Resource Consumption Evaluation of C++ Cryptographic Libraries on Resource-Constrained Devices

Razvan Raducu, Ricardo J. Rodríguez, and Pedro Álvarez

Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
{razvan, rjrodriguez, alvaper}@unizar.es

Abstract. With the constant growth of IoT devices, software performance and memory usage have become relevant aspects when choosing the most suitable and optimal configuration of these resource-constrained devices. Moreover, in certain scenarios security must be guaranteed to protect data confidentiality, which imposes another resource consumption overhead. In this work-in-progress we evaluate the resource consumption of two widely-used block ciphers (AES and 3DES) and stream ciphers (Salsa20 and Chacha20), implemented in two C++ libraries (`Crypto++` and `Botan`), to find out which library and algorithms are the most efficient for such devices. In addition, we also evaluate whether the type of input data affects the resource consumption. Our results show that the memory consumption is similar across both libraries and algorithms. In terms of CPU, `Crypto++` outperforms `Botan`, with ChaCha20 achieving the best performance rates. Regarding the type of input data, no major impact has been noticed.

Keywords: performance evaluation · memory usage · cryptographic libraries · resource-constrained devices

1 Introduction

The evaluation of a program's resource consumption helps engineering teams choose the system configuration in which their software programs can be optimally deployed. This kind of decision becomes especially critical when software programs are intended to be run on resource-constrained devices, such as Internet-of-Things devices or System-on-a-Chip boards [3].

In addition, these devices may require some sort of cryptography to guarantee data confidentiality. To incorporate this feature, software developers can make use of cryptographic libraries that provide cryptographic primitives implementing widely-known algorithms such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), or Salsa20, to name a few.

However, the large number of cryptographic libraries available can make it difficult to choose which one is the best for a particular scenario. As the implementation of the cryptographic primitives varies, some of them will be inefficiently implemented, affecting the resource utilization and, consequently, execution time and energy consumption.

Modern cryptography is mainly divided into two types of encryption schemes: symmetric and asymmetric [14]. In the symmetric encryption scheme the key is used to both encrypt and decrypt data, whereas in the asymmetric scheme (also known as public-key encryption scheme) the key used for encryption and the one used for decryption are different but mathematically linked. For the sake of space, in this work-in-progress we only focus on symmetric encryption. We plan to extend this work to asymmetric encryption schemes as immediate future work. Symmetric encryption algorithms can be further divided into block or stream ciphers. Block ciphers split the data to be encrypted into fixed-size blocks and encrypt one block at a time. Stream ciphers, on the contrary, break the data down into bits and individually encrypt each one of them.

We evaluate the performance of cryptographic primitives in two cryptographic libraries written in C++, which are chosen because of their popularity and comprehensiveness. In particular, we evaluate two implementations of block ciphers and another two implementations of stream ciphers which are common across these libraries. These two kinds of symmetric ciphers were chosen as they are best suitable for different use cases: block ciphers are a good choice when the amount of data is known in advance, whereas stream ciphers are more appropriate when the amount of data is either unknown or continuous (such as in network streams). In addition, we evaluate whether the type of input data affects the performance of the cryptographic primitives.

In brief, the research questions (RQ) that we address in this work-in-progress are the following:

RQ1. Which cryptographic primitive is the most suitable for resource-constrained devices?

RQ2. Does the type of input data (i.e., random, video, audio, or text data type) affect the performance of the cryptographic algorithms?

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 depicts the methodology we applied when performing the evaluation. Section 4 details the evaluation itself, describing the selected algorithms for comparison, the experimental setup, the discussion of the results, and the limitations of our work. Finally, Section 5 concludes our work and establishes future lines of work.

2 Related Work

The need of performance measurements dates several decades back [17]. One of the most common approaches when determining the performance of software in the discipline of software performance engineering is based on measurements during the actual program execution [30], as opposed to other approaches like model-based or performance prediction [4].

In any event, the evaluation of programs' resource consumption help developer teams choose the system in which their software can be optimally deployed.

This kind of decision becomes critical especially when software programs are aimed to be executed in resource-constrained devices [3].

Performance of cryptographic primitives has been largely discussed in the literature. In what follows, we focus on the works most similar to ours.

In [28], Tamimi compared the performance of four of the most common cryptographic algorithms (DES, 3DES, Blowfish, and AES), using their own implementation in C# and compiled with Microsoft Visual C# .NET 2003. The results showed that Blowfish has the best performance while, on the other hand, AES has the worst. The authors in [20] also implemented widely used cryptographic algorithms (namely, DES, 3DES, AES, RSA, and Blowfish), but in Java language instead of C#. Regarding the type of input, only text and image data were used. Metrics such as performance or memory use, among others, were evaluated. An evaluation of symmetric (AES, DES, Blowfish) as well as asymmetric (RSA) cryptographic algorithms implemented in Java by taking different types of data like binary, text, and image is also provided in [19].

A comprehensive performance evaluation of popular symmetric and asymmetric key encryption algorithms is provided in [12]. The main purpose of the evaluation is selecting the best algorithm for resource-constraint devices. The authors tested symmetric key encryption (AES, RC4, Blowfish, CAST, 3DES, and Twofish) and asymmetric encryption (DSA and ElGamal) algorithms implemented in Python on various types of input files (text, audio, and video).

The seminal work of D. A. Menascé [18], presents a quantitative analysis to illustrate the effect of using a specific set of cryptographic algorithms on the performance of a computer system. In particular, the analysis is focused on the performance of digital signatures using MD5 and SHA-1 cryptographic hash functions and on the SSL protocol [11] using different combinations of symmetric key algorithms (RC4 and 3DES), two hash functions (MD5 and SHA-1 again), and three key lengths (512, 768, and $1,024$ bits). Menascé concluded that there is a need to understand which level of security is required so as to be protected against possible threats while minimizing performance penalties as much as possible.

An extensive and complete analysis of eight open-source cryptography libraries is provided in [8], in which 15 different ciphers are examined and compiled using four different C++ compilers. However, unlike in this work, only electronic code-book (ECB) encryption mode is considered for the comparison.

Recently, the authors in [27] studied the performance of symmetric key algorithms (AES and DES) versus the RSA asymmetric key encryption algorithm, concluding that RSA takes the longest time for encryption while AES takes the shortest. Unlike ours, their work only considered text data for the evaluation.

The work most similar to ours is [1], in which the performance of ten block ciphers implemented in six C/C++ open source libraries under different data loads is assessed. Unlike us, they do not limit the experimental scenario to resource-constrained devices and only consider CBC mode for all block ciphers. However, no details are given on the type of input used for evaluation.

Our work differs in several ways from the works mentioned above. First, we focused on resource-constrained devices, carrying out the evaluation on a Raspberry Pi Model 4 B. Second, we evaluated four different cipher modes of operations, as well as two stream ciphers. Last, we considered four data types (random, video, audio, and textual data).

3 Methodology

The methodology we used to carry out the experiments is focused on finding out which algorithm of each type is the best in terms of performance and memory consumption, and then compare them.

We first evaluate and compare the block ciphers (AES and 3DES) with each other and the stream ciphers (Salsa20 and ChaCha20) in the same way. Regarding their configuration, we evaluate AES and 3DES with a key length of 192 bits in CBC, OFB, CTR, and CFB modes, and Salsa20 and ChaCha20 with key lengths of 128 and 256 bits. We use the same initialization vector and the same key for all algorithms. We then compare the best algorithm from each category using the same evaluation metrics.

For the input data of the assessment, we created a corpus comprising random data from `/dev/urandom`, audio data from a copyright-free version of *"Psychopathology of Everyday Life"*[1], video data from *"Night of the Living Dead"*[2] (also copyright-free), and textual data from *"El Quijote"*[3]. These inputs were divided into 128KiB, 256KiB, 512KiB, 1MiB, 4MiB, and 8MiB chunks.

Regarding execution, we launch one execution of each algorithm for each combination of input type, input size, key-length and operation mode, measuring the resource usage of each execution. Each execution consists of the encryption of the given input and the decryption of the result. In addition to the time measurement, we compute the MD5 of the decryption output to verify the correctness of the operations by comparing it to the MD5 of the original input file. This process was repeated 20 times to obtain the running average. The total execution time of these tests was almost 5780 minutes.

Furthermore, we developed a tool dubbed `EvalMe` to carry out all the experiments presented in this work-in-progress. It monitors the usage of two main resources: CPU and memory. We make use of `Hyperfine` [22], a cross-platform command-line benchmarking tool, to measure the CPU usage. Likewise, we use `Psutil` [24], a Python library for retrieving system resources utilization of running processes, to measure memory usage. `EvalMe` allows the user to specify how many executions of the program should be performed and monitored. By default, it performs 10 executions without a warm-up run. The results are the average resource consumption for all executions. `EvalMe` outputs results in either

[1] Available in `https://archive.org/details/psychopathology_everyday_life_ms_librivox`, accessed on April 28, 2021.

[2] Available in `https://archive.org/details/night_of_the_living_dead`, accessed on April 28, 2021

[3] Available in `https://www.gutenberg.org/ebooks/2000`, accessed on April 28, 2021

human-readable or JSON format, making it easy to integrate into pipelined analysis systems. `EvalMe` is open source and licensed under GNU/GPL v3, publicly available in our repository [23].

4 Evaluation

In this section we briefly present the tested algorithms, the settings we used in our experiments, the discussion of results, and the limitations of our work.

4.1 Selected Algorithms

Regarding the symmetric key algorithms selected for evaluation, we choose two block cipher algorithms and another two stream cipher algorithms. As block ciphers, we select 3DES and AES as they are the most secure block ciphers at the moment of this writing. As stream ciphers, we select Salsa20 and its evolution ChaCha20, which is becoming one of the most used stream ciphers [15]. We briefly explain them in the following.

Triple Data Encryption Standard (3DES). Developed in 1974, DES was the first encryption standard to be recommended by the National Institute of Standards and Technology (NIST). 3DES was proposed in 1998 as a replacement for DES due to advances in key searching [5]. 3DES applies the DES cipher algorithm three times to each data block. The block size is 64 bits and the key length varies between 168, 112, or 56 bits. 3DES supports different modes: ECB, Cipher Block Chaining (CBC), Cipher FeedBack (CFB), Output FeedBack (OFB), and Counter (CTR). Among these, ECB is generally not recommended as it is semantically insecure [2] (i.e., an adversary that merely observes an ECB-encrypted ciphertext can gain information about the corresponding plaintext).

Advanced Encryption Standard (AES). AES was also recommended by NIST as a replacement of DES in 1998, and standardized in 2001 [26]. The block size is 64-bit length and the key length varies between 128, 192, and 256 bits. As 3DES, it supports also different modes.

Salsa20. Salsa20 is a family of 256-bit stream ciphers designed in 2005 [7]. The block size is 64 bytes (512-bit) and the key length is either 128 bits or 256 bits. The encryption/decryption model used by Salsa20 is similar to the model followed by any block cipher in CFB, OFB, and OTR modes, among others modes (except CBC).

ChaCha20. ChaCha20 is an evolution of Salsa20, published in 2008 [6]. This stream cipher has been selected as a replacement for RC4 in the TLS protocol, used for Internet security. As in Salsa20, the block size is 64 bytes and the key length is either 128 bits or 256 bits. Regarding the encryption/decryption model, it is similar to the model used by Salsa20.

4.2 Experimental Setup

As experimental hardware, we use a Raspberry Pi 4 Model B rev 1.1 running a Raspbian GNU/Linux Debian 10 (buster) on top of a (32-bit little Endian architecture) ARM Cortex-A72 1.50GHz CPU and 4GiB of RAM. As software, we consider `Crypto++` version 8.4 [9] and `Botan` version 2.17.3 [16]. These software libraries were compiled with GNU `g++` version 8.3.0 (Raspbian 8.3.0-6+rpi1) and their default optimization flag (`O3` in both cases). We use `EvalMe` to monitor resource consumption and automate the process as described in Section 3.

4.3 Discussion of Results

Figure 1 depicts the performance results of block ciphers in `Crypto++` (top figures) and `Botan` (middle) and stream ciphers in `Crypto++` (left-bottom) and in `Botan` (right-bottom). Regarding block ciphers, we evaluate 4 operation modes (CBC, OFB, CTR, and CFB) with 192-bit key length. AES always outperforms 3DES in every possible combination, regardless of the cryptographic library. No major differences are observed regarding the performance of modes in 3DES-`Crypto++` (the best mode is CFB, with an average performance of 6.09 MiB/s, while OFB is the worst with 5.58 MiB/s). Regarding AES-`Crypto++`, the best performing mode is CBC (12.05 MiB/s) and the worst is CFB (10.92 MiB/s). On the contrary, the best 3DES-`Botan` mode is CBC with an average performance of 5.18 MiB/s, while the worst is CTR with 2.94 MiB/s. Regarding AES-`Botan`, the best operation mode is CBC (6.6 MiB/s) and the worst is CTR (3.29 MiB/s). Concerning stream ciphers, we evaluate both 128 and 256-bit key lengths. The difference of average performance is negligible in both libraries. In `Crypto++`, ChaCha20 with a 128-bit key length is the best cipher (13.12 MiB/s), while the worst is Salsa20 with a 128-bit key length (12.89 MiB/s). In `Botan`, the best cipher is Salsa20 with a 128-bit key length (4.16 MiB/s) and the worst is ChaCha20 with a 128-bit key length (3.93 MiB/s). As observed, `Crypto++` always outperforms `Botan`.

Figure 2 shows the performance results of the best block cipher against the best stream cipher for each library with different input types and same key length. Regarding `Crypto++` (top figures), ChaCha20 achieves better overall performance results than AES, regardless of the input type or its size. For AES, there are no noteworthy differences when encrypting and decrypting different input types with different sizes (except when working with textual data of 128 and 256 KiB and when dealing with video data of 128 KiB). The performance of ChaCha20 also tends to be similar, regardless of input type and sizes (except when dealing with textual and video data of 128 KiB). Regarding `Botan`, AES clearly outperforms Salsa20. There is also a clear tendency toward higher performance rates with larger files, regardless of the input type. This tendency has been previously documented [8]. We are currently conducting more detailed experiments in order to discover the reason for this behavior. There are also cases in which AES underperforms, such as audio data of 128 and 256 KiB, and

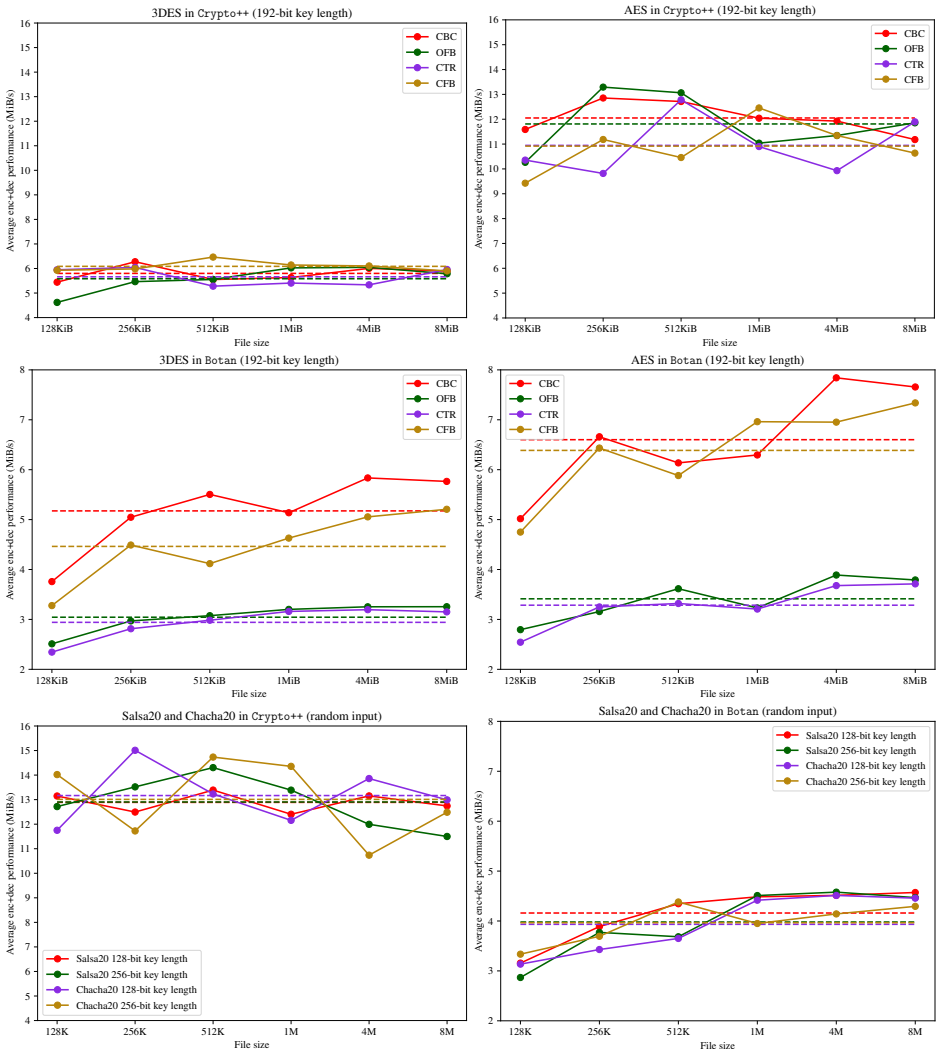

Fig. 1: Performance of block ciphers in `Crypto++` (top figures) and `Botan` (middle) and stream ciphers in `Crypto++` (left-bottom) and in `Botan` (right-bottom).

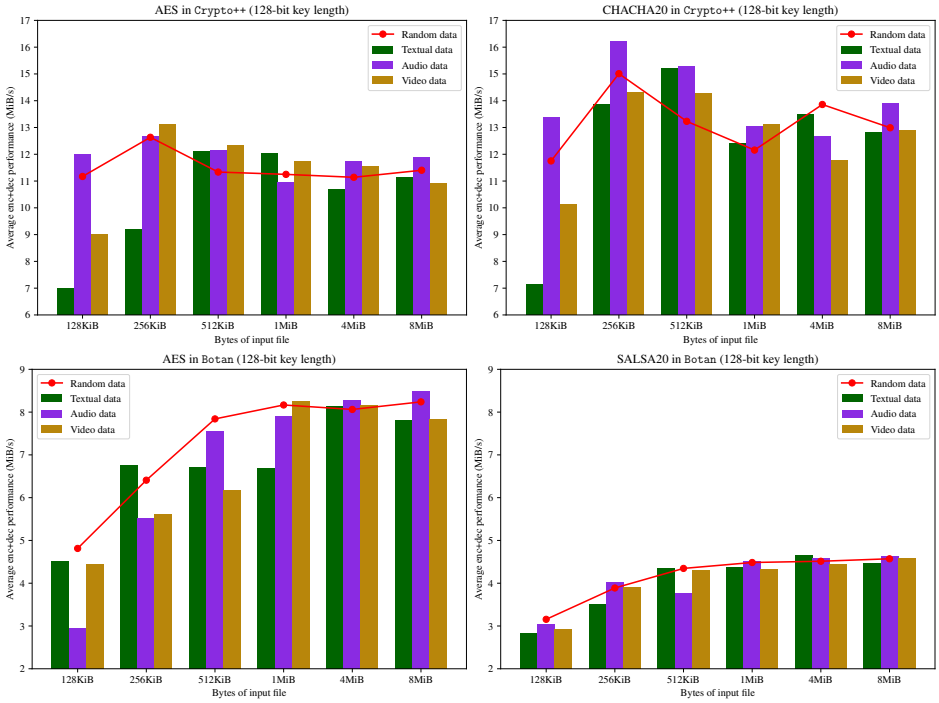

Fig. 2: Performance of the best block and stream cipher in `Crypto++` (top figures) and `Botan` (bottom).

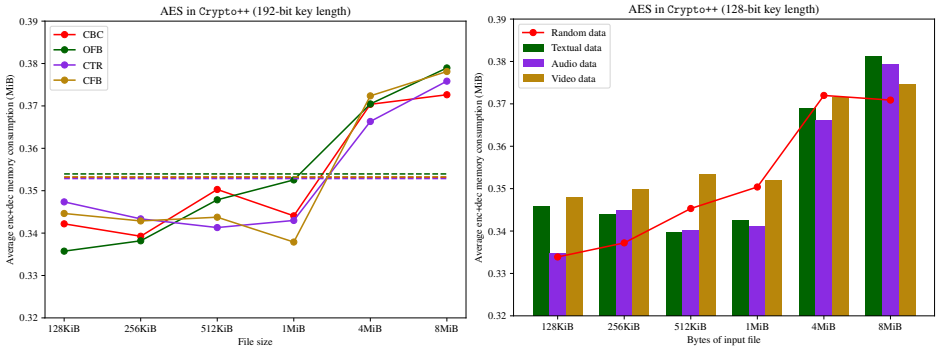

Fig. 3: Average memory usage for AES in `Crypto++`.

video data of 256 and 512 KiB. No major differences between input types are observed in Salsa20.

Regarding the memory consumption, we have observed no differences between executions. On average, 0.36 MiB of RAM were consumed, regardless of the library, algorithm, mode of operation, key length or input type. For the sake of space, we only show the memory consumption of AES in `Crypto++` (Figure 3).

4.4 Limitations

A main limitation of our work is the measurement. Since we are working at the user-space level, monitoring and measurement of resource usage is restricted to polling methods. While running a specified program to retrieve the resource usage for each time period and average computation may seem appropriate, we may be skipping consumption variations (high or low) that may occur during execution. Using a kernel space level monitoring tool would allow us to provide more accurate measurements.

Besides, our results are limited in terms of how the libraries were compiled (we only evaluated one compiler) and the optimizations they were compiled with (we used the default optimization flag). Also, we evaluated only CPU usage and RAM consumption, when there are other crucial measurements like latency, power consumption, or battery drainage.

5 Conclusions and Future Work

The results showed that the higher performance rates are achieved with larger files in `Botan`, unlike `Crypto++`, in which the input size has no major impact. In all our experiments, `Crypto++` clearly outperforms `Botan`. Let us remark that we have empirically observed small variations on performance between different runs of the same configuration, which may indicate that external factors, such as the processor's heat, can be affecting its performance. This is an important issue for resource-constrained devices that requires further research. The best algorithm, in terms of performance, is ChaCha20. Regarding memory consumption, there is no difference between any of the algorithms. Our results also showed that the type of input data has no impact on the performance of the cryptographic primitives, with few exceptions that we believe require further research.

Our immediate step continuing this work is to evaluate the performance and memory consumption considering different optimization flags and different C++ compilers, such as `Clang` and `Intel C++ Compiler`. Our preliminary results with `g++` show that the executions with the default optimizer flag, `O3`, do not have the shorter execution time, while memory consumption are equal across all the executions. Moreover, we plan to extend our evaluation so as to cover asymmetric encryption algorithms and also extend the study to other IoT devices, such as Arduino boards and ESP8266 chips.

In addition, we also aim to assess the power consumption of the tested devices, since power consumption is key in the context of resource-restrained

devices. Particularly, in battery-powered devices battery drain is critical. The power consumption of cryptographic primitives has already been measured in the literature [29,10,25,13,21], showing that the optimal algorithm for a given device is not necessarily the most efficient in terms of power consumption which, in turn, certifies that choosing the "best" algorithm implies more than just selecting the fastest.

References

1. Alrowaithy, M., Thomas, N.: Investigating the Performance of C and C++ Cryptographic Libraries. In: Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools. pp. 167–170. VALUETOOLS 2019, Association for Computing Machinery, New York, NY, USA (2019)
2. Aumasson, J.P.: Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press (2017)
3. Babovic, Z.B., Protic, J., Milutinovic, V.: Web Performance Evaluation for Internet of Things Applications. IEEE Access **4**, 6974–6992 (2016)
4. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Transactions on Software Engineering **30**(5), 295–310 (2004)
5. Barker, E., Mouha, N.: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. techreport NIST Special Publication 800-67. Revision 2, National Institute of Standards and Technology (Nov 2017)
6. Bernstein, D.J.: ChaCha, a variant of Salsa20. resreport, University of Illinois (2008)
7. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers, pp. 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
8. Bingmann, T.: Speedtest and Comparsion of Open-SourceCryptography Libraries and Compiler Flags. Online; `https://panthema.net/2008/0714-cryptography-speedtest-comparison/` (Jul 2008), accessed on December 11, 2020.
9. Dai, W.: Crypto++ version 8.4. [Online; `https://github.com/weidai11/cryptopp`] (Jan 2021), accessed on Jan 11, 2021.
10. Fotovvat, A., Rahman, G.M.E., Vedaei, S.S., Wahid, K.A.: Comparative performance analysis of lightweight cryptography algorithms for iot sensor nodes. IEEE Internet of Things Journal **8**(10), 8279–8290 (May 2021). https://doi.org/10.1109/JIOT.2020.3044526
11. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. techreport RFC 6101, Internet Engineering Task Force (IETF) (Aug 2011), [Online; `https://tools.ietf.org/html/rfc6101`]. Accessed on April 28, 2021.
12. Haque, M.E., Zobaed, S., Islam, M.U., Areef, F.M.: Performance Analysis of Cryptographic Algorithms for Selecting Better Utilization on Resource Constraint Devices. In: 2018 21st International Conference of Computer and Information Technology (ICCIT). pp. 1–6 (2018)
13. Hatzivasilis, G., Fysarakis, K., Papaefstathiou, I., Manifavas, C.: A review of lightweight block ciphers. Journal of Cryptographic Engineering **8**(2), 141–184 (Jun 2018). https://doi.org/10.1007/s13389-017-0160-y
14. Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press (2020)

15. Krasnov, V.: It takes two to ChaCha (Poly). [Online; https://blog.cloudflare.com/it-takes-two-to-chacha-poly/] (Apr 201), accessed on April 30, 2021.
16. Lloyd, J.: Botan version 2.17.3. [Online; https://github.com/randombit/botan] (Mar 2021), accessed on Mar 11, 2021.
17. Lucas Jr, H.: Performance evaluation and monitoring. ACM Computing Surveys (CSUR) **3**(3), 79–91 (1971)
18. Menascé, D.: Security Performance. IEEE Internet Computing **7**(3), 84–87 (May 2003)
19. Panda, M.: Performance analysis of encryption algorithms for security. In: 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES). pp. 278–284 (2016)
20. Patil, P., Narayankar, P., Narayan D.G., Meena S.M.: A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. Procedia Computer Science **78**, 617–624 (2016), 1st International Conference on Information Security & Privacy 2015
21. Pereira, G.C., Alves, R.C., Silva, F.L.d., Azevedo, R.M., Albertini, B.C., Margi, C.B.: Performance evaluation of cryptographic algorithms over iot platforms and operating systems. Security and Communication Networks **2017** (2017)
22. Peter, D.: Hyperfine version 1.11.0. [Online; https://github.com/sharkdp/hyperfine] (Jan 2021), accessed on Jan 19, 2021.
23. Raducu, R.: Evalme. [Online; https://github.com/reverseame/evalme] (Jan 2021), (Accessed on Jan 19, 2021)
24. Rodola, G.: psutil (version 5.8.0). [Online; https://pypi.org/project/psutil/] (Jan 2021), (Accessed on Jan 19, 2021)
25. Saraiva, D.A.F., Leithardt, V.R.Q., de Paula, D., Sales Mendes, A., González, G.V., Crocker, P.: Prisec: Comparison of symmetric key algorithms for iot devices. Sensors **19**(19) (2019). https://doi.org/10.3390/s19194312
26. Secretary of Commerce: Advanced Encryption Standard. techreport Federal Information Processing Standards Publication 197, National Institute of Standards and Technology (Nov 2001)
27. Sheikh, M.F.A., Gaur, S., Desai, H., Sharma, S.K.: A Study on Performance Evaluation of Cryptographic Algorithm. In: Rathore, V.S., Worring, M., Mishra, D.K., Joshi, A., Maheshwari, S. (eds.) Emerging Trends in Expert Applications and Security. pp. 379–384. Springer Singapore, Singapore (2019)
28. Tamimi, A.K.A.: Performance Analysis of Data Encryption Algorithms. Online; https://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf/index.html. (2006), accessed on December 11, 2020.
29. Thakor, V.A., Razzaque, M.A., Khandaker, M.R.A.: Lightweight cryptography algorithms for resource-constrained iot devices: A review, comparison and research opportunities. IEEE Access **9**, 28177–28193 (Jan 2021). https://doi.org/10.1109/ACCESS.2021.3052867
30. Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: Future of Software Engineering (FOSE'07). pp. 171–187. IEEE (2007)