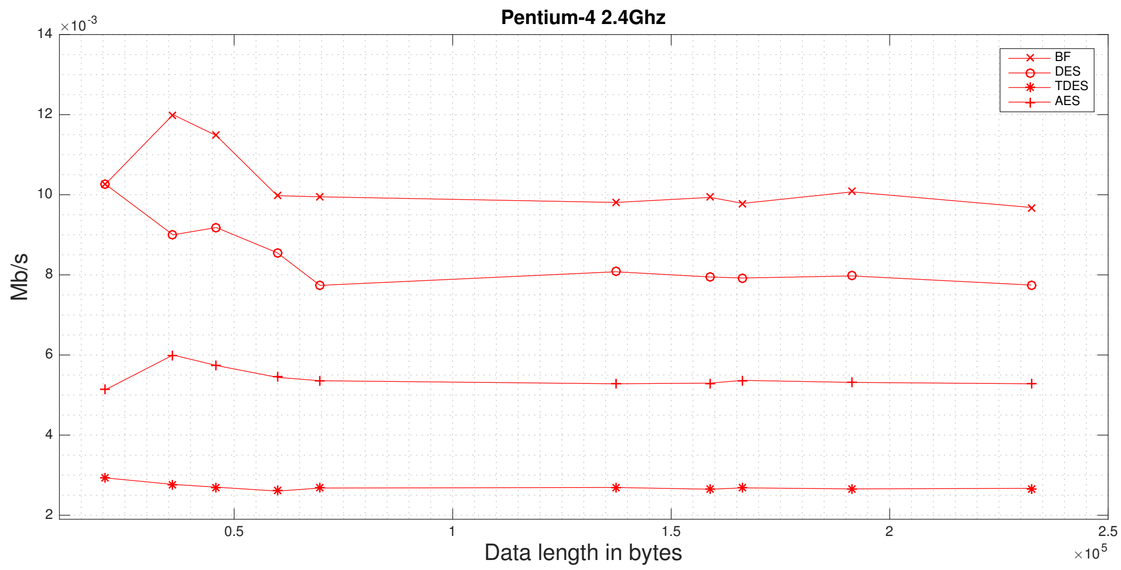


Figure 4.2: Comparative execution throughput of secret key algorithms in ECB mode on a Pentium-4 2.4-GHz machine[30].

In figures 4.1 and 4.2 one can observe that the average throughput is from 11 second up to 383 seconds which is really slow but one should not be concern about from this research. The main conclusion taken from the figures is that the performance of 3DES is really poor compared to AES, DES and Blowfish. The DES is almost two times faster than AES but still slower than Blowfish. Then again in research [31], made by Gurpreet Singh and Supriya, where DES, 3DES and AES were compared, the 3DES algorithm is also really slow. The other two block ciphers DES and AES have a similar speed performance though the Blowfish is the fastest in encryption for these data sizes. The reason for the poor speed performance of the 3DES can be related to the fact that it has to perform three iteration of the normal DES algorithm to meet the security standards that DES does not fulfill.

Timo Bingmann posted a blog the 14th of July 2008 named "Speedtest and Comparison of Open-Source Cryptography Libraries and Compiler Flags"[32]. There he compared many of the well-known open-source cryptography libraries available, which can implement many different ciphers on a Pentium 4 CPU at 3.2 GHz computer. In that research, OpenSSL the predecessor of ARM mbed TLS, is one of the compared libraries. There he compares the throughput of AES(Rijndael), Blowfish, CAST5 and 3DES as function of data length or size. One should notice that CAST5 is a cipher which is not in the thesis scope and will not be noticed or considered any further. Each test made consists of one encryption process immediately followed by decryption of a decided buffer sizes ranging from 16 bytes to 1MB. In figure 4.3, the AES is performing as the fastest cipher up to 20 kB but then the Blowfish takes the lead. That results reflects the research done by Nadeem and Javed where the Blowfish is reasonably faster than AES and 3DES having a larger than 20kB of data size.

Figure 4.3 shows clearly how the length of the encrypted data affects the speed and performance of the cipher algorithm. That is due to the key preprocessing/initialization time which becomes less effective with bigger buffer size. Although the key size was not specified in those test but assumed they have been set to be default sizes, that is AES(128-bit), DES(56-bits), 3DES(168-bits) and Blowfish(128-bits).

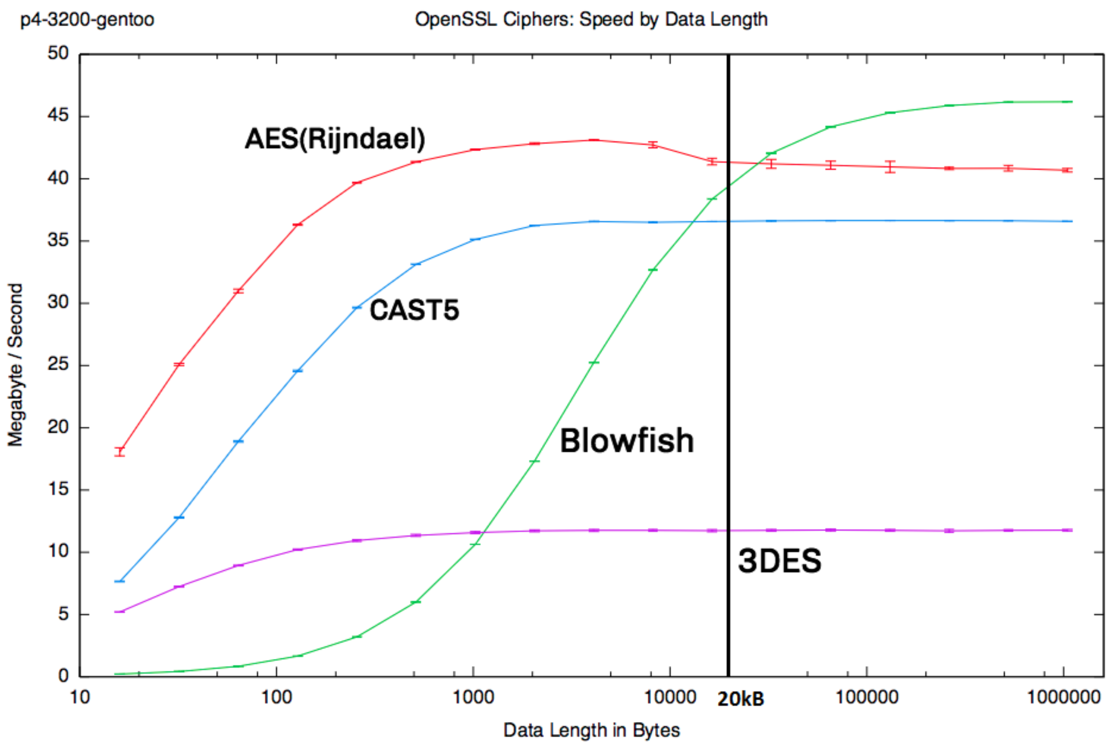


Figure 4.3: Performance of ciphers in OpenSSL.

When comparing speed performance of algorithms in encryption libraries, it is a great idea to see how it scales with different hardware or CPU power. From the test in [32], all the algorithms from its specific library was compared using a different computers with different clock rate. Using the OpenSSL library on a Intel Pentium 2 at 300Mhz computer the following results in figure 4.4 can be investigated.

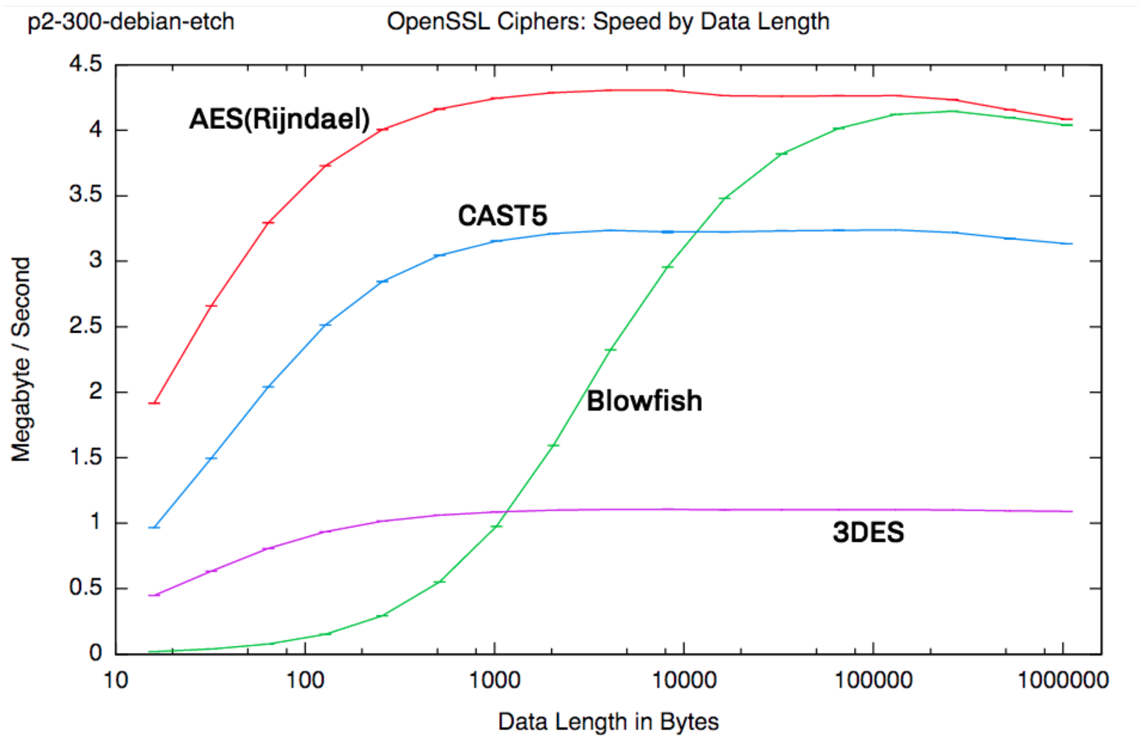


Figure 4.4: Performance of ciphers in OpenSSL on a Intel Pentium 2 at 300Mhz.

From the test results in Figure 4.4, figures A.1 and A.2 in appendix, one can draw the conclusion that the speed of the algorithm scales almost linearly with the CPU power within the same platform (32-bit computer), library and language. This feature can help when trying to decide which encryption algorithm should be chosen on a 32-bit hardware with relatively low CPU speed performance when having only researches which where made on 32-bit hardware with fast CPU. No other important observations can be found on the Charts in Figures 4.4, A.1 and A.2 which is really convenient in order to take an educational guess on how the ciphers AES, Blowfish, 3DES and DES will perform in speed on a 32-bit IoT device. Due to the fact that a speed performance test was not doable for this thesis it is save to assume from the test results from above leads to the conclusion that on average, for long data lengths Blowfish is faster than the AES, DES and 3DES. But for small data lengths, that is smaller than 20kB packages, the AES is the fastest of the encryption algorithms in this specific library. One thing should be cleared out, encryption time is generally the same as decryption time for almost all examined algorithms but the performance can change by different key sizes and key setups.

4.2.2 Key setup and IV

In table 4.1, a speed benchmarks were made by Wei Dai[21] for some of the most commonly used cryptographic algorithms from the Crypto++ 5.6.0 library. The algorithms were compiled with Microsoft Visual C++ 2005 SP1, optimized for speed which includes using the default key size and ran under Windows Vista in 32-bit mode on an Intel Core 2 1.83 GHz processor.

Cipher	Setup Key and IV[µS]	Cycles to Setup Key and IV
AES-CTR-128-bit	0.698	1277
DES-CTR	8.309	15320
3DES-CTR	27.317	49989
Blowfish-CTR	62.683	114710

Table 4.1: Key setup and IV comparison on a Intel Core 2 1.83 GHz processor PC using the Crypto++ 5.6.0 library[21].

Here the AES is significantly faster than DES, 3DES and Blowfish in the process of setting up the key and IV in CTR block cipher mode. This is crucial where the initialization of a key and IV is a frequent action. The main reason that Blowfish makes initial key setup a fairly slow operation is that it makes a brute-force (key-exhaustion) attack difficult.

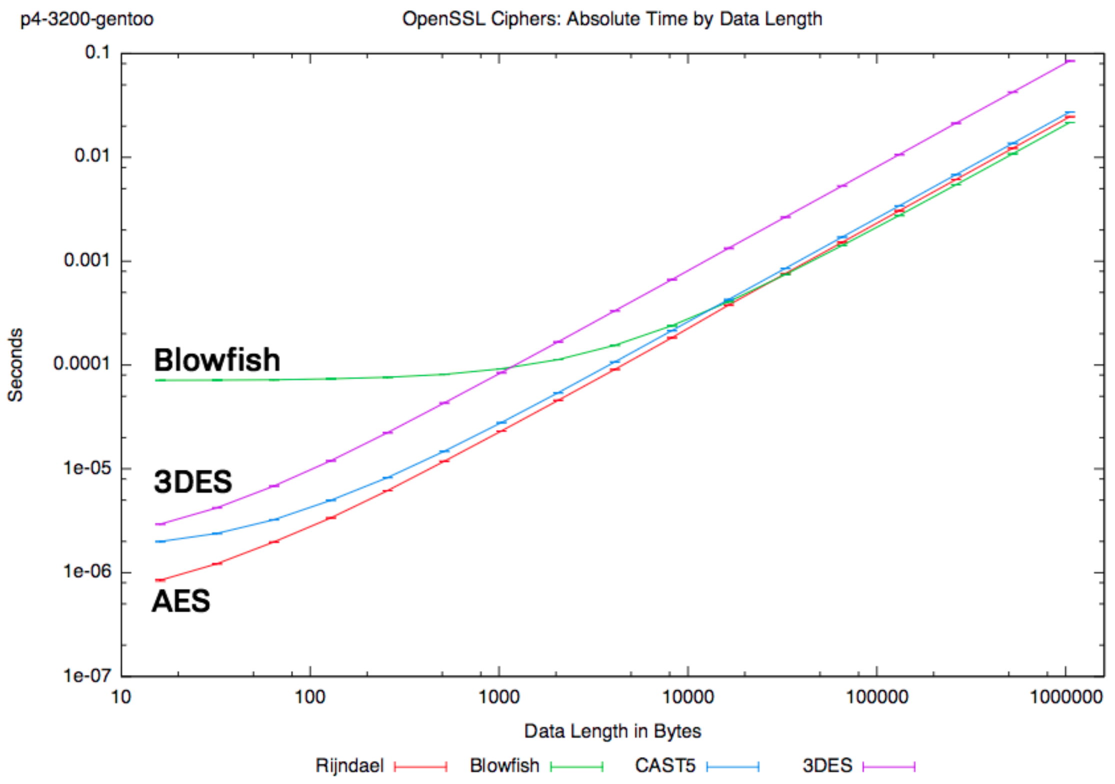


Figure 4.5: Absolute time in seconds required to run one unit of a small buffer size of the speed test [32].

From the test made by Timo[32] it also shows how Blowfish has the slowest start-up to reach it's peak performance in both OpenSSL library and Crypto++. AES, 3DES and Blowfish are all acting and showing the same characteristics for the OpenSSL library, that is, Blowfish is picking up speed between 10kB and 20kB of data length where the key initialization time becomes less relevant to the actual encryption/decryption time, AES is the fastest to begin with and 3DES is the slowest in all buffer sizes. In figure 4.5, a test was done by Timo[32] where the actual time taken to encrypt and decrypt each buffer size at the time. If only the small buffer sizes are investigated, one can measure the library overhead and cipher key preprocessing/initialization time indirectly.

There it shows clearly how the AES performs the fastest of the ciphers in the beginning with small data length but after the buffer size gets bigger, it will almost correlate with Blowfish but 3DES will give a worse performance as often in speed performance.

4.3 Space complexity

In research [33], written by Asif Mushtaque and three others, a comparison was made by DES, 3DES, AES and Blowfish algorithm based on space complexity. There he investigates how size of a plaintext will expand after it has been encrypted with different encryption algorithms. A plaintext of 240 KB size was encrypted with each encryption algorithm, then the size of the produced ciphertexts were compared, as can been seen in following table 4.2.

Cipher	Plaintext	After Encryption	After Decryption
DES	240KB	328KB	240KB
3DES	240KB	614KB	240KB
AES	240KB	847KB	240KB
Blowfish	240KB	955KB	240KB

Table 4.2: Comparison of space complexity [33].

From the results of this experiment, the DES algorithm is distinctly using the least amount of space compared to the other algorithm. There the Blowfish requires bit more of space for the ciphertext than the AES algorithm which could be a crucial factor in amount of data sent between IoT client and a server-side. Also if a IoT device appears to be offline and wants to store the incoming encrypted data until it gets back online again, the small memory IoT device might not afford the wasted space.

4.4 Key size

For all the algorithms in this thesis the key size affects the execution speed, that is, using larger keys decreases the execution speed. In a research made by Wei

Dai[21], the speed of AES using different key sizes using CBC and also for CTR are investigated. One can notice that one column is in MiB per second but MiB is a unit symbol for *mebibyte* which is a multiple of the unit byte for digital information and can be defined as 1 MiB = 2^{20}.

Cipher	Key Size	Cycles Per Byte	MiB/Second
AES/CTR	128-bits	12.6	139
AES/CTR	192-bits	15.4	113
AES/CTR	256-bits	18.2	96
AES/CBC	128-bits	16.0	109
AES/CBC	192-bits	18.9	92
AES/CBC	256-bits	21.7	80

Table 4.3: Comparison of speed performance using different key sizes in CTR and CBC modes. [21].

By comparing the speed performance for both using CTR or CBC for three different key sizes can give a reasonably good idea on how the key size affects the performance of a encryption algorithm. In table 4.3 one can see clearly that increasing the key size will decrease the speed performance of the algorithm. Even if it does not seems much to begin with because it only differs around six cycles per bytes using 128-bits and 256-bits keys, it will greatly affect on the speed when encrypting a data package using large amount of bytes, for instance megabytes or even gigabytes.

4.5 Summary

Comparison Table				
Factor	DES	3DES	AES	Blowfish
Key Sizes (bits)	56	56, 112 or 168	128, 196 or 256	32-448
Cipher Type	Block cipher	Block cipher	Block cipher	Block cipher
Blocks sizes (bits)	64	64	128	64
Rounds	16	48	10,12 or 14	16
First published	1975	1998	1998	1993
Brute Force Attack*	Less than day	Less than a day up to $\approx 10^{34}$ years	10^{19} years up to $\approx 10^{58}$ years	Few minutes up to $\approx 10^{115}$ years
Most effective attack[34]	Differential cryptanalysis	Meet in the middle attack	Side channel attack	Weak key attacks
Cracked	Yes	No	No	No
Security Status	Not Secured	Passes	Secure	Secure
Encryption Speed**	Fast	Slowest	Fastest	Slow
Key Initialization	Medium	Slow	Fast	Slowest
Space Complexity	Great	Good	Good	Bad
*Time taken to find all possible keys using Brute Force with the guessing rate 10^{12}. **Encrypting data length smaller than 10kB.				

Table 4.4: Comparison table of the examined algorithms.

From table 4.4 the first thing to observe is that DES is old, out-dated and not a secure data encryption algorithm. It has been cracked many times and it is doable to find the key with in a day using the brute force attack. It is reasonable fast compared to the other algorithms but in a favor of less security and small key size. Increasing the security with bigger key size of DES, using the 3DES, costs speed. The 3DES has to perform up to three iterations of DES depending on the key option used which will result poorer speed performance and space complexity. Even if it is made to have a better security, it is only fair enough to meet the requirements of FTC to be qualified in TLS 1.2.

The key-initialization of DES is slow which makes it even slower operation for 3DES. The AES and Blowfish are dominant in performance and can be considered a highly security encryption algorithms. They have never been cracked and Blowfish can use up to 448-bit encryption key size and 256-bit for AES which is more than enough to be able to securely store data in the next twenty years, or until the next computer technology breakthrough like quantum computers.

As mention in section 4.2, Blowfish is the fastest algorithm when encrypting data length larger than 20 KB and without using any hardware acceleration. On the other hand, Blowfish is extremely slow in key initialization were AES has the best performance among the other ciphers, giving AES the advantage to be chosen the fastest algorithm for encrypting small data packages as for Parakey IoT device. There are also few advantage for AES over Blowfish worth to mention, AES is well documented to be optimized to speed up the algorithm and was choosen to be the Advanced Encryption Standard. Today, there are many products that have a hardware acceleration for AES which is a great advantage and can dramatically decrease the execution time of the algorithm. Also, AES is the only block cipher of compared ciphers that uses 128-bit block size which will lower the chance of two or more plaintext blocks being the same, causing leak of information about the message contents.[35]

In Table 4.5, the block cipher modes of operations have been compared in terms of given aspects in chapter 3.2.

Comparison Table					
Aspects	**ECB**	**CBC**	**CFB**	**OFB**	**CTR**
Padding schemes	Yes	Yes	No	No	No
Hides repeating plaintext blocks	No	Yes	Yes	Yes	Yes
Proof of security	No	Yes	Yes	Yes	Yes
Parallelization	Encryption-Decryption	Decryption	Decryption	None	Encryption-Decryption
Random access	Reading-Writing	Writing only	Writing only	No	Reading-Writing
Preprocessing	No	No	No	Yes	Yes
Security service*	Confid.	Confid.	Confid.	Confid.	Confid.
Speed Performance	Fast (no IV)	Medium	Slow	Medium	Fast
*All modes only provide confidentiality security.					

Table 4.5: Comparison table of the examined block cipher modes of operations. The highlighted indicates that the mode passes the requirements of each aspect.

CBC and ECB do both need padding which can lead to security breach but where OFB, CFB and CTR are working as a stream ciphers they have the advantage of not needing padding.

Having the ability to encrypt or decrypt in parallel can increase the speed of the algorithm and from table 4.5, ECB and CTR have both the ability to use parallel procedure for encryption and decryption. This gives CTR a great speed performance compared to OFB, CFB and CBC but still ECB is faster where it does not use IV. Also, it is possible to use preprocessing for CTR and OFB which is a good advantage when the speed is a great concern. That being said, the ECB has the best speed performance but provides the worst security. The speed performance of the CBC and CTR are similar but using the preprocessing and parallelization ability the CTR can outperform the CBC when not encrypting a large amount of data.

In Table 4.5 one can notice that ECB is not semantically secure and has not been proven as a secure block cipher mode. By only observing ECB-encrypted ciphertext the information about the plaintext can leak out. This characteristic is already enough to imply that ECB will not achieve any desirable privacy property.

As stated before, the three modes of operation of a block cipher, OFB, CFB and CTR effectively turn the block cipher into a stream cipher. While there is a substantial literature on the design of stream ciphers, it remains useful to be able to use a block cipher in the kind of environments where stream cipher properties might be preferred. One motivation for considering a stream cipher mode might be when we need to encrypt single characters, instead of whole blocks. For this purpose the CFB, OFB, and CTR modes would all be suitable.[9]

Unfortunately, no message integrity nor authentication is provided for those types of block cipher modes of operation but so called MAC (message authentication code) is typically used whenever message integrity is required. The two most common modes are the CCM (Counter with CBC-MAC) and GCM (Galois/Counter Mode) which also provide integrity and authentication but will not be investigated further in this thesis.

Selecting an operation mode may depend on the data package size and amount of data. When encrypting a lot of small package like messages including few strings and variables, a block cipher mode acting like a stream cipher would be a great choice such as CTR it does not need padding, can use preproccessing and process in parallel.
From the requirements in chapter 3.2, the results in Table 4.5 indicates that the CTR block cipher mode is the most suitable mode for the job. It is fast due to the ability of preproccessing, process in parallel and random access. It is also secure because it hides repeating plaintext blocks and does not need padding.

5 Conclusion

There is always a trade-off between security and performance. The application for Parakey AB has to be really responsive and fast, which forces the encryption algorithm to be fast and secure at the same time. In the thesis, the final conclusion is that the best and most convenient data encryption algorithm for a mid-high end IoT device using ARM mbed-TLS library is the AES algorithm using 128-bit key size in CTR block cipher mode.

AES has also one advantage over the other encryption algorithms. As the name indicates, AES is an industry standard, is not patented and free for all to use. One of the attribute of an AES algorithm is that it should work securely on all platforms as hardware and software and therefore AES is well documented and world-spread. That means if AES gets broken, regardless of who it happens with, the headlines will not be about the company that used the hacked AES, they will be about the flaws in something that many companies and people are using. On the other hand, if a company uses a patented algorithm, for example Camellia, the headlines should be on why the company used the algorithm that led to a security breach.

Parakey uses small package sizes and according to table 4.3 in Chapter 4, AES has the best performance encrypting small data, is fast, has proven to be secure and is expected not to be broken in the next 30 years or until the arrival of the next technology breakthrough, such as the quantum computers. The data which are being secured is not top secret allowing the key size of the AES to be only 128-bits which will also increase the encryption speed. AES is a well documented algorithm and free to anybody, is suitable for hardware and software implementation and many chips for instance STM32F0XX, have hardware acceleration for AES algorithm which can enhance its performance even more.

Choosing the most suitable block cipher mode of operation can be hard when they have many different features. For a mid-high end IoT hardware with rather low CPU speed but demanding a fast encryption method for encrypting small amount of data, the CTR is the most suitable. CTR has the ability to preprocess and encrypt the data in parallel which is a great feature to increase the speed of the algorithm. CTR does not have any known flaws and is proven to be a secure block cipher mode, which does not need padding and hides repeating plaintext blocks.

If Parakey would add a feature to their device which could collect large chunk of files such as images and video, the space complexity could grow as an issue for algorithms like Blowfish and AES where the encrypted data becomes much larger than the original data as can be seen in table 4.2. Choosing a mode of operation, the CBC would be a better choice where CBC takes bigger blocks in each round to encrypt the data. Then the Blowfish would also be a good choice for an encryption algorithm where its peek performance is when encrypting large amount data where it does not have to generate or setup new keys or IV often. This is just a speculation indicating that type of data which being encrypted will affect how to choose an encryption algorithm or an operation mode.

5.1 Recommendation for future work

It is really hard to make an absolute conclusion on which encryption algorithm is the best for a specific task with specific constraints without doing any test by yourself and only relying on other studies and make an educational decision. The cryptography world is huge and the amount of ciphers, algorithms and libraries is way too large for one person to grasp on, especially if that person does not have a great cryptography background. The best way to find the most suitable data encryption algorithm for a project is to have a clear idea of the constraints and purpose of the project, and good programming and cryptography knowledge to be able to implement the AES algorithm into the hardware or software and make tests on it. That would give a great overview and performance results of a cipher on that specific hardware.

As stated before, only the confidentiality is provided by the block cipher modes ECB, CBC, OFB, CFB and CTR. In this thesis, the authentication and integrity was not included as an issue but in future work it should be investigated in order to provide more secure communication in IoT devices. As a starter, it would be good to look for AES-GCM (AES operating in Galois/Counter Mode (GCM)) mode or AES-CCM (Counter with CBC-MAC).
AES-GCM is fast and secure and works similar to stream ciphers and can achieve high speeds on low power hardware. The one drawback of using AES-GCM is that it only supported on TLS 1.2 and above but at the moment TLSv1.2 is not widely used.

While existing Internet security technologies, like SSL/TLS, can do a good job protecting communications channels between an uncompromised edge node and a server, they are not invincible. SSL/TLS does nothing to protect against attacks that do not involve the incoming network. It should be easy to see that SSL/TLS does not help if an attacker takes control of an edge node [12].

A physical attack on an IoT device involves a physical access to an IoT device which can led the attacker use the vulnerabilities as mention in OWASP[3] :

- Firmware extraction
- User CLI
- Admin CLI
- Privilege escalation
- Reset to insecure state

- Removal of storage media
- Tamper resistance
- Debug port
- Device ID/Serial number exposure

For a future work, this security issue should be investigated further, focusing on ways to prevent a physical attack on the IoT device. Although, there are also other factors to keep in mind when making an IoT device with Internet connection:

- Include security in the development process from day one.
- Do not collect data that you do not need
- Let the customer choose what data to share
- Secure all layers
- Know your enemy and prepare for security breaches [4]
- Lifecycle, future-proofing updates [4]
- Access control and device authentication [4]

Security is fundamental for the successful rollout of the Internet of Things. Edge nodes are currently the weakest link in ensuring IoT security and the protection of cryptographic key. The best way to achieve lockdown is by protected hardware. It is the only way to keep those keys and other secrets away from prying eyes. An IoT device can only be as secured as its weakest link.

Bibliography

[1] ITU. ITU Internet Reports 2005 The Internet of Things Executive Summary. [Online] Available:

[2] Rob van der Meulen, "Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015", STAMFORD, November 10, 2015. [Online] Available:http://www.gartner.com/newsroom/id/3165317

[3] "Top IoT Vulnerabilities", OWASP the free and open software security community, 29. November, 2015. [Online] Available: https://www.owasp.org/index.php/Top IoT Vulnerabilities

[4] Nermin Hajdarbegovic, "Are We Creating An Insecure Internet of Things (IoT)? Security Challenges and Concerns", 2015. [Online] Available:https://www.toptal.com/it/are-we-creating-an-insecure-internet-of-things

[5] C. Bormann, M. Ersue and A. Keranen, "RFC 7228 on Terminology for Constrained-Node Networks", May 2014.[Online] Available: http://www.rfc-editor.org/rfc/rfc7228.txt

[6] Bruce Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson,"AES Performance Comparisons",[Online] Available: http://csrc.nist.gov/archive/aes/round1/conf2/Schneier.pdf

[7] Wind River Systems,"Security in the Internet of Things",January 2015.

[8] Ramesh.A, Suruliandi.A, "Performance Analysis of Encryption algorithms for Information Security", IEEE, 2013. DOI: 10.1109/ICCPCT.2013.6528957

[9] Knudsen, L.R.; Robshaw, M.J. The Block Cipher Companion, New York: Springer-Verlag, 2011

[10] Tsai, C., Lai, C., & Vasilakos, V. (2014). Future internet of things: Open issue and challanges. *ACM/Springer Wireless Networks*,. doi:10.107/s11276-014-0731-0.

[11] Li, S.; Xu, L.D. and Zhao, S., "The internet of things: a survey", *Information Systems Frontiers*, vol. 17, No.2, pp 243-259, April 2015, DOI:10.1007/s10796-014-9492-7

[12] Atmel Corporation,"Integrating the Internet of Things: Necessary building blocks for broad market adoption", San Jose, USA: Atmel, 0776 Corporate IOT WhitePaper US 102014.

[13] Bandyopadhyay, D. and Sen, J., "Internet of Things: Applications and Challenges in Technology and Standardization", vol. 58, No. 1, pp. 49-69 , May 2011, DOI: 10.1007/s11277-011-0288-5.

[14] Li, T. and Chen, L., "Internet of Things: Principle, *Framework and Application" in Future Wireless Networks and Information Systems*, Zhang, Y., Volume 144

of the series Lecture Notes in Electrical Engineering, Springer-Verlag Berlin Heidelberg 2012, pp. 477-482

[15] R. Shirey "Internet Security Glossary",*The Internet Society*, May 2000.[Online] Available: https://www.ietf.org/rfc/rfc2828.txt

[16] Johannes A. Buchmann, "Introduction to cryptography", 2nd ed., New York, Springer, 2004. DOI: 10.1007/978-1-4419-9003-7

[17] Cirani, S., Ferrari, F. and Veltri, L., "Enforcing Security Mechanisms in the IP-Based Internet of Things: An Algorithmic Overview", *Algorithms*, vol. 6, no. 2, pp. 197-226, June 2013. [Online] Available: http://www.mdpi.com/1999-4893/6/2/197

[18] Swait, "Security Advisory 2868725: Recommendation to disable RC4", *Security Research and Defense Blog*, Nov. 12, 2013, Available: http://blogs.technet.com/b/srd/archive/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4.aspx.

[19] Prasetyo, K.N.; Purwanto, Y. and Darlis, D., "An implementation of data encryption for Internet of Things using blowfish algorithm on FPGA" Information and Communication Technology (ICoICT), 2014 2nd International Conference, pp.75-79, DOI: 10.1109/ICoICT.2014.6914043.

[20] Phillip Rogaway, "Evaluation of Some Blockcipher Modes of Operation", *University of California, Davis*, Davis, California, USA, February 10, 2011.

[21] Wei Dai, https://www.cryptopp.com/benchmarks.html, Accessed: 10th April, 2016.

[22] Nadhem J. Al Fardan and Kenneth G. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". Royal Holloway, University of London. Retrieved 20 April 2016. [Online] http://www.isg.rhul.ac.uk/tls/Lucky13.html

[23] Paar, Christof; Pelzl, Jan; Preneel, Bart. Understanding Cryptography: A Textbook for Students and Practitioners, Springer, 2010. ISBN 3-642-04100-0.

[24] Hagai Bar-El, "Introduction to Side Channel Attacks", Discretix Technologies Ltd.

[25] Sheffer, Y., Holz, R. and P. Saint-Andre, Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), RFC 7525, May 2015. [Online] Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc7525.txt.pdf

[26] Riman, C. and Pierre E. Abi-Char, "Comparative Analysis of Block Cipher-Based Encryption Algorithms: A Survey", *Information Security and Computer Fraud*, Vol. 3, No. 1, pp. 1-7, 2015. DOI:10.12691/iscf-3-1-1

[27] SciEngines, "Break DES in less than a single day". [Online] Available: http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html

[28] Lucks, S.,"Attacking Triple Encryption", *S. Vaudenay (Ed.): Fast Software Encryption* , pp. 239-253, Springer-Verlag Berlin Heidelberg, 1998.

[29] O.O. Adekanmbi, O.O. Omitola, T.R. Oyedare, S.O. Olatinwo, "Performance Evaluation of Common Encryption Algorithms for Throughput and Energy Consumption of a Wireless System", *Journal of advancement in engineering and technology*, vol.3, No.1, pp. 1-8, June 19, 2015.

[30] Nadeem, A. and Dr M. Younus Javed, "A Performance Comparison of Data Encryption Algorithms", September 2005, DOI: 10.1109/ICICT.2005.1598556

[31] Singh, G. and Supriya, "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security", *International Journal of Computer Applications*, vol. 67, No.19, April 2013.

[32] Timo Bingmann, "Speedtest and Comparsion of Open-Source Cryptography Libraries and Compiler Flags", July 14, 2008. [Online] Available: https://panthema.net/2008/0714-cryptography-speedtest-comparison/

[33] Mushtaque, A.; Hussain, S.; Dhiman, H. and Maheshwari, S., "Evaluation of DES, TDES, AES, Blowfish and Two fish Encryption Algorithm: Based on Space Complexity" *International Journal of Engineering Research and Technology (IJERT)*, vol.3, No.4, pp. 283-286, April 2014.

[34] P. Jindal and B. Singh, "Analyzing the Security-Performance Trade-off in Block Ciphers", *ICCCA2015*, pp. 326 - 331, 16th May 2015. DOI:10.1109/CCAA.2015.7148425

[35] "Why exactly is Blowfish faster than AES?", April 17, 2013. [Online] Available: http://crypto.stackexchange.com/questions/8009/why-exactly-is-blowfish-faster-than-aes

A

Appendix 1

Performance results from Timo[32] using the OpenSSL library with different computers.

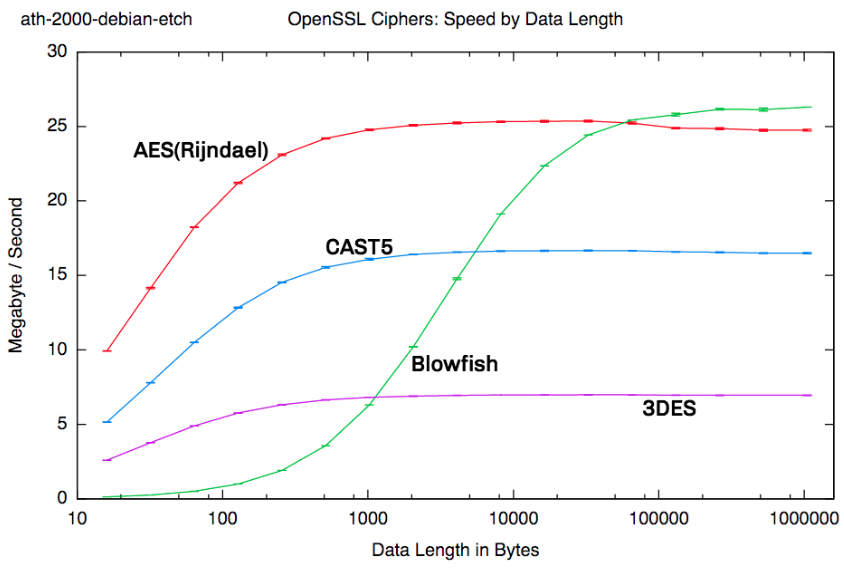


Figure A.1: Performance of ciphers in OpenSSL on a AMD Athlog XP 2000.

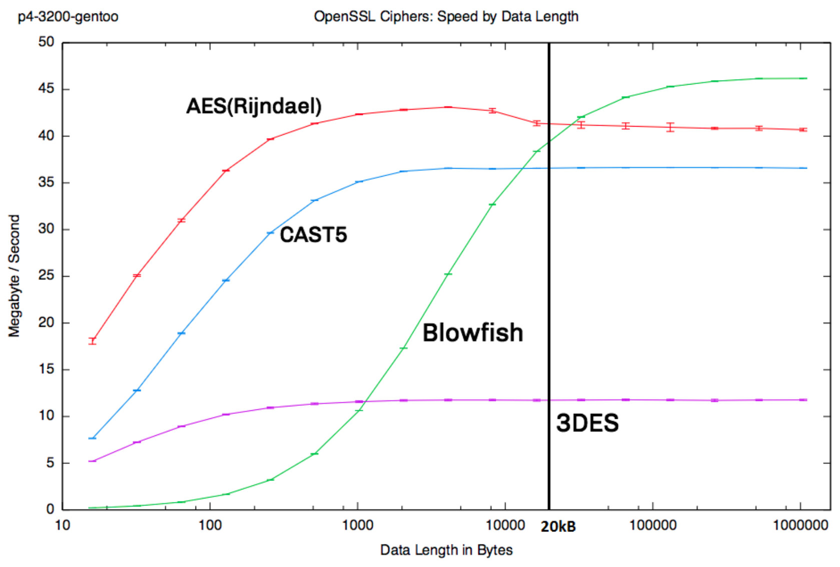


Figure A.2: Performance of ciphers in OpenSSL on a Intel Pentium 4 at 3.2Ghz.

Speed comparison of AES-128-bits for different block cipher modes of operations was made by Wei[21] with following results.

Cipher	Cycles Per Byte	MiB/Second	Cycles to Setup Key and IV
AES-ECB	16.0	109	462
AES-CBC	16.0	109	1041
AES-OFB	16.9	103	1285
AES-CFB	16.1	108	1695
AES-CTR	12.6	139	1277

Table A.1: Comparison of speed performance using different block cipher modes for AES-128-bits. [21].