CPP ASCON Benchmark

Sebastian Bendezu, Gil Gandionco

Department of Engineering Cal Poly Pomona University

sjbendezu@cpp.edu, grgandionco@cpp.edu

Isabel Warth, Jessica Ramirez,

Department of Engineering Cal Poly Pomona University

[iewarth@cpp.edu](mailto:iewarth@cpp.edu), [jessicar2@cpp.edu](mailto:jessicar2@cpp.edu)

***Abstract*— Lightweight cryptography can greatly benefit from parallelizing both the encryption and decryption scheme and this holds true for the ASCON algorithm. By breaking down the data processed by any device into smaller chunks this not only helps keep the memory requirements low but also ensures full utilization of all cores provided by the processor. The approach was focused on the development side of the benchmark. By using an open source implementation of the ASCON encryption and decryption schemes, it was integrated into a benchmark that measures both encryption and decryption times as well as their throughputs. In order to meet the goals of this ASCON benchmark, the process also made use of threads which serves as its distinctive characteristic. The performance of three x86 and one ARM v8 processors were recorded, graphed and tabulated highlighting comparisons between encryption/decryption times as well as the average throughputs for both single threaded and multithreaded processes. The results emphasized the advantages of the ARM architecture with regards to cryptographic workloads due to the processor’s lack of caches for faster memory access and only possessing single threaded cores which limits its parallelization. However, a different approach that utilizes the closer L1 and L2 caches of the x86 processors through smaller data chunks would yield different results by giving these processors more of an advantage.**

***Keywords—ASCON; Encryption; Decryption; Throughput; Benchmark; Threads; Multithreaded; x86; ARM; ARMv8***

# Introduction

ASCON is a lightweight cryptographic algorithm designed for secure and efficient encryption and decryption in resource-constrained environments like IoT (Internet of Things) and embedded systems. Internet of Things relates to the collective network of devices connected to the internet and exchange data with each other. It includes devices like smartwatches, smart thermostats, and home security systems. With the rise of small-scale devices, emulating performance under various conditions is important for any sort of practical deployment. ASCON was selected by NIST, National Institute of Standards and Technology, known for its capability in resource-constrained environments. It operates under different security levels and performance requirements and is utilized for high security and resistance against many known attacks. It offers both data encryption and integrity protection. ASCON also provides both authenticated encryption with associated data (AEAD) and hashing functionalities.

The goal of the project is to measure Encryption/Decryption times and throughput for ASCON across different files, multi-threading (a selection between 1-4 threads), and different processors (Intel I7, Ryzen 5, Intel I5, and ARM Cortex-A72.

# Overview of Cryptographic Process [1]

## Key Components of ASCON

1. Key (K)
2. Nonce (N): random or pseudo-random number used only once to ensure the security and integrity of data
3. Associated data (AD), optional data that is authenticated but not encrypted
4. Plaintext
5. Ciphertext
6. Authentication tag (T) - is attached to the ciphertext to ensure authenticity and integrity.

## Encryption Process

1. Initialization

Sets up the internal state for encryption or decryption, initializes the 320-bit internal state (S) with values derived from the key, nonce, and a fixed initialization vector (a fixed, predefined constant chosen to define the mode of operations, encryption or decryption). The internal state is represented as five 64-bit words. The key and nonce are split into 64-bit words. The fixed initialization vector is 64-bits.

S = [S0, S1, S2, S3, S4]

S0 = IV, S1 = K0, S2 = K1, S3 = N0, S4 = N1

1. Associated Data Absorption

If associated data is provided, each 64-bit block of AD is XORed into the state, followed by a permutation to ensure mixing (i.e. absorbed into the state).

1. Plaintext Encryption

The plaintext is XORed with the state to generate the ciphertext. The state undergoes a permutation after each block to ensure diffusion, a change of a bit in the plaintext will result in several changes in bits in the ciphertext.

1. Finalization

The key is XORed into the state twice for finalization encryption. A 128-bit tag is generated from the state and attached to the cipher text.

## Decryption Process

1. Initialization

The state is initialized with the same IV, key, and nonce used in encryption

1. Associated Data Absorption

The associated data is processed in the same manner as in encryption

1. Ciphertext Processing

The ciphertext is XORed with the state to recover the plaintext. The ciphertext is integrated into the internal state and undergoes a permutation for mixing.

1. Tag Verification

A tag is generated based on the current state. The tag is created in the current state and must match the tag provided in the ciphertext for the plaintext to be valid. If not, the decryption process failed and the plaintext was modified or corrupted.

# Methodology

The ASCON benchmark was created using an ASCON reference C code from Github implemented by Christoph Dobraunig and Martin Schläffer [2]. This was used for the implementation of the ASCON algorithm for both encryption and decryption and then integrated into the benchmark C code [3]. The major focus of the benchmark is to utilize the cores of each CPU by optimizing the code for that specific function. This is achieved using threads to provide a better understanding of each CPU’s performance per core. Rather than using one large data chunk to process through the ASCON algorithm it is split into 1MB data chunks and iterated over as many times needed to reach the designated data size, up to 1000 times in this case. The benchmark also includes timing routines to accurately time both processes and also calculate the throughput for the given data size.

This benchmark prioritizes the speeds of memory access for each given CPU due to the nature of the benchmark implementation. The processors used in the benchmark and their specifications provided in the table below.

| **CPU** | **Specifications** | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| ***Base Clock*** | ***Max Clock*** | ***Cores*** | ***Threads*** | ***Cache Size*** | ***Arch.*** |
| Intel Core i7-8565U | 1.80 GHz | 4.60 GHz | 4 | 8 | 8 MB | x86 |
| AMD Ryzen 5 3600 | 3.60 GHz | 4.20 GHz | 6 | 12 | 32 MB | x86 |
| ARM Cortex-A72 | 1.50 GHz | 1.50 GHz | 4 | 4 | - | x86 |
| Intel Core i5-1135G7 | 2.40 GHz | 4.20 GHz | 4 | 8 | 8 MB | ARM v8 |

Data for all CPU specs *[4], [5], [6], [7].*

The CPU’s were chosen based on what was readily available and which were also close in terms of specs.

# Results

Once finished compiling the data recorded for all CPUs with regards to encryption/decryption times as well as their respective throughputs for each data size (10 MB to 1GB) for both single threaded and multithreaded processes [8], they were compared against each other.

Figure 1. Comparison of Single Thread Encryption Times for all CPUs at Varying Data Sizes

Figure 2. Comparison of Single Thread Decryption Times for all CPUs at Varying Data Sizes

Figure 3. Comparison of Multi-Thread(4) Encryption Times for all CPUs at Varying Data Sizes

Figure 4. Comparison of Multi-Thread(4) Decryption Times for all CPUs at Varying Data Sizes

TABLE II

| **CPU** | **1 Thread** | **2 Threads** | **3 Threads** | **4 Threads** |
| --- | --- | --- | --- | --- |
| Intel Core i7-8565U | 4.55 MB/s | 9.05 MB/s | 13.53 MB/s | 16.07 MB/s |
| AMD Ryzen 5 3600 | 4.34 MB/s | 7.87 MB/s | 11.58 MB/s | 15.46 MB/s |
| ARM Cortex-A72 | 4.31 MB/s | 8.64 MB/s | 13.29 MB/s | 17.96 MB/s |
| Intel Core i5-1135G7 | 4.47 MB/s | 7.99 MB/s | 11.20 MB/s | 13.12 MB/s |

Data for all CPU Average Encryption Throughputs for Each Thread Amount

TABLE III

| **CPU** | **1 Thread** | **2 Threads** | **3 Threads** | **4 Threads** |
| --- | --- | --- | --- | --- |
| Intel Core i7-8565U | 4.48 MB/s | 8.93 MB/s | 13.58 MB/s | 16.22 MB/s |
| AMD Ryzen 5 3600 | 4.02 MB/s | 7.56 MB/s | 10.94 MB/s | 14.69 MB/s |
| ARM Cortex-A72 | 4.44 MB/s | 8.87 MB/s | 13.65 MB/s | 18.38 MB/s |
| Intel Core i5-1135G7 | 4.09 MB/s | 7.06 MB/s | 10.33 MB/s | 14.50 MB/s |

Data for all CPU Average Decryption Throughputs for Each Thread Amount

# Conclusion

The Raspberry Pi 4B’s ARM Cortex-A72 processor displayed better encryption and decryption times using the ASCON algorithm and produced better throughputs utilizing multiple threads when compared to the other x86 processors.

Since the ARM Cortex-A72 does not have a cache to improve the memory access speeds it not only relies on its cores but also its ARM v8 architecture. These results point towards the ARM architecture giving the advantage to the Raspberry Pi 4B with handling lightweight cryptography such as ASCON. Furthermore, its lack of multithreaded cores also gives credit to its efficiency per core processing.

However, many changes can be made to the benchmarks approach, starting with the size of the data chunks used in the interactions. Lowering the sizes down into the kilobytes may actually give the advantage to the x86 processors that actually do have caches and can leverage faster memory access with the L1 and L2 caches. Given that ASCON is a lightweight scheme this approach is still valid since the main benefit is its low memory requirements.

##### References

1. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "ASCON v1.2: Submission to NIST," NIST Lightweight Cryptography Round 2 Submission, [Online]. Available: <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf>
2. C. Dobraunig and M. Schläffer, "ascon-c: Reference implementation of Ascon AEAD128," GitHub repository, 2024. [Online]. Available:<https://github.com/ascon/ascon-c/tree/main/crypto_aead/asconaead128/ref>
3. S. Bendezu, "ASCON Benchmarking and Implementation," GitHub repository, 2024. [Online]. Available: <https://github.com/california-polytechnic-university/ECE4300_Fall2024/tree/main/GROUP%20H/ASCON>
4. PassMark Software, "ARM Cortex-A72 4 Core 1500 MHz CPU Benchmarks," CPU Benchmark, Nov. 2024. [Online]. Available: <https://www.cpubenchmark.net/cpu.php?cpu=ARM+Cortex-A72+4+Core+1500+MHz&id=3917>
5. PassMark Software, "AMD Ryzen 5 3600 CPU Benchmarks," CPU Benchmark, Nov. 2024. [Online]. Available: <https://www.cpubenchmark.net/cpu.php?cpu=AMD+Ryzen+5+3600&id=3481>
6. PassMark Software, "Intel Core i5-1135G7 @ 2.40GHz CPU Benchmarks," CPU Benchmark, Nov. 2024. [Online]. Available: <https://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i5-1135G7+%40+2.40GHz&id=3830>
7. PassMark Software, "Intel Core i7-8565U @ 1.80GHz CPU Benchmarks," CPU Benchmark, Nov. 2024. [Online]. Available: <https://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i7-8565U+%40+1.80GHz&id=3308>
8. S. Bendezu, G. Gandionco, I. Warth, and J. Ramirez, "Raw Data for ASCON Benchmarking," Google Sheets, 2024. [Online]. Available: <https://docs.google.com/spreadsheets/d/1Ro_3LR3veLTBdUkEsPR_yVBIj7Qq-iUNTovWKyqG_t0/edit?usp=sharing>