# Power consumption analysis of cryptographic algorithms

Wonseok Choi[1*], Laihyuk Park[2], Youngjin Kim[1] and Junsi Jeong[1]

[1] Telecommunications Technology Association
{Wschoi, networker, jless}@tta.or.kr
[2] Dept of Computer Science and Engineering, Seoul National University of Science and Technology
lhpark@seoultech.ac.kr

**Abstract**

In this paper, we tested popular encryption algorithms' characteristics and optimization efforts and measured their power consumption and performance according to the platform. The aim is to determine the energy efficiency of encryption and decryption for each encryption algorithm and contribute to improving energy efficiency in large data centers and other areas where encryption is heavily used. The test results show that optimizations using the CPU's instruction set significantly improve encryption and decryption time but not energy efficiency. In particular, using NEON instructions on ARM-based platforms resulted in performance gains but increased energy consumption. As the amount of data transferred increases with the spread of 5G and the cloud, additional research should continue to be conducted to improve the energy efficiency of cryptographic algorithms.

**Keywords:** Energy-efficient cryptographic algorithms, Power consumption of cryptography, Energy-efficient computing

# 1 Introduction

The growth of non-face-to-face industries due to COVID-19 and the expansion of various applications such as AI and cloud have led to a rapid increase in data centers, leading to a rapid increase in power consumption. The top 20 U.S. companies owned 597 massive data centers in 2020, double the number five years earlier. According to the IEA, data centers worldwide consumed between 200 and 250TWh of electricity in 2020, higher than the world's 16th largest consumer, South Africa (208TWh), and estimated to consume about 1% of the world's electricity. This trend is expected to overgrow, and to address the problem, Microsoft, Google, Meta, Intel, and others have formed the Open Compute Project (OCP) to explore open, energy-efficient computing. In particular, with the advent of 5G, encryption is becoming more critical for learning large amounts of data through AI, and the need to analyze the amount of power consumed in encryption operations and make efforts to improve it is increasing.

Various encryption algorithms have been developed, and most existing ones have been developed by focusing on encryption speed and security strength. The power consumption of cryptographic algorithms has not been a concern. Still, as mentioned above, countries that have identified the

The 7th International Conference on Mobile Internet Security (MobiSec'23), Dec. 19- 21, 2023, Okinawa, Japan, Article No. 42
*Corresponding author: Telecommunications Technology Association, Gyeonggi-do, 13591, Republic of Korea, Tel: +82-031-724-0114

seriousness of computing power consumption in data centers are beginning to research and develop hardware and software to improve power consumption. In line with this trend, it is necessary to develop encryption algorithms suitable for various mobile devices with low power consumption.
Symmetric-key encryption algorithms currently in use are AES, ARIA, and SEED, while public-key encryption methods include RSA and ECC.
Since this paper aims to analyze the power consumption of encryption algorithms, we selected an algorithm developed for lightweight encryption. The Adiantum encryption algorithm is the most recent algorithm developed by Google, and ChaCha20 and CHAM are all algorithms designed to speed up and lighten existing encryption algorithms.
In this paper, we further analyzed the power consumption of the AES encryption algorithm to compare it with these lightweight encryption algorithms. The purpose of this paper is to analyze the power consumption of each encryption algorithm and suggest directions for the development of low-power encryption algorithms.
This paper is organized as follows. First, we discuss the structure and characteristics of each cryptographic algorithm. We then identify techniques for speeding up each cryptographic algorithm, particularly how to increase processing speed by using processor-specific vector arithmetic instructions. We then describe the test environment and test methods for each cryptographic algorithm. In this paper, encryption algorithms implemented using processor-specific optimized instructions were used, and the time required for encryption and decryption of each encryption algorithm was measured by repeated testing. The power consumed is measured using a power analyzer, and the amount of power consumed by each algorithm for encryption and decryption is calculated. Finally, the test results show the amount of power consumed per byte for encryption and decryption and suggest future research directions.

# 2 Related Work

In this chapter, we will review the characteristics of cryptographic algorithms and the related research that has been done to make them lighter and faster. Low-power cryptographic algorithms have not been a focus of traditional cryptography researchers, so little research has been done on them. However, energy-efficient cryptographic algorithms are very relevant to light-weighting, so that we will discuss them.

## 2.1 AES (Advanced Encryption Standard)

AES (Advanced Encryption Standard) is currently the most widely used encryption algorithm, developed to replace DES. A symmetric key algorithm uses the same key for encryption and decryption. AES is a symmetric key algorithm that uses the same key for encryption and decryption. It has a free key size of 128bit, 192bit, and 256bit and has an SPN (Substitution - Permutation Network) structure. The SPN structure requires an inverse function in the encryption process, but it can be encrypted at once without moving bits so that it can perform encryption operations efficiently [1]. AES stores 16 input bytes in a 4x4 matrix (state) in 16 one-byte units. A round of AES consists of four operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The SubBytes step organizes the message into a square matrix and replaces each byte in the array using the S-box opesration. ShiftRows in AES are done in bytes, unlike DES, which is done in bits. MixColumns performs a column-by-column function, grouping four bytes into a 4X1 matrix and multiplying it with a 4x4 identity matrix to create a new matrix. The multiplication operation is computed in GF ($2^8$). AddRoundKey is the process of performing a subkey and XOR operation on each byte in the matrix during the round. The above four steps are repeated 10 to 14 times per round, depending on the number of bits, to make AES work. The operating modes are CBC, OFB, CFB, and GCM, with CBC and GCM modes being the most popular as they allow for parallel processing.

The AES encryption algorithm's most widely known optimization method is the look-up table-based method proposed by Bertoni et al. [2]. In their paper, Bertoni et al. combine the AES algorithm's SubBytes, ShiftRows, and MixColumns and generate a precomputed look-up table to apply to round operations. While the existing AES algorithm requires 256 bytes of S-Box memory, the optimized method requires four 32-bit tables with 256 entries, which means 4,096 bytes of memory space. In addition, Bernstein et al. proposed a technique to speed up the algorithm by storing and reusing some of the information in the operation using the characteristics of the CTR mode of the AES algorithm.

## 2.2 ChaCha20

ChaCha20 is an encryption algorithm that improves the existing Stream encryption algorithm Salsa2008 to increase the spread per round. It has the same 128-bit constant, 256-bit key, 64-bit counter, and 64-bit nonce as Salsa2008 and has the advantage of being able to encrypt and decrypt at high speed based on ARX (add rotate XOR) and XOR.

ChaCha, a popular symmetric-key cryptographic algorithm, has been developed and optimized for various platforms. On X86 platforms, 128-bit vectorization was first applied to speed up encryption rounds, as ChaCha's four-round operations can be processed independently, allowing them to be processed simultaneously via 128-bit vectors. With the development of processors, the AVX2 instruction, which can process 256-bit vector operations at once, appeared, and M. Goll and S. Gueron discovered that two block functions using 128-bit vectors could be processed simultaneously using this instruction. Later, the AVX512 was introduced to handle 512-bit vector operations, allowing four block functions to be processed simultaneously, making encryption more efficient [3][4].

## 2.3 CHAM

The CHAM cryptographic algorithm was developed in Korea and proposed in 2017 (revised CHAM submitted in 2019) through the Lightweight Cryptographic Algorithm Competition. CHAM is a block cipher algorithm with a four-branch Feistel structure and supports three modes depending on the block size and key size. The supported modes are shown in the table below. The CHAM encryption algorithm was developed to be lightweight, and, significantly, it uses a stateless round-key technique that does not store the state of the key, dramatically reducing storage space. It is also based on ARX operations, which has the advantage of applying to low-performance processors. Seo et al. proposed an optimization technique based on ARM-NEON processors' SIMD instructions (8x16 vector instructions) to perform operations in parallel [5]. Song et al. obtained a performance improvement of 15.87% in CHAM 64/128 CTR mode by optimizing based on ARMv8 processors [6].

| Cipher | Plaintext Length | Key Length | Round | Word Size |
|---|---|---|---|---|
| CHAM-64/128 | 64bit | 128bit | 88 | 16bit |
| CHAM-128/128 | 128bit | 128bit | 112 | 32bit |
| CHAM-128/256 | 128bit | 256bit | 120 | 32bit |

**Table 1:** CHAM cipher mode

## 2.4 Adiantum

Google requires Android device manufacturers to make their devices capable of AES encryption. Still, the reality is that AES encryption performance is very slow for low-end mobile devices and connected devices, so this algorithm was developed to solve the problem. Block ciphers were designed to enable parallel operations based on the XChaCha12 algorithm, minimizing the space for various nonces and MACs and simplifying the rounds. This allows it to perform more than five times better on ARMv7 devices than the AES-256-XTS encryption algorithm [7].

# 3 Evaluation

This chapter describes the test platforms, test environment, and test procedures. To analyze the power consumption of encryption algorithms, experiments were conducted on the X86 platform and Raspberry Pi 4 for embedded use. The specific test environments are shown in the table below.

| Platform | Item | Specification |
|---|---|---|
| X86 Platform | CPU | Intel i7-1165G7 |
| | Memory | LPDDR4 16GB |
| | SSD | 1TB (NVMe 1.4) |
| | OS | Ubuntu 20.04 64bit |
| | Kernel-Version | 6.5.4 |
| Embedded Platform | CPU | BCM2711 Quad-core 1.5GHz Arm Cortex-A72 |
| | Memory | 8GB |
| | OS | Debian 11 bullseye 64bit |
| | Kernel-Version | 6.1 |

**Table 2:** Test Platforms Specification

The power analyzer used to measure the power consumption is the WT332 model from YOKOGAWA. This model is a high-precision power analyzer corresponding to IEC 62301, with 5 ㎃ accuracy and a measurement resolution 10 ㎼. The details are shown in the table below.

| Item | Specification |
|---|---|
| Measurement accuracy | 0.1% + 0.05% of Range |
| Frequency Bandwidth | DC 0.1 ㎑ to 100 ㎑ |
| Sampling Rate | 100ks/s |
| Data Update Rate | 100 ㎳, 250 ㎳, 500 ㎳ |

**Table 3:** Power Analyzer Specification

The figure below shows the overall test environment configuration. The X86 Platform and Arm Platform were powered by the power analyzer to run the encryption algorithm. A separate test PC was connected to the power analyzer via USB to monitor the power consumption while the encryption algorithm was running, and the usage was recorded every 100 ㎳.
In addition, to calculate the net power consumption of the encryption algorithm, we measured the idle power for 10 minutes before the test, averaged it, and subtracted it from the estimated power. The CPU was set to the maximum frequency using the CPU Governor command, as low power mode can affect the power consumption calculation.

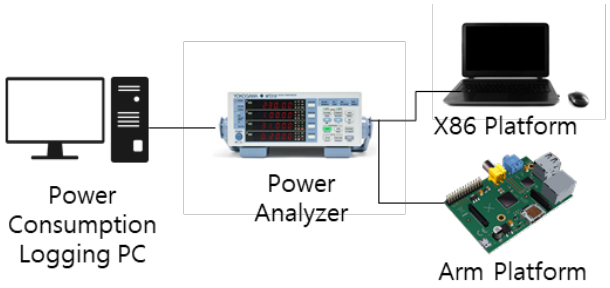


**Figure 1:** Test Configuration

# 4 Test Results

The results show a significant difference in power consumption depending on the encryption algorithm. All tests were performed with a buffer size of 4096 bytes and 10,000 cycles. When encryption and decryption were performed repeatedly, the power consumption of each was measured. For the Arm platform, we measured both with and without NEON-based instructions.

| Cipher Algorithm | Run Time (s) | Net Power Consumption (W) | Performance (KB/s) | CPB (Cycle per byte) |
|---|---|---|---|---|
| ChaCha20 Encryption | 16 | 13.7475 | 655,403 | 7.171 |
| ChacCha20 Decryption | 15 | 13.8720 | 858,313 | 5.476 |
| AES-128-XTS Encryption | 57 | 13.7242 | 184,436 | 25.483 |
| AES-128-XTS Decryption | 100 | 14.9976 | 102,769 | 45.734 |
| AES-256-XTS Encryption | 79 | 13.2881 | 131,705 | 35.686 |
| AES-256-XTS Decryption | 143 | 12.0613 | 72,101 | 65.186 |
| Adiantum-XChaCha20 Encryption | 21 | 14.3657 | 513,948 | 9.145 |
| Adiantum-XChaCha20 Decryption | 15 | 14.4480 | 507,349 | 9.264 |
| CHAM 128/128-XTS Encryption | 46 | 12.1461 | 225,518 | 20.841 |

| | | | | |
|---|---|---|---|---|
| CHAM 128/128-XTS Decryption | 68 | 13.1824 | 151,997 | 30.922 |
| CHAM 128/256-XTS Encryption | 54 | 13.1867 | 191,611 | 24.529 |
| CHAM 128/256-XTS Decryption | 82 | 11.9371 | 127,210 | 36.947 |
| LEA-128-XTS Encryption | 16 | 15.2100 | 632,616 | 7.429 |
| LEA-128-XTS Decryption | 24 | 13.9500 | 435,234 | 10.799 |
| LEA-256-XTS Encryption | 22 | 15.2836 | 487,460 | 9.642 |
| LEA-256-XTS Decryption | 31 | 13.8542 | 331,178 | 14.192 |

**Table 4:** Test results (X86 Platform)

On the X86 platform, the ChaCha20 encryption algorithm performed best. Google's Adiantum algorithm, which is based on XChaCha20, also performed as well as ChaCha20. The CHAM128/128-XTS encryption algorithm consumed the least power when encrypting, as shown in Figure 2. Unusually, the CHAM 128/256-XTS method was measured to consume the least power for decryption. Figure 4 shows the sum of encryption power consumption and decryption power consumption, or total power consumption. CHAM 128/256-XTS performed the best in terms of total power consumption. We expected LEA and other lightweight encryption algorithms to measure lower, but they consumed more power than the AES encryption algorithm.

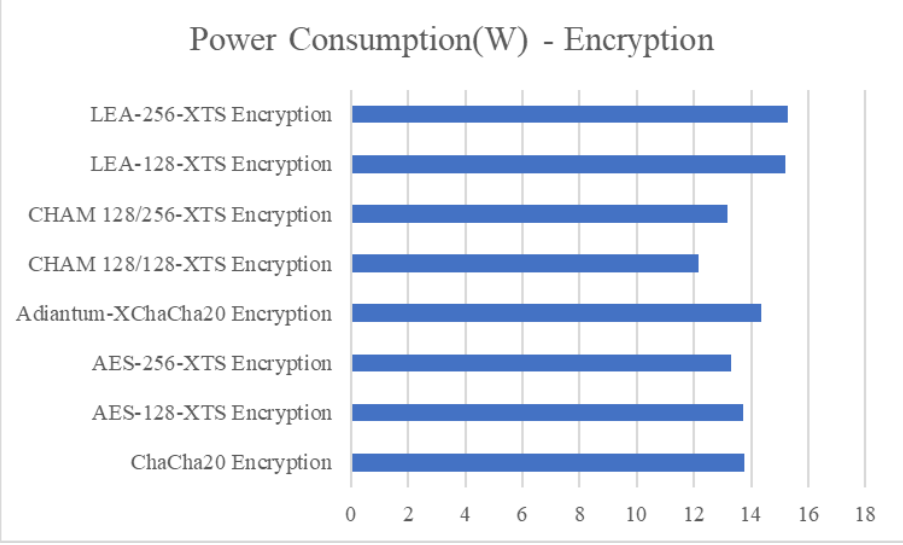


**Figure 2:** Power Consumption - Encryption

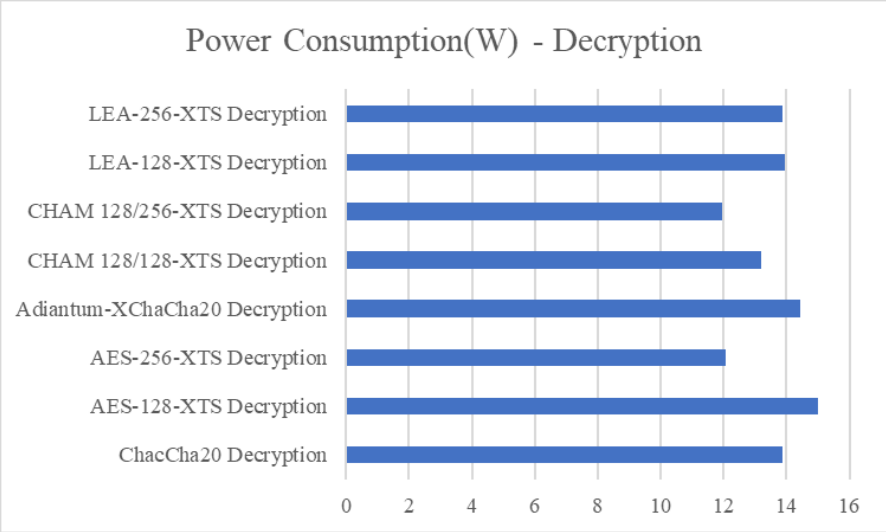


**Figure 3:** Power Consumption - Decryption

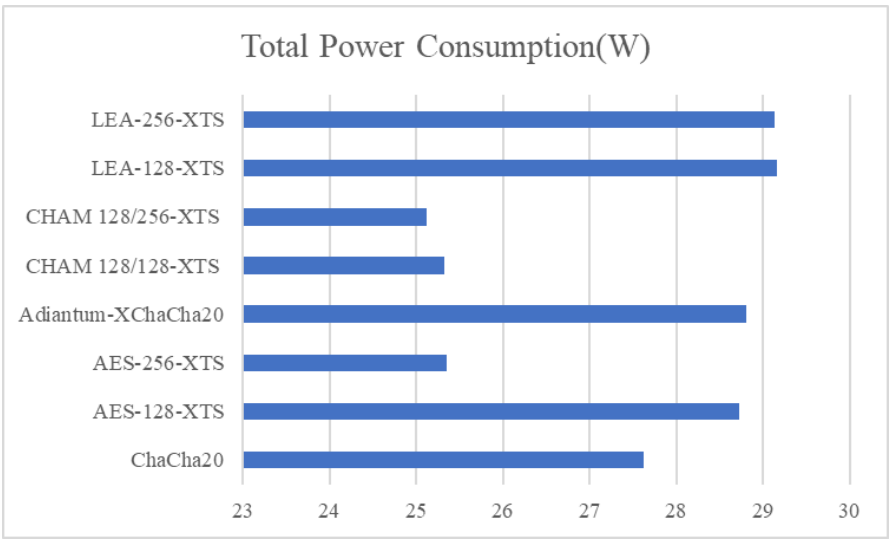


**Figure 4:** Total Power Consumption

The results of encrypting and decrypting each encryption algorithm on the ARM platform are shown below. The ChaCaha20 algorithm using the ARM-optimized NEON instructions performed best on the ARM platform.

| Cipher Algorithm | Run Time (s) | Net Power Consumption (W) | Performance (KB/s) | CPB (Cycle per byte) |
|---|---|---|---|---|
| ChaCha20 Encryption | 51 | 1.5671 | 194,772 | 9.242 |
| ChacCha20 Decryption | 52 | 1.5300 | 194,847 | 9.238 |
| ChaCha20 Encryption - NEON | 24 | 2.0100 | 412,217 | 4.367 |
| ChacCha20 Decryption - NEON | 25 | 2.2464 | 411,150 | 4.378 |
| AES-128-XTS Encryption | 147 | 2.1820 | 68,164 | 26.407 |
| AES-128-XTS Decryption | 305 | 1.9995 | 33,044 | 54.472 |
| AES-256-XTS Encryption | 203 | 2.1653 | 49,305 | 36.507 |
| AES-256-XTS Decryption | 428 | 1.8017 | 23,485 | 76.643 |
| Adiantum-XChaCha20 Encryption | 69 | 1.9304 | 147,127 | 12.234 |
| Adiantum-XChaCha20 Decryption | 69 | 1.8939 | 145,067 | 12.408 |
| Adiantum-XChaCha20 Encryption - NEON | 35 | 2.6229 | 286,138 | 6.291 |
| Adiantum-XChaCha20 Decryption - NEON | 36 | 2.4000 | 280,505 | 6.417 |
| CHAM 128/128-XTS Encryption | 110 | 2.1207 | 91,285 | 19.718 |
| CHAM 128/128-XTS Decryption | 136 | 2.1626 | 73,351 | 24.539 |
| CHAM 128/256-XTS Encryption | 130 | 2.1822 | 77,666 | 23.176 |
| CHAM 128/256-XTS Decryption | 162 | 2.1578 | 61,813 | 29.120 |
| LEA-128-XTS Encryption | 59 | 2.2332 | 170,596 | 10.551 |
| LEA-128-XTS Decryption | 66 | 2.2090 | 151,896 | 11.850 |
| LEA-256-XTS Encryption | 87 | 1.7131 | 115,555 | 15.577 |
| LEA-256-XTS Decryption | 88 | 1.9473 | 114, 405 | 15.734 |

**Table 5:** Test results (Arm Platform)

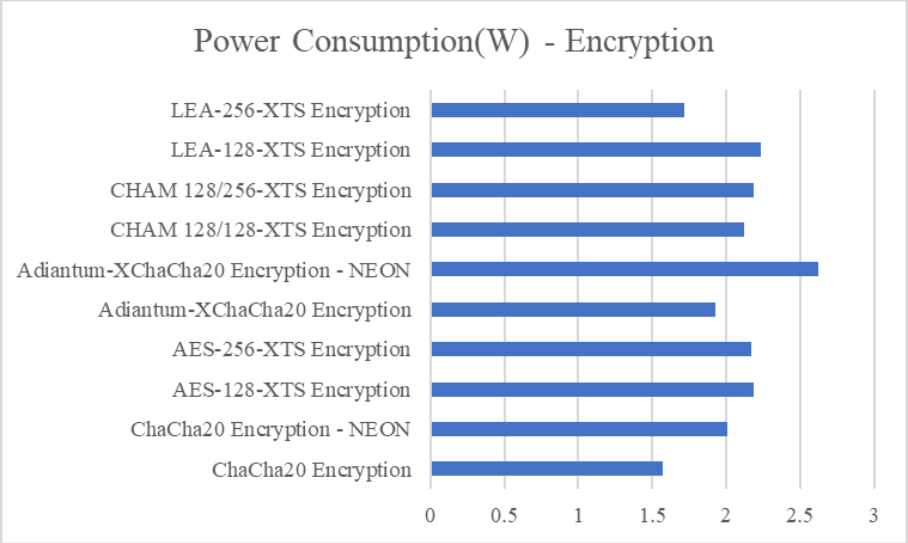


**Figure 5:** Power Consumption - Encryption

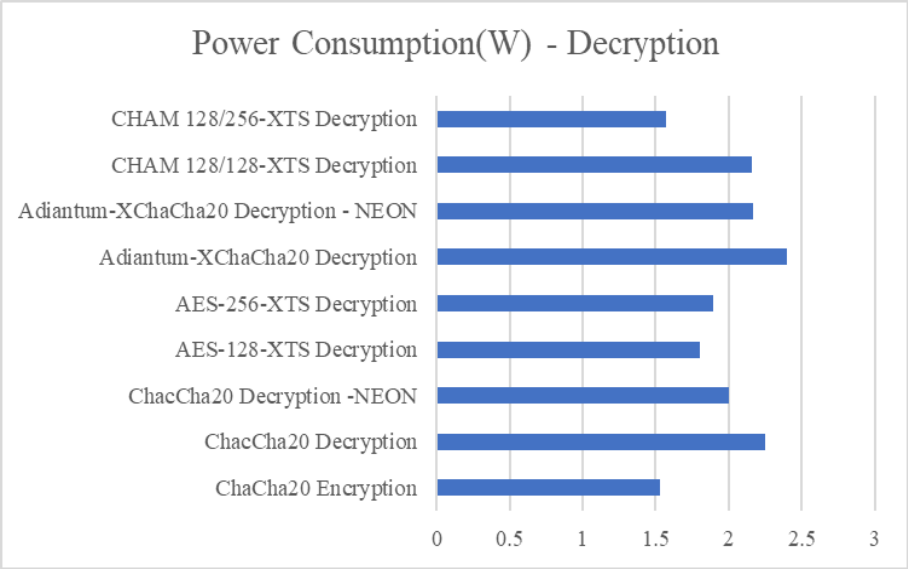


**Figure 6:** Power Consumption - Decryption

On the ARM platform, the power consumption for encryption was higher than the power consumption for decryption. This is probably due to the hardware limitations of the ARM platform. Interestingly, the power consumption of the lightweight encryption algorithms did not show a significant difference compared to AES.

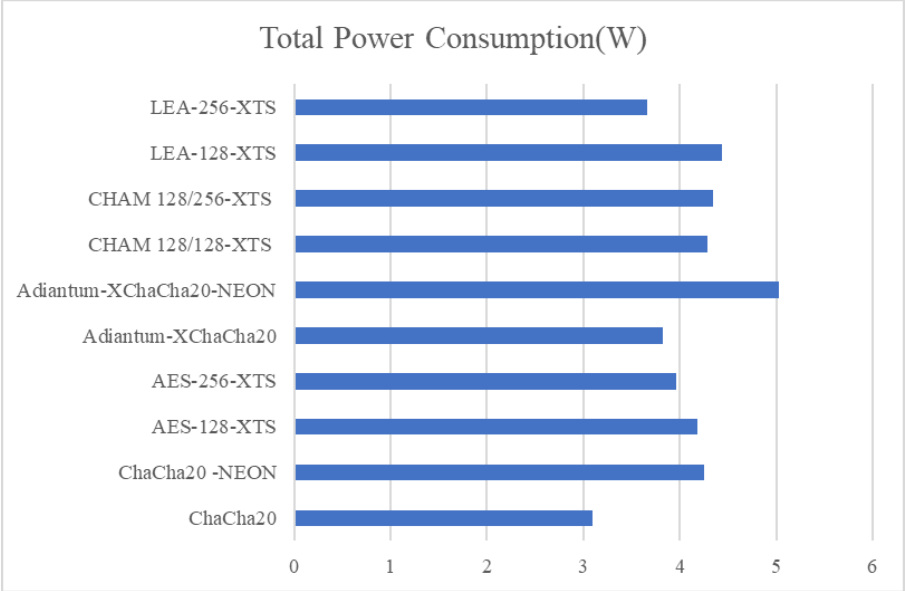


**Figure 7:** Total Power Consumption

On Arm-based platforms, the ChaCha20 encryption algorithm was the most power-efficient. Specifically, when using instructions optimized for the NEON architecture, the ChaCha20 algorithm showed a speedup of approximately 52.4% in encryption and decryption speeds. Using NEON architecture instructions resulted in a 37.43% increase in power consumption despite the speedup.

# 5 Conclusion

In this paper, the power consumption of each encryption algorithm was measured on X86 and Arm platforms. The results showed that the power consumption of existing lightweight encryption algorithms is more energy efficient than encryption algorithms such as AES. The CHAM encryption algorithm, developed as a lightweight encryption algorithm, showed high energy efficiency on X86-based platforms but was not as energy efficient as other encryption algorithms on ARM platforms. The ChaCha20 encryption algorithm showed low energy efficiency on X86 platforms but had the best energy efficiency on ARM platforms. LEA, another lightweight encryption algorithm, showed moderate energy efficiency on both X86 and ARM platforms. Our results show that X86-based and ARM-based platforms can improve energy efficiency by selecting and using cryptographic algorithms tailored to their respective CPU instruction sets and characteristics. Further research is needed to understand what makes the difference and how existing cryptographic algorithms can be optimized to improve energy efficiency for each platform.

# Acknowledgments

This research was supported by the MSIT (Ministry of Science and ICT), Korea, and supported by the IITP (Institute of Information & communications Technology Planning & Evaluation). (No.2022-0-

00979, Development of technology and test criteria for evaluating the security of self-driving vehicle data and V2X communication network.).

# References

[1] NIST, “Advanced Encryption Standard (AES),” FIPS PUB 197, Nov. 2001.
[2] Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, Marco Macchetti, and Stefano Marchesin, “Efficient Software Implementation of AES on 32-Bit Platform”, Proceedings of CHES'02, volume 2523 of Lecture Notes in Computer Science, pp. 129-142, 2003
[3] Goll, M.; Gueron, S. Vectorization on ChaCha stream cipher. In Proceedings of the 2014 11th International Conference on Information Technology: New Generations, Las Vegas, NV, USA, pp. 612–615, April 2014.
[4] Langley, A.: ChaCha20 and Poly1305 based Cipher Suites for TLS Draft 02. IETF DRAFT 02, IETF (2013), http://tools.ietf.org/html/draft-agl-tls-chacha20poly1305-02
[5] Changho Seo et al., “Research for Speed Improvement Method of Lightweight Block Cipher CHAM using NEON SIMD,” Journal of KIISE, Vol.46, No.5, pp.485-491, May. 2019.
[6] Song, JinGyo, Seo, Seog Chung. “Efficient Parallel Implementation of CTR Mode of ARX-Based Block Ciphers on ARMv8 Microcontrollers”, Applied sciences, vol.11, no.6, 2548
[7] Paul Crowley and Eric Biggers, "Adiantum: length-preserving encryption for entry-level processors," IACR Transactions on Symmetric Cryptology, Vol. 2018, No. 4, pp. 39–61.
[8] Y.B. Kim, H.D. Kwon, S.W. An, H.J. Seo, and S.C. Seo, “Efficient implementation of ARX-based block ciphers on 8-bit AVR microcontrollers”, Multidisciplinary Digital Publishing Institute Mathematics, 8(10), 1837, pp 1-22, Oct. 2020.