TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Yulia Derbeneva 184925IVSB

Comparative Analysis of Encryption Algorithms for Resource Constraint Environment

Bachelor's thesis

Supervisor: Tauseef Ahmed
Ph. D

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Yulia Derbeneva 184925IVSB

Krüpteerimisalgoritmide võrdlev analüüs piiratud ressurssidega keskkonna jaoks

Bakalaureusetöö

Juhendaja: Tauseef Ahmed
Ph. D

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Yulia Derbeneva

17.05.2021

Abstract

The Internet of Things is a vast and rapidly growing technology that has evolved into such a large scale project as Smart Cities. Internet of Things nodes introduce security vulnerabilities due to data collection and transmission. Furthermore, they have many constraints and limitations relating to computational and power resources, which could be crucial for data security [1].

The aim of this thesis is to evaluate the importance of applying security measures and to identify the most efficient encryption algorithms in terms of security and performance for low-powered devices.

The goal of this study is to set security implemented for all communicating devices, whereas environments with limited resources require a specific approach for implementing enhanced security measures.

Encryption application is considered to be an effective defensive measure against cyber threats, which provides data security. Performance analysis of various encryption algorithms, carried out on several constraint environments, can give the concept of the most appropriate options for the analogue devices.

Carried out research in this paper gives a comparative analysis of the encryption algorithms performance conducted in the constraint computational resource environments. The results of this study provides an evaluation of the most appropriate algorithms in terms of performance.

This thesis is written in English and is 51 pages long, including 6 chapters, 23 figures and 8 tables.

List of abbreviations and terms

IoT	Internet of Things
SBC	Single Board Computer
VPN	Virtual Private Network
CoAP	Constrained Application Protocol
BACnet	Building Automation and Control Network
DOS	Denial of Service
DDOS	Distributed Denial of Service
SQL	Structured Query Language
AES	Advanced Encryption Standard
DES	Data Encryption Algorithm
RC4	Rivest Cipher 4
RSA	Rivest-Shamir-Adleman
MD5	Message-Digest 5
SHA	Secure Hash Algorithm
IDEA	International Data Encryption Algorithm
NIST	National Institute of Standards and Technology
NSA	National Security Agency
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
RFID	Radio Frequency Identification
ROM	Read Only Memory
RAM	Random Access Memory
PC	Personal Computer
LAN	Local Area Network
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
LibreELEC	Libre Embedded Linux Entertainment Center
OS	Operating System
TLS	Transport Layer Security
SSL	Secure Sockets Layer
CBC	Cipher Block Chaining
CFB	Cipher Feedback
ECB	Electronic Code Book

CTR	Counter Mode
MD4	Message-Digest 4
XOR	Exclusive Or
DTLS	Datagram Transport Layer Security
SD	Secure Digital
USB	Universal Serial Bus
HDMI	High-Definition Multimedia Interface
RFC	Request For Comments

Table of Contents

1 Introduction

The Internet of Things, also known as IoT, is gradually becoming a part of the everyday life of people. Every year, more and more new items at home become “smart”, giving their owners enhanced control over their environment. In connection with the growing popularity of the IoT, both the demand for ready-made devices and materials for creating custom ones. And as is known, demand creates supply.

The Arduino microcontroller based development kit that appeared on the market was highly appreciated by users, which gave a push to development in the direction of single-board computer development, also known as SBC. Companies such as the non-profit BeagleBoard.org and Raspberry Pi, Asus, Nvidia and many others have also joined the technology race, gradually evolving their product into increasingly powerful and functional single-board computers. Over the time, it has become impossible not to notice their advantages and the opportunities they provided, and thanks to an easy access to information, it became incredibly simple to start using a SBC for projects without special skills. With the improvement of SBCs and their popularity, their scope has also expanded. From simple controllers of light, music, cameras, users began to use computers for more educational and entertainment purposes, such as a weather station, game console, a statistics monitoring mirror. From a simple device quickly, SBC grew into an efficient device with the ability to execute complex tasks. Now SBC are also used as web servers, databases, network file system clients, firewalls, Virtual Private Networks, network attached storages and much more [2]. While the term „Thing” in the IoT refers to a device having following properties:

- Unique identity
- Capable to connect with other devices
- Self powered with long lasting battery
- Capable to perform control commands [3]

Currently there are a massive amount of online courses, including provided by the well-known CISCO company, on how to create an IoT device using a SBC [4][5].

Now communication between devices goes beyond one home and develops into a larger network with interconnected devices. Moreover, smart cities have already become a reality. Entire urban zones of connected devices have been created that collect and transmit various data. Now it is not the transfer of data from one to another instead about the huge flows of information associated with the whole society. And here the question of data protection arises sharply, how to allow technologies to progress while maintaining the confidentiality, availability and integrity of all valuable information. These three principles are the basis of information security, the practical application of which is the basis of this study [6][7]. While the aim of this research is to apply in a resource constraint environment one of the information security measures – cryptography. SBCs are used as an example of the constraint environment on which basis a comparative analysis of different cryptography algorithms is made. The provided results create an overview of their overall performance in the constraint environment and in the comparison to other tested algorithms. Used lightweight algorithms show their performance differences compared to standard algorithms. The test outcome gives an understanding of whether those algorithms can be used in such environments, are they effective and suitable, and what matters they arise.

2 IoT and Security

This chapter describes why security measures should be also implemented within the IoT devices, what threats and vulnerabilities IoT are subject to, and why cryptography is an important security measure that can be implemented for risk mitigation.

The IoT would not be integrated into the everyday life of people if its capabilities and advantages were not appreciated. According to statistics, by 2030, the number of IoT devices around the world will grow by 15.4 billion, which is more than double the current number of devices. They are used in all types of industries, but as of 2020, 60 percent of all connected devices are in the consumer market [8].

On the other hand, the desire to make all devices “smart” creates additional access spots for hackers and new tasks for cyber security professionals. If solutions to reduce security risks will not be provided, then, the surface of risk will substantially extend and grow in the same progression as the number of devices.

In addition to the risks of information loss, which can lead to financial losses, many IoT devices are safety-related. Connected to medical sensors, fire alarm and surveillance systems, door locks, the compromise of such devices may lead to physical harm and property and even death. That is why the use of protective measures is very necessary for the IoT devices.

However, IoT devices have limited functionality and limited resources, such as storage, processing capacity, and limited power consumption, and therefore are generally not designed with any security mechanisms. Also, their low cost affects the increased demand, so manufacturers may also compromise on security measures in order not to lose their bulk of buyers [9][10]. These factors make IoT devices an easy target for intruders, although both humans and natural forces can be the source of threats. Unfortunately, natural disasters cannot be prevented, so the usage of a disaster recovery is the best approach to protecting a system. While human-related threats require a more thorough approach. Even if all risks cannot be ruled out, it is necessary to eliminate the vulnerabilities with the most likelihood and impact.

2.1 Threats and Vulnerabilities

IoT is a combination of opportunities and risks that new technologies bring. Unlike the already standardized processes and security measures for traditional computers, IoT devices have bounded security functions, while they are directly connected to the Internet, and are a part of a system that does not have proper security procedures implemented. Therefore, major IoT security risks are connected to their main differences from traditional technologies, such as:

- Interaction of active analogue and physical elements are the main characteristics of the IoT devices. Since resources are often obtained from the physical world, the edge nodes are accessible for a physical contact, therefore have additional risks [11].

- Massive scale points increase the likelihood of attacks. Despite the fact that the computing power of such devices is rather small in comparison with computer ones, being combined into a system of such a scale, they become a truly threatening force.

- A wide variety of network types. Protocols such as Constrained Application Protocol, 6TiSCH, ZigBee, Building Automation and Control Network have been created and optimized for a limited environment. However, optimization is also followed by related reliability and transmission issues.

- Resources such as battery, memory and performance are limited. This leads to the use of simpler codes and protocols, which will affect the ability to enforce protective measures. [1]

- The necessary policies and frameworks are still under development as IoT technology began to gain popularity not so long ago, unlike standard computers. [12]

The potential risks can also be considered depending on which architecture layer they affect. Despite the fact that there is no general agreement on the IoT architectural layer, the researchers observe the need for a four-layer architecture [13]. With certainty three

main architectural layers, such as Physical, Network and Application layers, can be distinguished.

1. The physical layer includes objects that collect information and control the data of executive devices that transmit information about their work.

2. The network layer transfers information obtained using physical objects to applications.

3. The application layer provides the services to the applications which are using IoT technology [14] [15].

Table 1 shows the basic three-layer architecture of IoT and the parameters that need a special attention in ensuring security. The table also includes threats that are common to each of the architectural layers [16][17][18]. Despite the fact that there are no uniform protection measures to cover all vulnerabilities, cryptography plays a significant role in the protection of IoT technologies.

Table 1: IoT Architecture Layers, Security Parameters and Major Threats.

Architecture Layer	**Security Parameters**	**Security Threats**
Physical Layer	• Device immunity • Authentication • Access Control • Integrity	• Unauthorised access • Denial of Service • Channel blocking • Sybil attack • Replay attack • Tunnel attack • Synchronization attack • Data tampering • Electromagnetic leakage and interference
Network Layer	• Confidentiality • Integrity	• Distributed Denial of Service

Architecture Layer	Security Parameters	Security Threats
	• Authentication • Access Control	• Routing attack • Sinkhole attack • Wormhole attack • Blackhole • Spoofing attack • Routing loop attack • False routing information • Trapdoor • Tunnel attack • Spoofing attack • Hello flooding attack • Eavesdropping
Application Layer	• Access Control • User Privacy • User-Anonymity • Accountability • Non-Repudiation • Authentication	• Privacy data leaking • Unauthorised access • Malicious code • Forged control commands • Loophole • Viruses and Trojan horses • SQL injection attack

Considering the gathered information, the IoT needs strong security measures to get protected from such a large scale and variation of threats. One of the measures can be the implementation of cryptography.

2.2 Cryptography and IoT

Cryptography is an essential aspect that is used as a defense mechanism in traditional computer technology and is playing an increasing role in IoT technology. Since, in

addition to providing services and applications, the IoT should also provide a level of trust in the technology, the ability to protect its data. The use of reliable and updated encryption methods can increase security and trust respectively[19].

The advantage of cryptography as a protective measure for IoT is that the goals of cryptography match the security parameters of IoT devices that should be taken into account. Moreover, those parameters are introduced in all three main layers of IoT architecture, that are mentioned in the Table 1, such as:

- Confidentiality refers to providing an access to information only to those who are authorized to do so.
- Integrity ensures data has not been manipulated by unauthorized parties.
- Authentication can be divided into entity authentication, which refers to verifying the identity of the sender, and data authentication that identifies the data has not been tempered during a transmission over a channel.
- Non-repudiation prevents the sender from denying the events committed by the sender.

These security practices are achieved using various encryption techniques, which are divided into two main types, symmetric encryptions and asymmetric encryptions. Some researchers append the third "hashing" type.

Symmetric encryption uses a shared key to encrypt and decrypt information transmitted between a sender and a receiver. This technique is also categorized into two encryption schemes as block and stream ciphers. The block cipher performs an encryption operation by splitting a plaintext into blocks, converting each bit of the plaintext in parallel. The most commonly used symmetric algorithms are Advanced Encryption Standard, Blowfish, Triple Data Encryption Standard, CAST-128 and Rivest Cipher 4.

Asymmetric encryption uses different, pairwise keys for encryption and decryption. Rivest-Shamir-Adleman and Elgamal are well-known asymmetric algorithms.

Hashing is a one-way encryption technique, as a ciphertext cannot be converted back to plaintext. This method ensures data integrity and authenticity. Often mentioned such hash functions are Message-Digest 5, SHA256 and their modified versions [20][21].

Despite the fact that some traditional ciphers, such as AES, Camellia, CLEFIA and International Data Encryption Algorithm, are approved by the International Organization for Standardization for use in environments with limited resources [22]. However, the prevailing number of cryptographic algorithms are not suitable for IoT devices as most of the encryption algorithms require a significant amount of memory, battery and processing power. These resources are consumed by mathematical calculations, data aggregation, output redirection and other processes [23][24][25]. Therefore, with the development of IoT technologies, the need for lightweight algorithms has increased. This is evidenced by the fact that NIST initiated the project on the standardization of lightweight cryptographic algorithms and that the International Organization for Standardization has already adopted the international standards for it [26].

To adjust encryption algorithms for resource constraint environments, lightweight algorithms use smaller circuitry, Read Only Memory and Random Access Memory sizes, processing speed, power that is used by power harvesting devices, and a power consumption for battery-powered devices such as a camera or sensor. Examples of such algorithms are SPECK, SIMON, PRESENT, TWINE [26].

3 Research Technical Background

This chapter includes consideration and analysis of the used devices and encryption algorithms in the course of this study. While a technical analysis of algorithms gives a general understanding of their aspects, methods of use and hints for consideration of these algorithms within other environments. In the research used SBCs are taken as an example of the resource constraint environment. Gather data will provide an overview to how encryption algorithms perform in specific conditions and requirements.

3.1 Tested Devices

In the research two models of the SBCs are used: Raspberry Pi 3 model B and Raspberry Pi Zero W. These devices of the British company named Raspberry Pi Foundation. The products of this company are widely known due to the ratio of quality and price [27]. The Raspberry Pi has the advantages of PC-like computers, great for interacting with additional and auxiliary devices. The ability to connect via Bluetooth or Wi-Fi allows the use of remote control, which makes the product concept well suited for an IoT device [28]. The Raspberry Pi Zero W extends the Pi Zero family and has optional 802.11 b/g/n wireless Local Area Network, Bluetooth 4/1 and Bluetooth Low Energy. While the Raspberry Pi 3 model B is the predecessor to the latest release - Raspberry Pi 4.

Table 2 shows the main parameters of the tested SBCs, which are related to the results of the comparative analysis in the practical part [29][30][31].

Table 2: SBC Technical Specifications.

Parameter	**Raspberry Pi 3 B**	**Raspberry Pi Zero W**
Processor Core	4	1
CPU	1.2 GHz	1 GHz
RAM	1 GB	512 MB
Wireless LAN	BCM43438	802.11 b/g/n
Power Supply	5.1V 2.5A	5.1V 2.5A

The Raspberry Pi supports multiple operating systems such as Ubuntu, Manjaro ARM Linux, RISC OS Pi, LibreELEC, which stands for Libre Embedded Linux Entertainment Center. The Raspberry Pi Foundation has also developed its own Debian-based operating system called the Raspberry Pi OS, which is also available for installation in light and full versions [32].

3.2 Encryption Algorithms

The encryption algorithms used in the research are selected based on their prevalence and approval by authorities. An attempt to cover a wider range of encryption techniques is also pursued.

1. Advanced Encryption Standard, or simply AES. AES has such variations as AES128, AES192, AES256 that are widespread symmetric block cipher algorithms. AES is a part of Transport Layer Security and Secure Sockets Layer standards to ensure secure communication between hosts on the Web, and approved by NIST, which stands for National Institute of Standards and Technology. The latest Central Processing Unit hardware has AES integrated for better processing speed of the algorithm.

   AES uses 128, 160, 192, 224 or 256 bits symmetric keys. It combines plaintext with a provided key, then calculates the ciphertext using a previous result, nonce or initialization vector and the mode. AES algorithms are also using block cipher modes of operation, such as Cipher Block Chaining, Cipher Feedback, Electronic Code Block and others. Those modes have different parameters and are meant to improve efficiency and provide a stronger security. For example, a Counter Mode and random initial vector are recommended to avoid a dictionary attack [33][34]. For the research purpose, intermediate key sizes of 128, 192, 256 bytes and CFB mode, approved by NIST, are chosen.

2. Secure Hash Algorithm 256, or shortly SHA256, is a hash algorithm that is used in the latest SSL TLSv1.2 protocol [35]. This algorithm is derived from a simpler cipher called Message-Digest 4. SHA256 was proposed and approved by NIST for use by federal departments and agencies. The message digest

formed by this algorithm can be used in software, firmware, hardware for determination of a message's integrity [36].

SHA256 breaks a padded message into 512 bits block size that is expressed as a sequence of sixteen 32-bit words and sets initialization values. Then, it uses a message schedule of sixty-four 32-bit words, eight working variables of 32 bits each and a hash value of eight 32-bit words to create a 256-bit message digest [37].

3. SHA3-256 is the SHA-3 family cryptographic hash function recommended by NIST. It is considered to be more secure as it has improved security features such as resistance to collision, preimage and second preimage attacks. SHA3-256 uses 1088 bits block size compared to 512 bits blocks of SHA256 [38]. This algorithm is also taken for the comparative purpose between SHA256 and SHA3-256.

4. SHAKE256 is an extendable-output function of the SHA-3 family approved by NIST. Its main difference from SHA3 functions is that the output message length can vary depending on the requirements. The index 256 indicates the supported security level, not the digest length as for other hash functions [39].

5. Poly1305-AES is a one-time message authenticator. Originally, Poly1305 was designed in combination with the AES algorithm for nonce encryption. Yet in time, it became more widely spread in combination with ChaCha20. However, Poly1305-AES has advantages such as consistent high speed, even for long messages, performance is not influenced by the overflown keys cache and low re-computational cost for modification of long messages [40].

   Poly1305 encrypts the plaintext using a shared by sender and receiver 16-byte AES key, 16-byte additional key and a unique 16-byte nonce that is processed by AES. The message is broken on 16-byte chunks and padded by appending an extra byte to each produced chunk [41].

6. ChaCha20 is a symmetric stream cipher, an improved version of Salsa20 as follows the same basic design but with a higher transmission level per round of

total 20 rounds [42]. ChaCha20 encrypts a plaintext using a keystream produced from a block function applied to the 32-byte key, 12-byte nonce, block counter and plaintext blocks XORed, that stands for Exclusive Or, with the block function output. The result of the algorithm is a truncated 16-byte digest of the message [43].

7. ChaCha20-Poly1305 is an authentication encryption with associated data. This encryption method is a result of the ChaCha20 stream cipher and Poly1305 authenticator combination. ChaCha20-Poly1305 is used for high performance in software implementation and to reduce information leakage through side-channels. This encryption type is supported by SSL TLSv1.2, TLSv1.3 and for Datagram Transport Layer Security protocols [44].

   ChaCha20-Poly1305 uses Poly1305 to generate from a 32-byte key a one-time key and a 96-bit nonce. While ChaCha20 encrypts the plaintext with generated by Poly1305 key and nonce. As the last step, Poly1305 function uses a key to construct a ciphertext of the same length as the plaintext and a 128-bit tag produced by Poly1305 function. It is important not to use the same nonce and the key as it creates identical one-time keys and the key stream that leads to a security vulnerability [45].

8. Speck is a light-weight block cipher introduced by the U.S. National Security Agency in 2015. It is designed to satisfy IoT needs of a light and flexible encryption algorithm. Speck's block and key sizes can be changed based on requirements, from a 32-bit block with a 64-bit key to a 128-bit block with a 256-bit key. The simplicity for the algorithm is kept by using a short list of operations performed by the cipher, such as modular addition, bitwise XOR, left and right circular shift [46].

9. Simon is another light-weight block cipher proposed by the U.S. NSA together with Speck. Simon has similar parameters as Speck does, but a different set of operations: bitwise XOR, bitwise AND, and left circular shift. This difference gives Simon a slight advantage in using a cipher for hardware systems. Additionally, Simon requires more rounds than Speck as it has a weaker non-linear function [42].

In 2014 both Speck and Simon algorithms were proposed by the U.S. National Body for inclusion in the International Organization for Standardization / International Electrotechnical Commission lightweight cryptography standard but did not reach required votes [47]. However, in November of 2018, the ISO published new standards for the use of the block ciphers Speck and Simon. The new standards were adopted for practical applications in the air-interface of RFID, which stands for radio frequency identification technology. This technology has important government and military applications in supply chain management and asset tracking. Speck and Simon algorithms play an important role providing a security to the resource constraint in circuity and power RFID tags that share data vulnerable to exposure or manipulation [48].

The cryptographic algorithms described above are used for a comparison analysis in Chapter 4.2 of the research part. Table 3 shows the parameters overview of the encryption algorithms used in this research [33-46][42].

Table 3: Algorithms Overview.

Encryption Algorithm	Key Size (byte)	Initialization vector (byte)	Nonce Size (byte)	Block Size (byte)	Rounds
AES128	16	16	-	16	10
AES192	24	16	-	16	12
AES256	32	16	-	16	14
SHA256	-	-	-	64	64
SHA3-256	-	-	-	136	24
SHAKE256	-	-	-	136	24
Poly1305-AES	16	-	16	16	-
ChaCha20	32	-	12	64	20
ChaCha20-Poly1305	32	-	12	-	-
Speck	16	-	-	16	32
Simon	16	-	-	16	68

AES256, ChaCha20 and ChaCha20-Poly1305 have the largest key size equal to 32 bytes, while SHA3-256 and SHAKE256 have the largest block size equal to 136 bytes. In the results overview of Chapter 5 can be seen how cryptographic algorithms with specified parameters that are used in this research, perform in the constraint computational resource environment.

4 Research

This chapter includes a description of the practical activities, essential information about the details of the research and step-by-step actions. Also, included an overview of the gathered data after the execution of the encryption algorithms listed in the Chapter 3.2, within the SBC environment.

4.1 Preparation Steps

1. The hardware used for this research: monitor, wireless keyboard, wireless mouse, Raspberry Pi 3 Model B and Raspberry Pi Zero W, 32 GB Secure Digital card, charger 5V 2A, digital multimeter, Universal Serial Bus flash drive for simpler algorithm transferring. For Raspberry Pi Zero W additional hardware items: High-Definition Multimedia Interface cable, mini HDMI to HDMI adapter and micro USB to USB type A adapter.

2. In this study, the Raspberry Pi OS 32-bit operating system is used since it is the most optimized for the SBCs under study [32]. The operating system is installed on a 32GB SD card using the Raspberry Pi Imager v1.6.1 application [49].

3. Created text files with the extension ".txt" for encryption. The file sizes are chosen as follows: 1 byte, 1 kilobyte (1024 bytes), 1 megabyte (1048576 bytes). Intermediate sizes of 100 byte and 0.5 megabyte (524288 bytes) have also been added.

4. Thonny, a built-in Python development environment is used for code execution.

4.2 Execution Steps

Before running the algorithms, required to install the Python third-party package of cryptographic primitives called „pycryptodome". Speck and Simon encryption algorithms are not included in the library and for this reason need to be installed separately. The RFC7539 module, where RFC stands for Request For Comments, is required for the ChaCha20-Poly1305 algorithm. Figure 1 provides commands that are used for the module installation [50][51].

```
pip3 install pycryptodome
pip3 install simonspeckciphers
pip3 install rfc7539
```

Figure 1: Package installation.

To measure the speed results of the algorithm's execution, the “time” module is used. “time()” function returns the current time. It is used in such a way to capture time from the beginning and the end of the encryption process, to receive a final result by subtracting the gathered time amount. Figure 2 shows an example of using the “time()” function [52].

```
import time
start_time = time.time()

# encryption code here

end_time = time.time()
execution_time = (end_time – start_time)
```

Figure 2: Example of the time module application.

All encryption algorithms open a prepared file of a certain size, encrypt the data gathered from it and write a ciphertext to a new file. Figure 3 gives an example of how the files are handled [53]. This approach is used to be able to easily provide large size files for the encryption, whereas the output file is used for memory usage measurements.

```
filename = "/home/pi/Desktop/1024bytes.txt"
with open(filename, "rb") as f:
    plaintext = f.read()
    f.close()

# encryption code here

with open("/home/pi/Desktop/aes128_encrypted.txt",
"wb") as file:
    file.write(ciphertext)
    file.close()
```

Figure 3. Example of file input and output application.

Encryption key is compiled using an “os.urandom()” method presented in Figure 4 [54]. The key length is specified in bytes.

```
from os import urandom
key = urandom(24)
```

Figure 4. Random key compilation.

The encryption algorithms used for comparative analysis are Python-based since it is an officially recommended and most widely used programming language within Raspberry Pi SBCs [55]. The complete code of the cryptographic algorithms is provided in the Appendix 1 in the Figure 5-15.

4.3 Gathered Data

Encryption algorithms are executed three times each to capture the average execution time for every device. The gathered results from the execution and measurements are provided in the Tables 4 – 8 and parameters as follows:

- File size: Plaintext file size presented in bytes.
- Memory usage: Ciphertext file size in bytes that depends on the algorithm output size.
- Execution time: Time of the encryption algorithm execution in seconds.
- Throughput: A plaintext file size divided by the encryption time measured in bytes per second.
- Power consumption increase: The power consumed during a quiescent state subtracted from the power consumed during an execution of the algorithm. The increase in the power consumption of the device during the active phase is indicated in percentage.

Table 4: Data for 1 byte file.

Algorithm	File size (byte)	Memory usage (byte)	Execution time (sec)		Throughput (bytes/sec)		Power consumption increase (%)	
			3 B	Zero W	3 B	Zero W	3 B	Zero W
AES128	1	1	0.00352	0.02261	284	44	84	36
AES192	1	1	0.00360	0.02218	278	45	87	36
AES256	1	1	0.00350	0.02244	286	45	81	36
SHA256	1	64	0.00004	0.00016	25,000	6,250	77	29
SHA3-256	1	32	0.00029	0.00132	3,448	758	84	36
SHAKE256	1	26	0.00094	0.00227	1,064	440	81	36
Poly1305-AES	1	16	0.00086	0.00467	1,163	214	84	36
ChaCha20	1	9	0.00053	0.00225	1,887	397	81	36
ChaCha20-Poly1305	1	1	0.00017	0.00066	5,882	1,515	77	29
Speck	1	36	0.00068	0.00288	1,470	347	77	29
Simon	1	38	0.00129	0.00560	775	179	81	29

Table 5: Data for 100 bytes file.

Algorithm	File size (byte)	Memory usage (byte)	Execution time (sec)		Throughput (bytes/sec)		Power consumption increase (%)	
			3 B	Zero W	3 B	Zero W	3 B	Zero W
AES128	100	100	0.00374	0.02511	26,738	3,982	81	36
AES192	100	100	0.00369	0.02449	27,100	4,083	77	29
AES256	100	100	0.00377	0.02476	26,525	4,039	77	43
SHA256	100	64	0.00004	0.00018	2,500,000	555,555	77	29
SHA3-256	100	32	0.00038	0.00128	263,138	78,125	84	36
SHAKE256	100	26	0.00031	0.00130	322,581	76,923	77	36
Poly1305-AES	100	16	0.00086	0.00559	116,279	17,889	77	36
ChaCha20	100	108	0.00060	0.00229	166,667	43,668	81	29
ChaCha20-Poly1305	100	100	0.00018	0.00071	555,555	140,845	81	21
Speck	100	39	0.00070	0.00400	142,857	25,000	81	29
Simon	100	39	0.00131	0.00580	76,336	17,241	81	29

Table 6: Data for 1024 bytes file.

Algorithm	File size (byte)	Memory usage (byte)	Execution time (sec)		Throughput (bytes/sec)		Power consumption increase (%)	
			3 B	Zero W	3 B	Zero W	3 B	Zero W
AES128	1024	1024	0.00349	0.02307	293,410	44,387	87	43
AES192	1024	1024	0.00343	0.02278	298,542	44,952	84	36
AES256	1024	1024	0.00347	0.02316	295,101	44,214	77	43
SHA256	1024	64	0.000054	0.00020	18,962,963	5,120,000	77	36
SHA3-256	1024	32	0.00040	0.00153	2,560,000	669,281	84	29
SHAKE256	1024	26	0.00041	0.00149	24,975,60 0	687,248	84	29
Poly1305-AES	1024	16	0.00089	0.00544	1,150,562	188,235	81	36
ChaCha20	1024	1032	0.00063	0.00243	1,625,397	421,399	84	21
ChaCha20-Poly1305	1024	1024	0.00023	0.00089	4,452,174	1,150,562	90	39
Speck	1024	39	0.00072	0.00351	1,422,222	291,738	84	29
Simon	1024	39	0.00129	0.00505	793,798	202,772	81	29

Table 7: Data for 524288 bytes file.

Algorithm	File size (byte)	Memory usage (byte)	Execution time (sec)		Throughput (bytes/sec)		Power consumption increase (%)	
			3 B	Zero W	3 B	Zero W	3 B	Zero W
AES128	524288	524288	0.06029	0.15926	8,696,102	3,292,026	81	43
AES192	524288	524288	0.06877	0.18278	7,623,789	2,868,410	77	43
AES256	524288	524288	0.07619	0.19360	6,881,323	2,708,099	84	36
SHA256	524288	64	0.00653	0.02387	80,289,127	21,964,307	81	36
SHA3-256	524288	32	0.04879	0.10784	10,745,808	4,861,721	77	36
SHAKE256	524288	26	0.04819	0.10938	10,879,601	4,793,271	77	29
Poly1305-AES	524288	16	0.01057	0.02645	49,601,514	19,821,852	77	29
ChaCha20	524288	524296	0.01710	0.04304	30,660,117	12,181,413	81	36
ChaCha20-Poly1305	524288	524288	0.02817	0.08132	18,611,572	6,447,221	84	36
Speck	524288	39	0.01034	0.03136	50,704,835	16,718,367	77	36
Simon	524288	37	0.01073	0.03547	48,861,882	14,781,167	42	43

Table 8: Data for 1048576 bytes file.

Algorithm	File size (byte)	Memory usage (byte)	Execution time (sec)		Throughput (bytes/sec)		Power consumption increase (%)	
			3 B	Zero W	3 B	Zero W	3 B	Zero W
AES128	1048576	1048576	0.11682	0.29897	8,975,997	3,507,295	87	43
AES192	1048576	1048576	0.13226	0.33979	7,928,141	3,085,953	87	43
AES256	1048576	1048576	0.14863	0.36398	7,054,942	2,880,862	81	36
SHA256	1048576	64	0.01260	0.04819	83,220,317	2,175,9203	87	36
SHA3-256	1048576	32	0.09432	0.21298	11,117,218	4,923,354	87	36
SHAKE256	1048576	26	0.09465	0.215916	11,078,457	4,856,407	81	36
Poly1305-AES	1048576	16	0.02014	0.04972	52,064,349	21,089,622	84	29
ChaCha20	1048576	1048584	0.03299	0.08541	31,784,662	12,276,970	84	36
ChaCha20-Poly1305	1048576	1048576	0.05555	0.15778	18,876,256	6,645,810	84	36
Speck	1048576	39	0.01970	0.06182	53,227,208	16,961,760	81	29
Simon	1048576	39	0.01733	0.06212	60,506,405	16,879,845	39	36

Code execution has passed without problems. The gathered data is analyszed and processed in the Chapter 5.

5 Analysis of Results

In this Chapter is all the collected data in a Chapter 4.3 is converted into the line and column charts and presented for a visual demonstration. Charts are also used for an easier comparative analysis and a general overview on the cryptographic algorithm's performance.

One of the main indicators of the research performed is the throughput of the algorithms. Separate line charts for each file size are created and include data for two testing devices. Figure 5 shows that Raspberry Pi 3 B has a higher throughput of all the cryptographic algorithms in comparison with Raspberry Pi Zero W device. That result is expected as Zero W has lower overall performance due to technical specifications. From the chart it can be seen that SHA256 algorithm stands out as has the best throughput among other hash algorithms and cryptographic ciphers. However, ChaCha20-Poly1305 also showed good results and reached a throughput of 5,882 bytes per second. For both devices the lowest results have shown all AES encryption algorithms that have gained a throughput of 284 to 286 bytes per second for the 3B model, and between 44 and 45 bytes per second for Zero W.

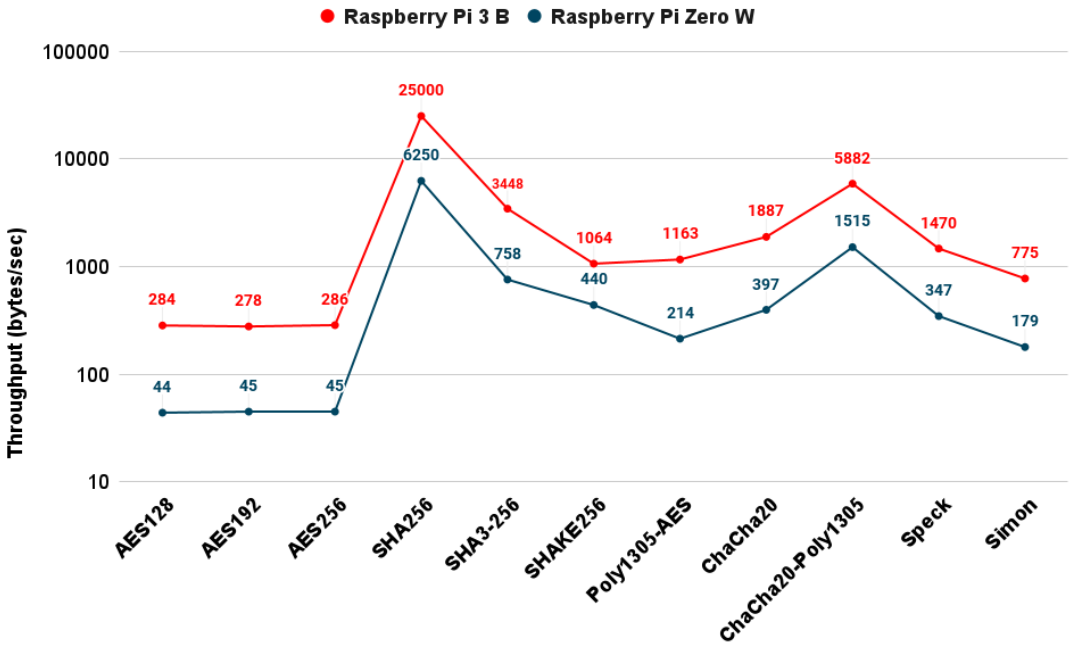

Figure 5: Throughput of 1 byte file.

Chart lines of the Figure 6 and Figure 7 depict results for the encrypted 100 and 1024 bytes files that have similar test outcomes compared to each other. However, the throughput during encryption significantly grown grew 2,500,000 bytes per second for 3 B model and 100 bytes file to 18,962,963 bytes per second for the same device but 1024 bytes file. This pattern also noticed in other algorithms. Also can be noted that SHAKE 256 showed better outcomes for both devices, while Poly1305-AES performed slightly slower for the Zero W model than in the previous tests.

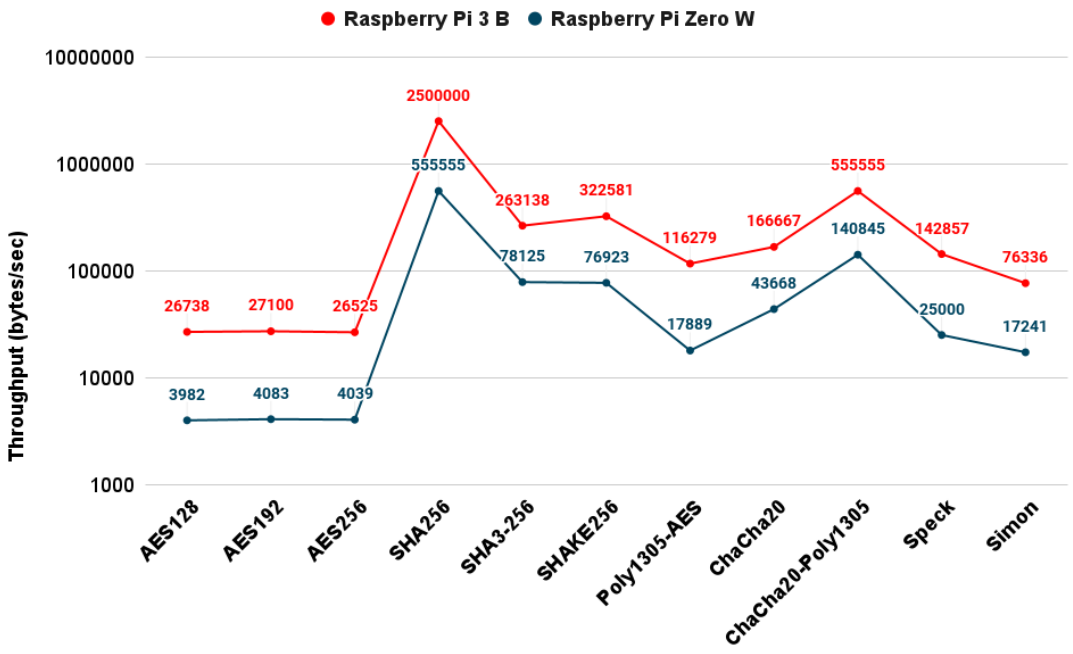

Figure 6: Throughput of 100 bytes file.

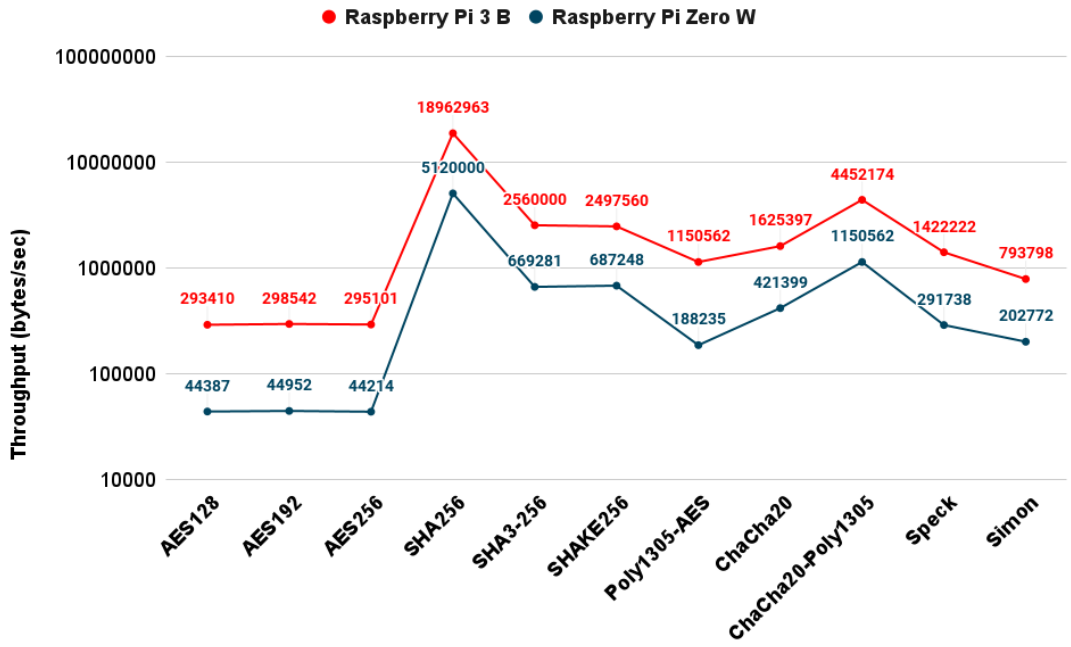

Figure 7: Throughput of 1024 bytes file.

Results for the test execution described in the Figure 8 and Figure 9 showed related throughput results on both devices. Comparing to the previous test of the 1024 bytes size file, an overall throughput of all cryptographic algorithms increased not less than by 4 times. Other changes among algorithms performance also happened due to processing of the large size files. Better results are gained by Poly1305-AES, ChaCha20, Speck and Simon. In Raspberry Pi 3B Simon improved results by 60 times and reached the throughput of 48,861,882 bytes per second.

Less noticeable but bigger changes have happened to algorithms executed in the Zero W model. Can be viewed that AES algorithms had a significant raise by 61 to 74 times compared to the Figure 7. Also Speck achieved 16,718,367 bytes per second compared to previous 291,738, which is on 5630% more. Yet, the most remarkable difference is performed by Poly1305. It gained a throughput of 19,821,852 bytes per second that is 104 times more than before.

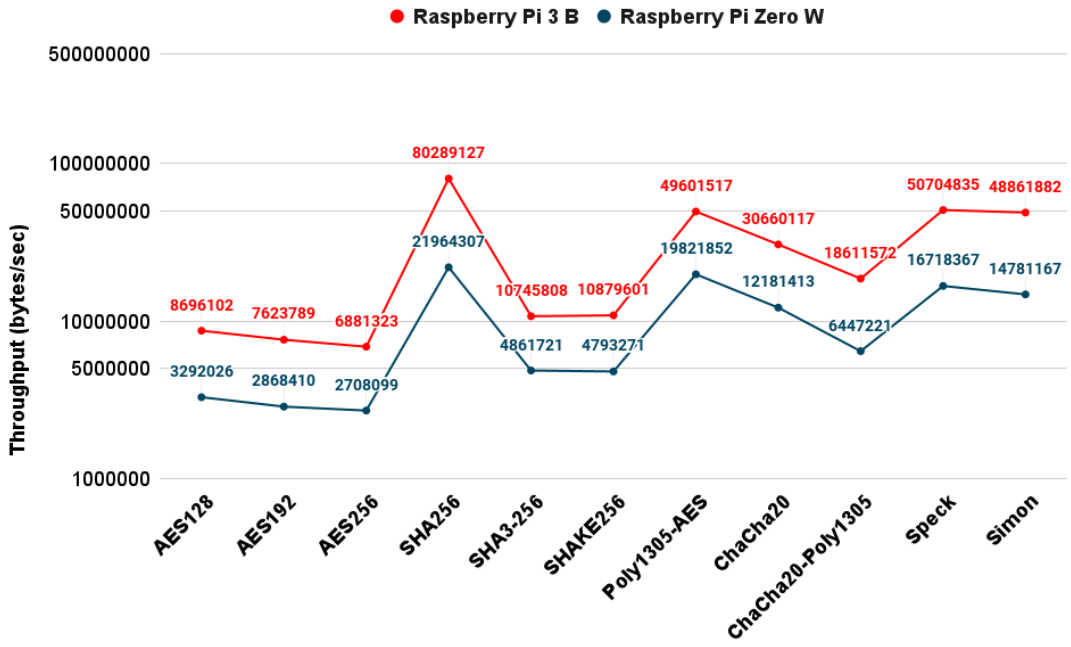

Figure 8: Throughput of 524288 bytes file.

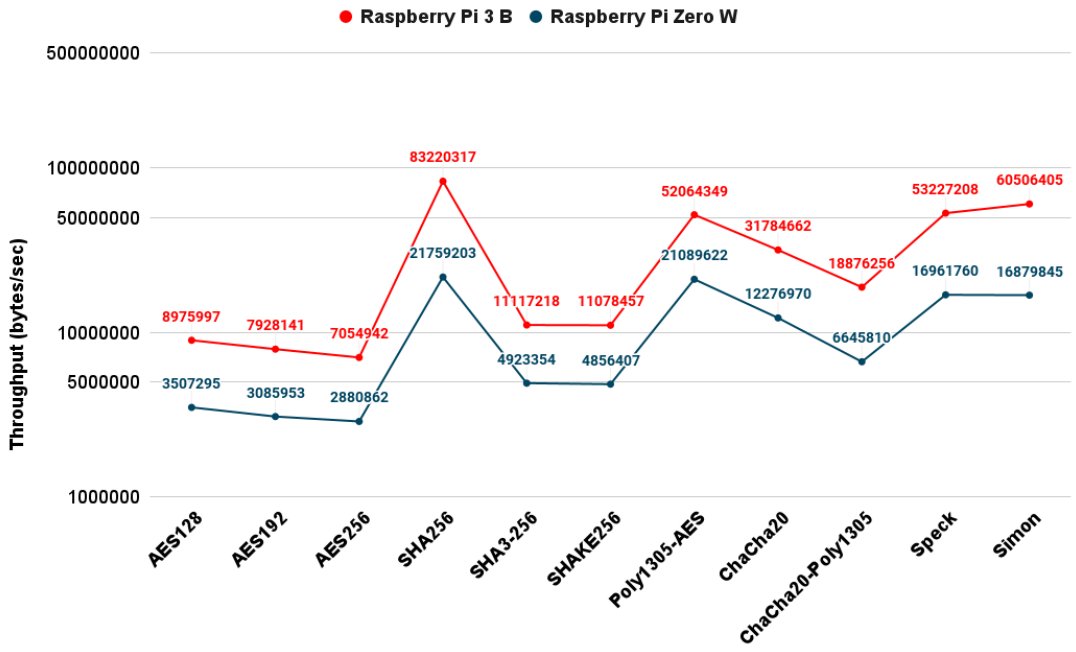

Figure 9: Throughput of 1048576 bytes file.

From the test results, it can be seen that the algorithm's throughput increased in parallel with the file size. Also there are no cardinal changes in connection to the device model as lower performance for the Zero W model is uniform across all cryptographic algorithms. The hash function SHA256 kept showing the best results in every throughput test. AES128, AES192 and AES256 showed similar and comparatively low results along the way. In the test with 1,048,576 bytes file for Raspberry Pi 3 model B, AES128 block cipher reached 8,975,997 bytes per second, while Simon, the lightweight block cipher, gathered 60,506,405 bytes per second. That demonstrates the difference between standard and a lightweight cryptography.

During power consumption measurements indicated that SBCs consumed a similar amount of the power within the Raspberry Pi 3 B model, as Figure 10 depicts. If correlated results between 1 and 1,048,576 bytes file tests, there is no significant raise of the power consumption. An average difference in the power consumption raise is just 3%. The biggest increase in the power consumption is produced by ChaCha20-Poly1305 for 1024 bytes file size. At the time of the encryption process, it consumed on 90% more than in a quiescent state, while an average consumption by other algorithms increased by 82%. The only remarkable outcome is produced by the lightweight Simon block cipher. During the encryption operation of the heavy files, Simon has shown a power increase only by 39-42%, that is a twice lower intake compared to other algorithms.

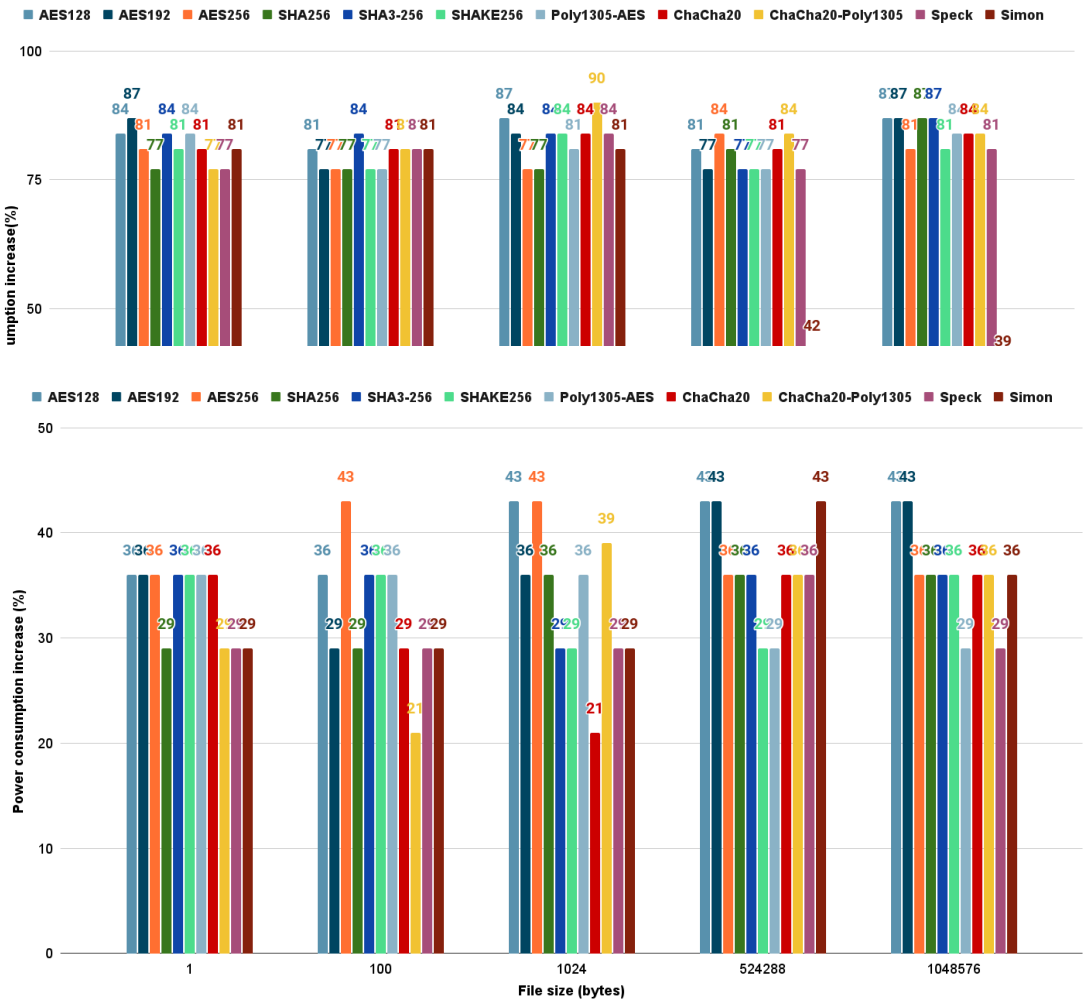

Figure 11: Power consumption increase for Raspberry Pi Zero W.

Encryption in the Zero W model provided more heterogeneous results. Figure 11 demonstrates that almost half of the algorithms had exactly 36% higher consumption

than before encryption, while other varies between 21% and 43%. As for example, AES128 and AES192 required 43% more power during an execution of the heavy algorithms. ChaCha20 and ChaCha20-Poly1305 showed better outcomes for smaller plaintexts and the increase is on 21%. In overall power consumption increase for Zero W is twice lower than for 3 B model.

Figure 12 displays a memory usage by the algorithms. It can be viewed that AES128, AES192, AES256, ChaCha20 and ChaCha20-Poly1305 have output file sizes identical or around to the input size. Lightweight block ciphers Simon and Speck have a small memory usage that is kept between 36 and 39 bytes independently of the input text size. This should be considered as an advantage for the encryption of heavy files within the resource constraint environment. During the result analysis is taken into account that hash algorithms only produce a tag instead of encrypting the whole text, for this reason the output requires less memory. Moreover, SHAKE256's output message length can be changed depending on the requirements as it is an extendable-output function.

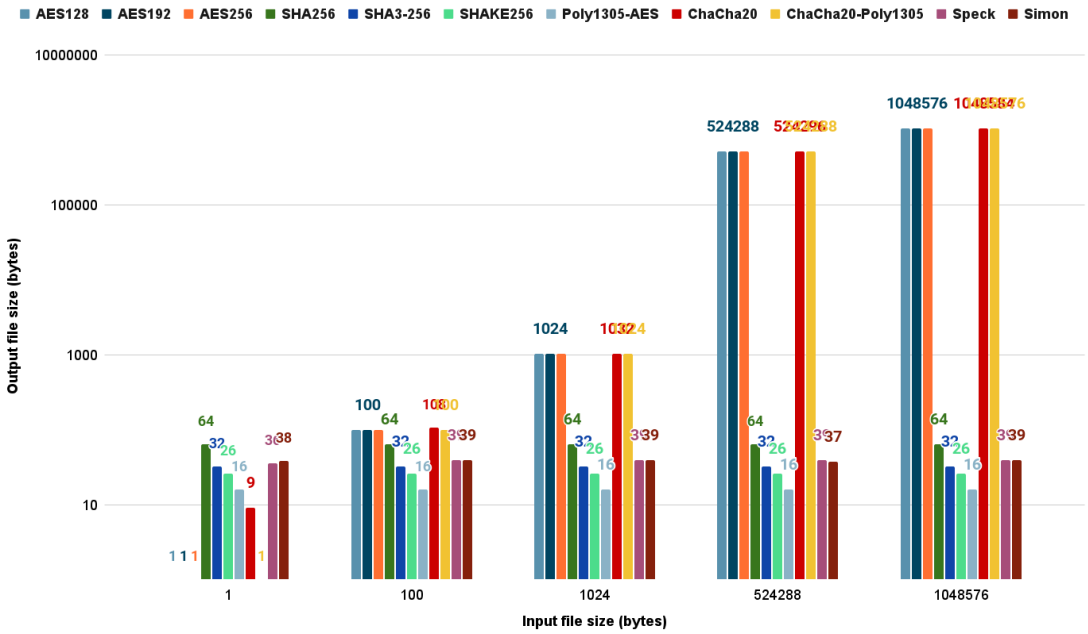

Figure 12: Memory usage by cryptographic algorithms.

To summarize all gathered data such short overviews are made:

AES128, AES192, AES256: In comparison to the lightweight block algorithms, demonstrated significantly lower throughput. The AES algorithms are on average 5 times slower than lightweight algorithms. Though, the power consumption during encryption of small files is similar to other algorithms but twice higher compared to Simon for encrypting large plaintexts. Memory usage is not adapted to the needs of the resource constraint environment, as it outputs same size ciphertexts.

SHA256: Showed the best results of the throughput measurements among all algorithms. There are no deviations from the average power consumption compared to others. Among hash functions SHA256 required the biggest memory space, about twice more than the rest of the hash functions.

SHA3-256 and SHAKE256: Demonstrated analogous test results. Throughput is noticeably smaller than SHA256 had, while power consumption results are slightly better for the large size files.

Poly1305-AES: Performed well for the encryption of the large plaintexts, had an average power consumption and a fixed output smaller than SHA family have.

ChaCha20: For a stream cipher, ChaCha20 showed good throughput and average power consumption results. Produced ciphertext file sizes are slightly bigger than the input, therefore not adapted for the use in a constraint environment.

ChaCha20-Poly1305: Showed better throughput and power consumption results than ChaCha20 and Poly1305 for the small sized plaintexts. As ChaCha20 it produces large ciphertexts.

Speck and Simon: Speck demonstrated improved performance except the cases when an extra large plaintext is provided, in that case Simon is more effective. Same situation is noticed during the power consumption tests, while memory usage is almost identical.

The executed results and analysis provided an overview of the tested algorithms performance to consider their strong and weak sides for the implementation in the resource constraint environments.

6 Conclusion

The IoT is a spreaded and vulnerable technology. Due to its nature, it requires special and timely measures to provide protection since the rapid distribution of this technology and the lack of sufficient protection can lead to serious problems in the future. Whereas cryptography can provide confidentiality, integrity, authentication and non-repudiation to IoT devices. The purpose of this research is to analyze the cryptographic algorithms executed in the constraint environment in order to compare their performance and understand which of the algorithms are best suited for use in a given environment and whether the standard algorithms are suitable for the non-standard environment.

Executed tests showed that Poly1305-AES, ChaCha20 and their combination ChaCha20-Poly1305 are compatible with the constraint environment, yet in cases where memory usage is a critical parameter, ChaCha20 and ChaCha20-Poly1305 might not be the most suitable choice. While SHA2 and SHA3 approved family functions of the are suitable for the implementation in the constraint environment, though SHA256 still can be considered as a preferable choice. Also, the standard AES algorithms are suitable for use in a limited environment, but are far inferior to specially designed lightweight algorithms for this purpose. However, since the standardization process of the lightweight encryption algorithms is not yet complete, the choice arises to use proven and reliable algorithms or more suitable for specific tasks. These facts lead to the conclusion that standardization of algorithms is necessary and requires special attention.

References

[1] Sebastien Ziegler, Internet of Things Security and Data Protection, 2019, pp 3-4.

[2] PiMyLifeUp, 20+ Raspberry Pi IoT Projects, www.pimylifeup.com/category/projects/iot/

[3] Chintan Patel & Nishant Doshi, Internet of Things Security: Challenges, Advances, and Analytics, September 2018, pp. 2.

[4] Udemy Online Courses, www.udemy.com/course/learn-hands-on-iot-with-raspberry-pi-and-slack/

[5] CISCO Networking Academy, www.netacad.com/courses/cybersecurity/iot-security

[6] Umit Isikdag, Enhanced Building Information Models: Using IoT Services and Integration Patterns, May 2015, pp. 44-49.

[7] Brian Russell & Drew Van Duren, Practical Internet of Things Security, June 2016, pp. 1-2.

[8] Statista, www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[9] Tesjasvi Alladi & Vinay Chamola & Biplab Sikdar & Kim-Kwang Raymond Choo, Consumer IoT: Security Vulnerabilities Case Studies and Solutions, October 2019, pp. 1.

[10] L. Mary Shamala & Dr. G. Zayaraz & Dr. K. Vivekanandan, Dr. V. Vijayalakshmi, Lightweight Cryptography Algorithms for Internet of Things enabled Networks: An Overview, 2021, pp 1.

[11] Brian Russell & Drew Van Duren, Practical Internet of Things Security, June 2016, pp. 5, 15.

[12] Chintan Patel & Nishant Doshi, Internet of Things Security: Challenges, Advances, and Analytics, September 2018, pp. 27.

[13] Muhammad Burhan & Rana Asif Rehman & Bilal Khan & Byung-Seo Kim, IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey, August 2018, pp. 6-8.

[14] Chintan Patel & Nishant Doshi, Internet of Things Security: Challenges, Advances, and Analytics, September 2018, pp. 46.

[15] Otmane El Mouaatamid & Mohammed Lahmer & Mostafa Belkasmi, Internet of Things: Layered classification of attacks and possible Countermeasures, September 2016, pp. 29.

[16] Chintan Patel & Nishant Doshi, Internet of Things Security: Challenges, Advances, and Analytics, September 2018, pp. 44-45.

[17] Sebastien Ziegler, Internet of Things Security and Data Protection, 2019, pp 25-30.

[18] Brian Russell & Drew Van Duren, Practical Internet of Things Security, June 2016, pp. 132-133.

[19] Md. Enamul Haque & Sm Zobaed & Muhammad Usama Islam & Faaiza Mohammad Areef, Performance Analysis of Cryptographic Algorithms Selecting Better Utilization on Resource Constraint Devices, December 2018, pp. 2-3.

[20] Alfred J. Menezes & Paul C. van Oorschot & Scott A. Vanstone, Handbook of Applied Cryptography, August 2001, pp. 4-6.

[21] Charalampos Manifavas & George Hatzivasilis & Konstantinos Fysarakis & Konstantinos Rantos. Lightweight Cryptography foe Embedded Systems – A Comparative Analysis, pp. 4.

[22] Diaa Salama Abd Elminaam & Hatem Mohamed Abdual Kader & Mohiy Mohamed Hadhoud, Evaluating The Performance of Symmetric Encryption Algorithms, May 2010, pp. 213.

[23] Charalampos Manifavas & George Hatzivasilis & Konstantinos Fysarakis & Konstantinos Rantos. Lightweight Cryptography foe Embedded Systems – A Comparative Analysis, pp. 2.

[24] Chintan Patel & Nishant Doshi, Internet of Things Security: Challenges, Advances, and Analytics, September 2018, pp. 44.

[25] NIST, Computer Security Resource Center, www.csrc.nist.gov/projects/lightweight-cryptography

[26] Okamura Toshihiko, Lightweight Cryptography Applicable to Various IoT Devices, October 2017, pp. 68-69.

[27] Umit Isikdag, Enhanced Building Information Models: Using IoT Services and Integration Patterns, May 2015, pp. 48.

[28] Marjana Maksimovic & Vladimir Vujovic & Nikola Davidovic & Vladimir Milosevic & Branko Perisic, Raspberry Pi as Internet of Things hardware Performances and Constraints, June 2014, pp. 6.

[29] Raspberry Pi, Buy a Raspberry Pi 3 Model B, www.raspberrypi.org/products/raspberry-pi-3-model-b/

[30] Raspberry Pi, Buy a Raspberry Pi Zero W, www.raspberrypi.org/products/raspberry-pi-zero-w/

[31] Raspberry Pi Documentation, Power Supply, www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md

[32] Raspberry Pi, Operating system images, www.raspberrypi.org/software/operating-systems/

[33] International Engineering Task Force, ChaCha20 and Poly1305 for IETF Protocols, May 2015, pp. 19-25.

[34] Dobre Blazhevski & Adrijan Bozhinovski & Biljana Stojchevska & Veno Pachovski, Modes of Operation of the AES Algorithm, 2013, pp.

[35] IBM Documentation, Transport Layer Security – TLSv1.2, www.ibm.com/docs/en/zvse/6.2?topic=openssl-transport-layer-security-tlsv12

[36] Nigel Smart, Cryptography: An Introduction (3rd Edition), April 2013, pp. 156.

[37] Federal Information Processing Standards Publication, Secure Hash Standard, August 2015, pp. iv-21.

[38] NIST, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015, pp. 1-2, 22.

[39] International Engineering Task Force, ChaCha20 and Poly1305 for IETF Protocols, May 2015, pp. 13.

[40] Daniel J. Bernstein, The Poly1305-AES message-authentication code, March 2015, pp. 1-4.

[41] Daniel J. Bernstein, ChaCha, a variant of Salsa20, January 2008, pp. 1.

[42] National Security Agency, Simon and Speck: Block Ciphers for the Internet of Things, July 2015, pp. 1-5.

[43] Gordon Procter, A security Analysis of the Composition of ChaCha20 and Poly1305, pp. 2.

[44] International Engineering Task Force, ChaCha20-Poly1305 Cipher Suites for Transport Layer Security, June 2016, pp. 3.

[45] International Engineering Task Force, ChaCha20 and Poly1305 for IETF Protocols, May 2015, pp. 19-25.

[46] National Security Agency, Simon and Speck: Block Ciphers for the Internet of Things, July 2015, pp. 1-3.

[47] Simon and Speck Information Paper, www.nsacyber.github.io/simon-speck/papers/SimonandSpeckInfo-ISO-SC27-WG2.pdf

[48] Simon and Speck New Publication Update, www.nsacyber.github.io/simon-speck/papers/SimonandSpeckInfoPaperFINAL.pdf

[49] Raspberry Pi OS, www.raspberrypi.org/software/

[50] Pycryptodome 3.9.9 documentation, www.pycryptodome.org/en/latest/src/introduction.html

[51] PyPi, RFC7539, www.pypi.org/project/rfc7539/

[52] The Python Standard Library, www.docs.python.org/3/library/time.html

[53] Source code: www.geeksforgeeks.org/how-to-open-and-close-a-file-in-python/

[54] The Python Standard Library, www.docs.python.org/3/library/os.html?highlight=urandom#os.urandom

[55] Lucy Hattersley, Get Started with Raspberry Pi, Novermber 2019, pp. 20.

[56] Source code: www.dlitz.net/software/pycrypto/api/current/Crypto.Cipher.AES-module.html

[57] Source code: www.pythonpool.com/python-sha256/

[58] Source code: www.pycryptodome.readthedocs.io/en/latest/src/hash/sha3_256.html?highlight=sha3-256

[59] Source code: www.pycryptodome.readthedocs.io/en/latest/src/hash/shake256.html?highlight=shake128

[60] Source code www.pycryptodome.readthedocs.io/en/latest/src/hash/poly1305.html?highlight=poly1305

[61] Source code: www.pycryptodome.readthedocs.io/en/latest/src/cipher/chacha20.html?highlight=chacha20

[62] Source code: www.github.com/AntonKueltz/rfc7539

[63] Source code: www.asecuritysite.com/encryption/speck

[64] Source code: www.pypi.org/project/simonspeckciphers/

List of Figures

List of Tables

Appendix 1 – Cryptographic Algorithm's Code

```
from Crypto.Cipher import AES
from Crypto import Random
from os import urandom

key = urandom(16)
iv = Random.new().read(AES.block_size)
cipher = AES.new(key, AES.MODE_CFB, iv)
ciphertext = iv + cipher.encrypt(plaintext)
```

Figure 13. AES128 [56].

```
from Crypto.Cipher import AES
from Crypto import Random
from os import urandom

key = urandom(24)
iv = Random.new().read(AES.block_size)
cipher = AES.new(key, AES.MODE_CFB, iv)
ciphertext = iv + cipher.encrypt(plaintext)
```

Figure 14. AES192 [56].

```
from Crypto.Cipher import AES
from Crypto import Random
from os import urandom

key = urandom(32)
iv = Random.new().read(AES.block_size)
cipher = AES.new(key, AES.MODE_CFB, iv)
ciphertext = iv + cipher.encrypt(plaintext)
```

Figure 15. AES256 [56].

```
import hashlib

hash = hashlib.sha256(plaintext).hexdigest()
byte_hash = hash.encode('utf-8')
```

Figure 16. SHA256 [57].

```
from Crypto.Hash import SHA3_256

hash = SHA3_256.new()
hash.update(plaintext)
```

Figure 17. SHA3-256 [58].

```
from Crypto.Hash import SHAKE256

shake = SHAKE256.new()
shake.update(plaintext)
```

Figure 18. SHAKE256 [59].

```
from Crypto.Hash import Poly1305
from Crypto.Cipher import AES
from os import urandom

key = urandom(16)
binary_tag = Poly1305.new(key=key, cipher=AES,
data=plaintext).digest()
```

Figure 19. Poly1305-AES [60].

```
from base64 import b64encode
from Crypto.Cipher import ChaCha20
from os import urandom

key = urandom(32)
cipher = ChaCha20.new(key=key)
ciphertext = cipher.encrypt(plaintext)
nonce = b64encode(cipher.nonce).decode('utf-8')
result = b64encode(ciphertext).decode('utf-8')
```

Figure 20. ChaCha20 [61].

```
from rfc7539 import aead
from os import urandom

key = urandom(32)
nonce = b'thisisanonce'
additional_data = b'Some additional data'
ciphertext, mac = aead.encrypt_and_tag(key, nonce,
plaintext, additional_data)
```

Figure 21. ChaCha20-Poly1305 [62].

```
import speck
import binascii
import sys

def get_binary(word):
    return int(binascii.hexlify(word), 16)

k = '0x1b1a1918131211100b0a090803020100'

if len(sys.argv) > 1:
    message = str(sys.argv[1])
    m = get_binary(plaintext)

if len(sys.argv) > 2:
    k = str(sys.argv[2])

key = int(k, 16)
key_size = (len(k) - 2) * 4

if key_size == 64: bsize = 32
if key_size == 72: bsize = 48
if key_size == 96: bsize = 48
if key_size == 128: bsize = 128

cipher = speck.SpeckCipher(key, key_size=key_size,
block_size=bsize)
ciphertext =
cipher.encrypt(int.from_bytes(plaintext,
byteorder='big'))
```

Figure 22. Speck [63].

```
from simon import SimonCipher

int_message = int.from_bytes(plaintext, "big")
simon =
SimonCipher(0xABBAABBAABBAABBAABBAABBAABBAABBA,
key_size=128, block_size=128)
ciphertext = simon.encrypt(int_plaintext)
```

Figure 23. Simon [64].

Appendix 2 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Yulia Derbeneva

1 Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Comparative analysis of encryption algorithms for resource constraint environment”, supervised by Tauseef Ahmed

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2 I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3 I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

17.05.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.