Analyzing the Efficiency of Lightweight Symmetric Ciphers on IoT Devices

Jericho Rivero, Tran Ngoc Bao Huynh, Angelica Smith-Evans, Ong Thao, Yuan Cheng

California State University, Sacramento

Abstract

Modern Internet-of-Things (IoT) devices need to have their sensitive data encrypted, or they will be easily compromised. These resource-constrained IoT devices would need cryptographic algorithms that are both lightweight and secure. However, most conventional ciphers are not designed to run in a resource-constrained setting, making them inadequate for IoT devices. This paper analyzes the performance of several lightweight ciphers on a Raspberry Pi 4 Model B to reveal which of them is the most efficient on a resource-constrained IoT device. We gather the performance data for each algorithm through four metrics: average execution time for encryption and decryption, throughput, power consumption, and memory usage. We find that PRESENT and CHACHA20 are great options for a resource-constrained IoT device amongst our chosen ciphers.

1. Introduction

Internet of Things (IoT) refers to a vast number of interconnected physical devices that collect rich data and enable rapid communication with each other. Encrypting sensitive data on IoT devices is essential to ensure the confidentiality and integrity of the data. However, a hurdle for deploying cryptographic algorithms on these devices is resource constraint. IoT devices are usually built with mini processors, compact memory and storage, and low power capacity. Most conventional cryptographic algorithms do not cater for the limitations posed by the resource-constrained IoT devices. As a result, several lightweight ciphers have been proposed to accommodate the need.

In this paper, we pick several popular lightweight block and stream ciphers and design an experiment to analyze their performance on a Raspberry Pi 4 Model B. The performance of these chosen ciphers is evaluated in terms of four metrics, average execution time, throughput, power consumption, and memory usage. We then discuss the results from the experiment and provide our perspective on which ciphers are most suitable for IoT devices. We hope our findings can help the IoT community identify a good cipher that ensures data security while not posing significant overhead on a resource-constrained host device.

The remainder of the paper is organized as follows. Section 2 introduces the motivation and the cryptographic algorithms we examined. In Section 3, we describe the setup for our experiment and the metrics we used to evaluate the chosen ciphers. Section 4 presents the results of the experiment. Finally, Section 5 concludes the paper.

2. Background

In this section, we will describe each of the lightweight cryptography algorithms used in the experimentation.

2.1. Lightweight Cryptography

Lightweight cryptography is a category of cryptographic primitives designed for resource-constrained devices, such as embedded systems, RFID (Radio-frequency identification) devices, and sensor networks [1]. Compared to conventional cryptographic algorithms, lightweight cryptography has a smaller footprint [2].

Security is not a trade-off in lightweight cryptography. Although constrained by resource, lightweight cryptography still strives for a good balance between security, performance, and resource requirements [1].

Most lightweight cryptographic algorithms we have seen belong to symmetric key cryptography since public key cryptography usually requires larger resource than its symmetric key counterpart. However, this does not mean that there is no lightweight algorithm based on public key cryptography. Actually, elliptic-curve cryptography (ECC) is considered an attractive option for resource-constrained environments due to its relatively lower computational requirements [3].

2.2. Symmetric Key Cryptography

In this work, we focus on lightweight symmetric key cryptography. In particular, we are interested in the efficiency of block ciphers and stream ciphers. Below, we briefly summarize the background information of the block ciphers and stream ciphers we studied.

2.2.1. Block Ciphers

AES [4] is a block cipher with a block size of 128 bits and three key lengths: 128-bit, 192-bit, and 256-bit. The key length also determines the number of transformation rounds. It runs 10, 12, and 14 rounds for 128-bit keys, 192-bit keys, and 256-bit keys, respectively.

DES [5] is a block cipher that was approved as a Federal Information Processing Standard (FIPS) in 1977. DES uses a 56-bit key, which has been proved to be inadequate. 3DES is a more secure version of DES, which extends the key size by applying the DES algorithm three times (168-bit).

CLEFIA [6] is a lightweight block cipher striving for high efficiency in both software and hardware. The 128-bit block cipher also supports key lengths of 128-bit, 192-bit, and 256-bit, making it compatible with AES. In this project, we use a CLEFIA implementation with a 128-bit key.

PRESENT [7] is a lightweight block cipher with design goals of both security and hardware efficiency. The key length is either an 80-bit or 128-bit key. The block size is 64 bits. This project utilizes a PRESENT implementation with a 128-bit key.

2.2.2. Stream Ciphers

CHACHA20 [8] is a stream cipher with a 256-bit key and a 96-bit initialization vector (IV). It is a variant of the Salsa20 family of stream ciphers [9].

GRAIN [10] is a stream cipher submitted to ECRYPT II eSTREAM project [11]. GRAIN is designed for resource-constrained devices, especially for devices with limited gate counts, power, and memory. GRAIN has an 80-bit key and a 64-bit IV.

MICKEY [12] is also a stream cipher submitted to the eSTREAM project portfolio. With low hardware complexity and high security, the MICKEY cipher aims to handle hardware platforms that are also strained in resources. MICKEY has an 80-bit key and an IV of 0 to 80 bits in length.

3. Methodology

In this section, we will describe the methodology of this study. We will also discuss the relevance of the data sets chosen to be analyzed.

3.1. Software and Hardware

We utilized the OpenSSL [13] library (version 1.1.1) to run different modes of operation of AES and 3DES ciphers with a Bash script. Meanwhile, for the ciphers not supported by the OpenSSL library, including CHACHA20, CLEFIA, PRESENT, GRAIN, and MICKEY, we took advantage of the implementations of these ciphers from the Internet. Additional software interfacing was created using the C programming language. All tests were run on Raspberry Pi OS version 4.1 [14], which is based on Debian optimized for the Raspberry Pi.

We chose Raspberry Pi 4 Model B as the testbed since it is the most powerful IoT device available for its price point. Released in 2019, this model utilizes a Quad-core 64-bit ARM-Cortex A72 CPU @ 1.5 GHz. It has 4 GB of memory available. A Klein Tools Digital USB multimeter was used to measure the voltage and current in the experiment.

3.2. Performance Metrics

In this project, we aim to figure out the trade-off between the performance of lightweight ciphers and the resource constraints posed on IoT devices. We are particularly interested in the following metrics.

3.2.1. Execution Time and Throughput

The average execution time of each cipher was measured by calculating the time it takes to fully encrypt and decrypt a random set of files of different sizes. Throughput is the rate at which encryption and decryption outputs are produced. This is another way of evaluating the efficiency of the chosen ciphers.

The random set of files were created by using a Linux library called `urandom`. The file sizes ranged from 150 MB to 1 GB for OpenSSL ciphers and 1 MB for lightweight algorithms. After the data set was generated, we ran encryption and decryption on each file in the set.

3.2.2. Power Consumption

We ran each cipher 20 times and took the average readings to determine power consumption. Running the code 20 times in a row on one process allows us to record the power consumption by hand with our tool-set. Power (P) was deduced implicitly by recording voltage (V) and current (I), then using the power equation $P = V \times I$.

3.2.3. Memory

We used the `getrusage()` function to record the resource usage of the encryption and decryption processes. We called the function with $RUSAGE\_SELF$ and ru_maxrss (max resident set size) to get the peak RAM used before encryption or decryption. After the process was done, we called the function again and calculated the difference. The result (in kilobytes) is the memory usage of the process.

4. Experimental Results

This section will describe the results of our experiment on the chosen ciphers.

4.1. Different Modes of Operation in AES and 3DES

We wanted to identify the best mode of conventional ciphers to compare with other lightweight ciphers. Therefore, we first ran a test with AES and 3DES, both of which are supported by OpenSSL. From Table 1 and Figure 2, we found that AES-CTR-128 had the best performance for both large and small data sets, compared to other modes of operation. Meanwhile, 3DES (i.e., DES EDE3) had the longest run time and consumed the most power among all the modes.

The overall shape of our results for encryption and decryption of AES and 3DES are roughly the same. Therefore, for simplicity, we decided to only show the encryption results as a representative.

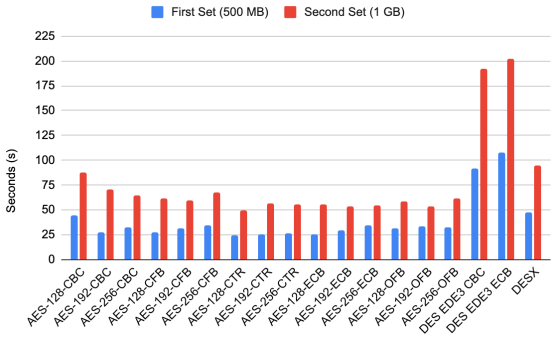

Figure 1: Average Encryption Time of AES and DES

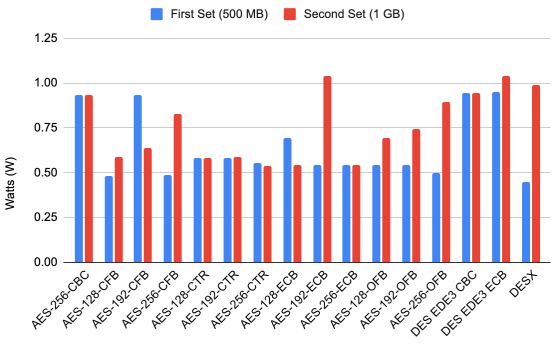

Figure 2: Power Consumption of AES and DES

4.2. Block Ciphers

Out of the three block ciphers that we tested, the PRESENT cipher outperformed the others in terms of execution time and power usage, as shown in Figure 3 and Figure 4. Our data showed that the PRESENT cipher is at least 73 times faster than the other two. It also exhibited less power consumption in comparison to the other two ciphers.

In our experiment, AES was the second place in terms of run time and power usage. Meanwhile, CLEFIA performed the slowest and used the most power out of the bunch. While this may be the case for our experiment, CLEFIA has a much higher throughput and higher efficiency in hardware implementations than AES, as seen in its original publication [6]. The poor performance that CLEFIA exhibited in our experiment could be attributed to the code we used was not tailored and optimized for the hardware we chose.

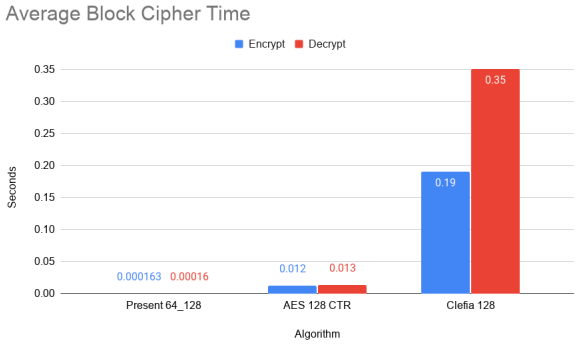

Figure 3: Average Run Time for Lightweight Block Cipher

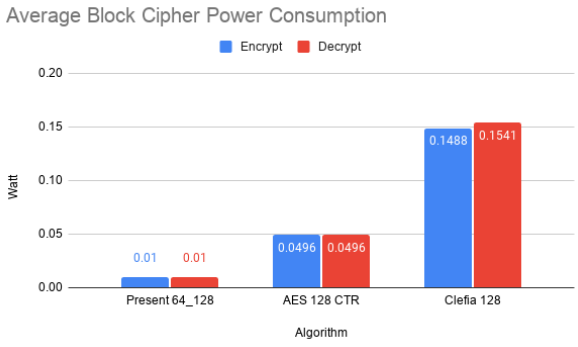

Figure 4: Average Power Consumption for Lightweight Block Ciphers

4.3. Stream Ciphers

From the results shown in Figure 5 and Figure 6, we noticed that CHACHA20 was the fastest and consumed the

Table 1: **Throughput of Ciphers (MB/sec)**

Data Size (MB)	50MB	150MB	500MB	750MB	1000MB
AES CBC 128	17.593	12.202	11.211	11.206	11.401
AES CBC 192	11.338	18.242	18.032	17.660	14.078
AES CBC 256	10.498	15.249	15.474	14.882	15.546
AES CFB 128	18.854	18.046	18.045	17.949	16.223
AES CFB 192	11.513	13.622	15.888	16.612	16.861
AES CFB 256	10.790	14.184	14.582	17.165	14.804
AES CTR 128	19.976	19.641	20.042	22.502	20.086
AES CTR 192	16.010	17.359	19.261	18.395	17.790
AES CTR 256	19.113	18.682	18.970	17.846	17.978
AES ECB 128	15.518	17.313	19.292	19.354	18.031
AES ECB 192	17.247	18.206	16.918	18.678	18.617
AES ECB 256	16.324	17.401	14.357	18.015	18.241
AES OFB 128	16.223	9.834	15.702	15.443	17.020
AES OFB 192	16.863	14.703	14.822	18.586	18.499
AES OFB 256	15.778	15.895	15.585	14.626	16.287
DES EDE3 CBC	4.349	5.667	5.464	5.290	5.203
DES EDE3 ECB	4.810	4.222	4.640	5.106	4.938
DESX	10.607	10.476	10.552	11.217	10.530

Table 2: **Performance of Lightweight Stream Cipher - 1 MB message size**

Stream Cipher	Power Consumption (W)		Throughput (MB/s)		Memory Usage (KB)	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
CHACHA20	2.535	2.584	6.605	6.485	909	1049
GRAIN 80	3.231	3.274	0.053	0.065	898	1048
GRAIN 128	3.224	3.274	0.047	0.062	912	1056
MICKEY 80	2.678	2.684	0.513	0.526	912	1052
MICKEY 128	2.884	2.684	0.435	0.423	908	1052

least power during both encryption and decryption, compared to GRAIN and MICKEY. The run time of GRAIN in this experiment, however, was unexpectedly high. It contradicts the findings in [10], where both GRAIN-80 and GRAIN-128 outperformed MICKEY-80 and MICKEY-128. Nonetheless, we observed the opposite result. Aside from GRAIN's high run time and power consumption, Table 2 shows that GRAIN has a low throughput. We also saw that the memory usage of GRAIN is lower than both CHACHA20 and MICKEY from Table 1. We found the following statement about GRAIN from the eSTREAM Project website, "Like many stream ciphers, there is some cost incurred during initialisation and the impact of this will depend on the intended application and the likely size of the messages being encrypted." We believed the size of data (we encrypted/decrypted 1 MB of data each test) and inappropriate initialization environment setup for the cipher might be the main reasons for this poor result.

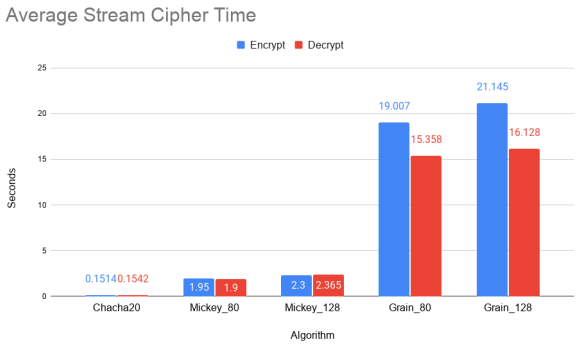

Figure 5: Average Run Time of Lightweight Stream Cipher

4.4. Limitations

Due to security and copyright issues, some of the cipher implementations we used are only for education. Therefore, they are not optimized. We also did not consider

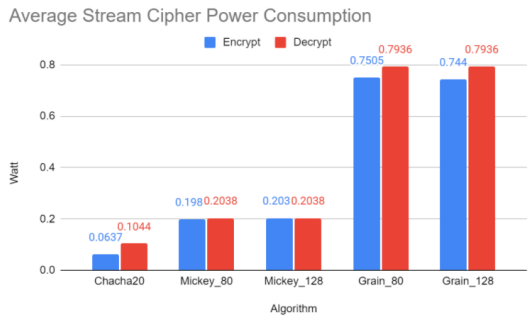

Figure 6: Power Consumption of Lightweight Stream Cipher

the underlying hardware into the scope. Some of the ciphers may have a better performance with certain hardware implementations. However, we could not set up the environment and perform an analysis accordingly. Finally, our project scope was changed with our own growth and understanding of IoT devices and lightweight ciphers. For example, we could not include some IoT sensors and modules in our experiment, such as microphone sensors, ZYMKEY security module for Raspberry Pi, etc.

5. Conclusion

In this paper, we designed an experiment to analyze the efficiency of several lightweight symmetric ciphers on a Raspberry Pi device. Efficiency was evaluated based on four criteria: encryption and decryption time, throughput, memory usage, and power consumption. The experiment showed that PRESENT (128-bit key) and CHACHA20 seem to be the clear winner in the categories of block ciphers and stream ciphers, respectively, regarding the overall efficiency on the Raspberry Pi 4 Model B. We believe that with more optimization on both software and hardware implementations, the performance of lightweight ciphers in this experiment could be even better.

Acknowledgment

We would like to express our deep gratitude to Dr. Ted Krovetz for his wisdom and insight.

References

[1] McKay K, Bassham L, Sönmez Turan M, Mouha N. Report on lightweight cryptography. Technical Report NISTIR 8114, National Institute of Standards and Technology, 2017.

[2] Katagi M, Moriai S, et al. Lightweight cryptography for the internet of things. Sony Corporation 2008;2008:7–10.

[3] Eisenbarth T, Kumar S, Paar C, Poschmann A, Uhsadel L. A survey of lightweight-cryptography implementations. IEEE Design Test of Computers 2007;24(6):522–533.

[4] Rijmen V, Daemen J. Advanced encryption standard. Proceedings of Federal Information Processing Standards Publications National Institute of Standards and Technology 2001;19–22.

[5] Davis R. The data encryption standard in perspective. IEEE Communications Society Magazine 1978;16(6):5–9.

[6] Shirai T, Shibutani K, Akishita T, Moriai S, Iwata T. The 128-bit blockcipher CLEFIA. In International workshop on fast software encryption. Springer, 2007; 181–195.

[7] Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A, Robshaw MJ, Seurin Y, Vikkelsoe C. PRESENT: An ultra-lightweight block cipher. In International workshop on cryptographic hardware and embedded systems. Springer, 2007; 450–466.

[8] Bernstein DJ. ChaCha, a variant of Salsa20. In Workshop Record of SASC, volume 8. 2008; 3–5.

[9] Bernstein DJ. The Salsa20 family of stream ciphers. In New stream cipher designs. Springer, 2008; 84–97.

[10] Hell M, Johansson T, Meier W. Grain: a stream cipher for constrained environments. International journal of wireless and mobile computing 2007;2(1):86–93.

[11] Robshaw M. The eSTREAM project. In New Stream Cipher Designs. Springer, 2008; 1–6.

[12] Babbage S, Dodd M. The MICKEY stream ciphers. In New Stream Cipher Designs. Springer, 2008; 191–209.

[13] OpenSSL. `https://www.openssl.org/`. Accessed: 2021-1-17.

[14] Raspberry Pi OS. `https://www.raspberrypi.org/software/`. Accessed: 2021-1-17.