Characterization of AES Implementations on Microprocessor-based IoT Devices

1st Sunanda Roy
ECE Department
George Mason University
Fairfax, USA
sroy9@gmu.edu

2nd Angelos Stavrou
Kryptowire LLC
McLean, USA
angelos@kryptowire.com

3rd Brian L. Mark
ECE Department
George Mason University
Fairfax, USA
bmark@gmu.edu

4th Kai Zeng
ECE Department
George Mason University
Fairfax, USA
kzeng2@gmu.edu

5th Sai Manoj P D
ECE Department
George Mason University
Fairfax, USA
spudukot@gmu.edu

6th Khaled N. Khasawneh
ECE Department
George Mason University
Fairfax, USA
kkhasawn@gmu.edu

***Abstract*—The increased proliferation of IoT devices and the emergence of 5G networks have necessitated increased security of data storage and communication in such connected devices. Thus, cryptography is used in IoT environments to provide secrecy and integrity to the data as well as both authentication and anonymity to the communications across the IoT network. However, IoT devices are resource-constrained devices; have limited memory, network bandwidth, power, and compute units. Since most of the existing cryptographic algorithms were designed to run on resource powerful devices (e.g., desktops or servers), many of these algorithms may not fit into resource-constrained devices. Therefore, in this work, we present a practical performance analysis of different implementations of the Advanced Encryption Standard (AES), which is the most widely used symmetric-key cryptosystem in the IoT environment. Specifically, we explore execution times, energy consumption, and memory usage of the different AES implementations across 4 different public libraries. Furthermore, our analysis is done using various modes, key sizes, plaintext sizes, and microprocessor-based IoT devices. Our results show that for the same combination of inputs and a given algorithm, different crypto library implementations give results with widely varying relative differences. As per the obtained results, the PyCryptodome library seems to be the most suitable one in terms of both execution time and energy on a resource-constrained IoT device and has the most efficient memory usage.**

***Index Terms*—IoT, energy, time, cryptography, symmetric, encryption, decryption, AES**

I. Introduction

The Internet of Things (IoT) paradigm facilitates having devices connected anytime from everywhere and 5G networks enable the scaling up of IoT networks in terms of the number of devices and overall capacity. The term "Things" in the IoT refers to any device that has sensors (e.g., temperature, pressure, humidity, light, motion, and acceleration) and is connected to the Internet, such as smart thermostats, wearable electronics, structural health monitoring devices [1], and smart home devices [2]. Furthermore, IoT devices are resource-constrained in a multitude of ways; they are battery-powered and have limited memory and compute units.

To ensure the confidentiality, authenticity, and integrity of the data generated by the IoT devices and communication across the IoT network, cryptographic algorithms are utilized. However, most of the standardized cryptographic algorithms are designed to run on resource powerful devices, e.g., desktops, laptops, and servers. Thus, many of these algorithms may not fit into the resource-constrained IoT devices. Therefore, it is very important to understand and compare the energy consumption and performance of current cryptographic algorithms implementation across different state-of-the-art public libraries to be able to streamline their use in IoT environments.

While evaluation of the security levels and resource consumption trade-offs on IoT boards has been done in prior works [3]–[7], these works have limitations; they either use powerful devices in their evaluation, such as smartphones and FPGAs [5]–[8], or focus on a specific platform or a specific implementation [3], [4], making the results not generalizable. Furthermore, while lightweight cryptographic algorithms have been proposed by both academics and industry [9]–[11], they are not standardized, are mostly designed for microcontroller-based devices, and have been found to trade security for energy efficiency [12], [13].

In this work, we experimentally evaluate the execution time, energy, and memory consumption of AES, the most widely used symmetric key cryptographic algorithm in IoT devices. AES is an integral part of numerous open protocol standards, such as IPsec and TLS. Our analysis evaluates different AES software implementations across four of the prominent C language-based crypto libraries. Our exploration includes various encryption/decryption modes, plaintext sizes, key sizes, and microprocessor-based devices.

The contributions of this paper are the following:

1) We present a comparative evaluation study of AES implementation in four state-of-the-art crypto libraries.
2) To the best of our knowledge, this is the first experimental evaluation study that focuses on running

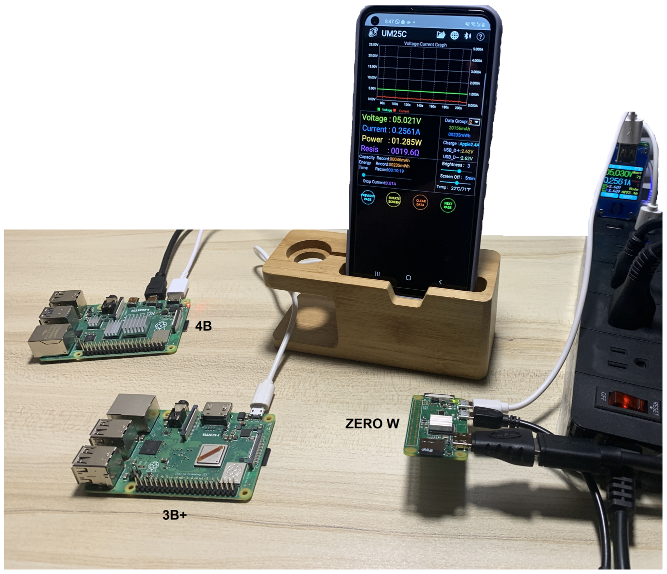


Fig. 1. Experimental setup of our characterization process.

crypto algorithms across a variety of microprocessor-based devices.

3) We provide a characterization of the supported encryption/decryption modes across all studied crypto libraries.

II. Experimental Methodology and Setup

A. Testbed hardware

In this paper, we focus on the microprocessor-based class of devices. The most widely used IoT device belonging to this category is the Raspberry Pi [14]. We used three models with varying hardware configurations as shown in Table I to span across frequently used metrics like speed, memory, size, and weight [15]. While Zero W has the benefit of being ultra-low in cost and size, 3B+ features lower RAM but higher frequency than its more recent counterpart 4B. The experimental setup for our characterization process is shown in Figure 1. In particular, the Zero W model was our prime target due to its highly-resource constrained design. As shown in Table II, we confirmed that this device does not support any ARM NEON instructions [16] in its list of supported features. While 3B+ and 4B models support ARM NEON instructions, in our experiments, we focus on running the algorithms on the CPU without hardware support and leave evaluation with hardware support for future work.

B. Testbed software

All the candidate IoT devices were installed with the same latest version of Raspbian OS version 5.10.63 32-bit. Furthermore, the AES implementation in the following C language-based crypto libraries were used to obtain our results: PyCryptodome [17], GnuPG [18], OpenSSL [19] and wolfSSL [20]. We downloaded the source code of all of the evaluated libraries and compiled them using the default configuration. Furthermore, after studying each of these libraries, we found that different crypto libraries support a different set of block cipher modes as shown in Table IV (refer Table III for mode acronym expansion).

TABLE I
Hardware used in our characterization process.

Raspberry Pi Model	On-chip Memory	Microprocessor	Bus Width (bits)	Operating Frequency (GHz)
Zero W	512MB RAM	ARM1176JZ(F)-S	32	1
3B+	1GB LPDDR2 SDRAM	Cortex-A53 (ARMv8)	64	2.4
4B	4GB LPDDR4-3200 SDRAM	Quad-core Cortex-A72 (ARMv8)	64	1.5

TABLE II
Hardware features supported in Raspberry Pi Zero W.

Attribute	Values
Processor	0
Model Name	ARMv6-compatible processor rev 7 (v6l)
Features	half thumb fastmult vfp edsp java tls
Hardware	BCM2835
Model	Raspberry Pi Zero W Rev 1.1

C. Time measurements

We have developed a testing framework based on the Python programming language. This framework executes a crypto operation (encryption/decryption in this case) for a user-specified number of iterations. The start and end times of each such crypto operation in the Python framework are used to calculate the duration of one such operation. Subsequently, the arithmetic mean of all these consecutive durations is obtained

TABLE III
Block cipher mode acronyms

Acronym	Full Form
ECB	Electronic codebook
CBC	Cipher block chaining
CFB	Cipher feedback
OFB	Output feedback
CTR	Counter

TABLE IV
Block cipher modes support across different crypto libraries

Crypto Library	Block Cipher Mode				
	ECB	CBC	CFB	OFB	CTR
PyCryptodome	✓	✓	✓	✓	✓
GnuPG			✓		
OpenSSL	✓	✓	✓	✓	✓
wolfSSL		✓			✓

TABLE V
MEASUREMENT RESOLUTION SETTINGS

Parameter	Resolution
Time	1 second
Voltage	0.001 V
Current	0.0001 A

TABLE VI
MEASUREMENT ACCURACY

Parameter	Accuracy
Voltage	(0.5 % + 2 digits)
Current	(1 % + 4 digits)

and reported as the average execution time of the evaluated crypto operation.

D. Energy measurements

To measure power, we have used a USB power meter device [21]. The resolution and accuracy of the power measurement device are shown in Tables V and VI respectively. The USB power meter connects the power supply output port and the IoT device input port. For each second, the instantaneous values of voltage and current are logged in an Excel spreadsheet and multiplied to get the instantaneous power. The total energy consumed for a single encryption or decryption operation is obtained by dividing the sum of power values between a start and end time by the number of iterations for that particular encryption or decryption operation.

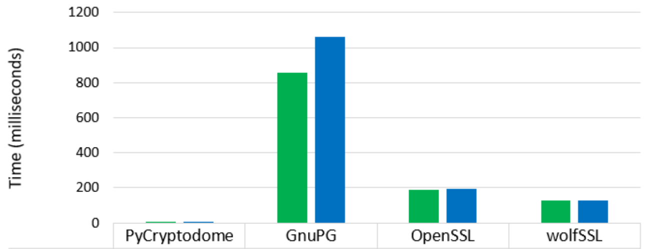
III. Experimental Results

This study aims to provide a comprehensive evaluation of different AES implementations in different crypto libraries across different microprocessor-based IoT devices while varying the encryption/decryption mode, plaintext size, and key size.

A. Execution time, energy consumption, and memory usage

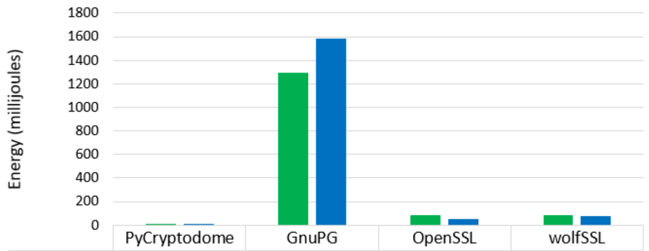
We ran both encryption and decryption operations on a Raspberry Pi Zero W device for a sample plaintext size of 1,024 bytes. In addition, the key size was set to 256 bits and all supported block cipher modes were tested to obtain the geometric mean across all supported modes. Each encryption/decryption operation was executed 1,000 times and the average execution time and energy consumption were measured.

Figure 2 shows the average execution time, while Figure 3 shows the average energy consumption from our experiment. Our results show that the PyCryptodome library consumes significantly less time and energy compared to the other tested libraries. Furthermore, we evaluated the memory usage of running the encryption/decryption operations using the four evaluated libraries for a single iteration of plaintext size 1,024 bytes and key size 256 bits. The results are shown in Figure 4. The results show that wolfSSL is the best-performing library in terms of memory usage. On the other hand, PyCryptodome has very close memory usage to wolfSSL but offers significantly better execution time and energy consumption as shown in Figures 2 and 3.



	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟩 Encryption	2.007	857.384	190.16	125.768
🟦 Decryption	1.944	1061.749	195.752	129

Fig. 2. Average execution time of AES encryption and decryption operations for different crypto libraries



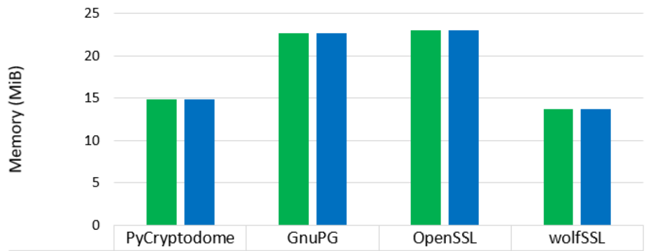
	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟩 Encryption	4.9	1291.21	83.94	84.53
🟦 Decryption	3.8	1587.62	54.51	79.17

Fig. 3. Average energy consumption of AES encryption and decryption operations for different crypto libraries

B. Impact of plaintext sizes on the execution time

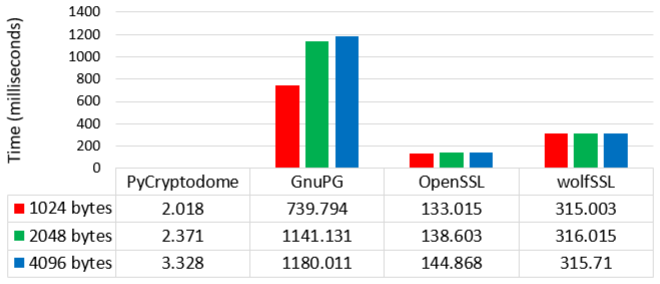
To test the effect of plaintext size on the execution time, we conducted an experiment where we fixed the key size to 256 bits and varied the plaintext sizes. The experiment was done on a Raspberry Pi Zero W device. The number of iterations was set to 2,000 for each crypto operation and the geometric mean of the supported modes was calculated and reported.

Figures 5 and 6 show the average execution time while varying the plaintext size for encryption and decryption operations, respectively. Although the results show that the



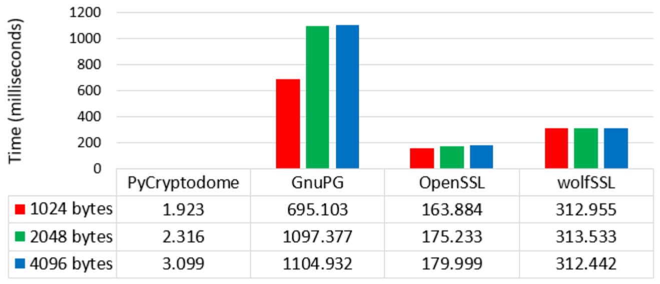
	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟩 Encryption	14.85	22.74	23.05	13.67
🟦 Decryption	14.85	22.74	23.05	13.67

Fig. 4. Memory usage of AES encryption and decryption operations for different crypto libraries



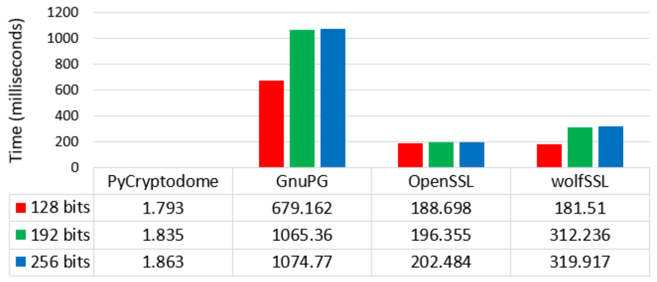
	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟥 1024 bytes	2.018	739.794	133.015	315.003
🟩 2048 bytes	2.371	1141.131	138.603	316.015
🟦 4096 bytes	3.328	1180.011	144.868	315.71

Fig. 5. Average AES encryption execution time while varying plaintext size for different crypto libraries



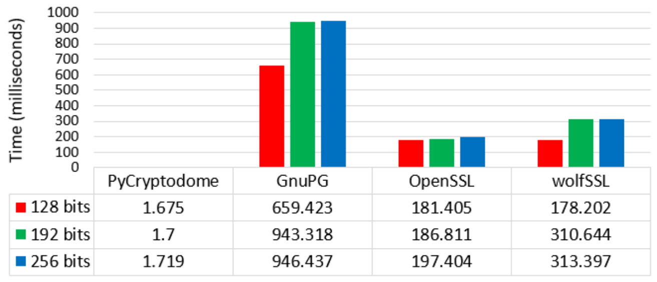
	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟥 1024 bytes	1.923	695.103	163.884	312.955
🟩 2048 bytes	2.316	1097.377	175.233	313.533
🟦 4096 bytes	3.099	1104.932	179.999	312.442

Fig. 6. Average AES decryption execution time while varying plaintext size for different crypto libraries



	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟥 128 bits	1.793	679.162	188.698	181.51
🟩 192 bits	1.835	1065.36	196.355	312.236
🟦 256 bits	1.863	1074.77	202.484	319.917

Fig. 7. Average AES encryption execution time while varying key size for different crypto libraries



	PyCryptodome	GnuPG	OpenSSL	wolfSSL
🟥 128 bits	1.675	659.423	181.405	178.202
🟩 192 bits	1.7	943.318	186.811	310.644
🟦 256 bits	1.719	946.437	197.404	313.397

Fig. 8. Average AES decryption execution time while varying key size for different crypto libraries

execution times of the PyCryptodome library increased while increasing the plaintext size, the slope of this increase was much lower compared to the other evaluated crypto libraries for different plaintext sizes. In addition, PyCryptodome is still the best performing in terms of execution time across different plaintext sizes.

Moreover, we observed that wolfSSL's CBC mode of encryption seems to fail for plaintext sizes greater than 1,024 bytes across all tested devices. We have reported this observation to their software team. As a result, a heap overflow bug was fixed in their GitHub repository[1].

C. Impact of key size on the execution time

To evaluate the impact of the key size used on the execution time, we fixed the plaintext size at 512 bytes and varied the key sizes to obtain the results on a Raspberry Pi Zero W device. The number of iterations was set to 2,000 for each crypto operation and the geometric mean is calculated and reported.

Figures 7 and 8 show the average execution time while varying the key size for encryption and decryption operations, respectively. For all four evaluated crypto libraries, the obtained results show very little variation in execution times across key sizes. This behavior can be attributed to the AES algorithm's translation of key sizes to the number of rounds. In the AES algorithm, this translation occurs as follows: key sizes 128, 192, and 256 are mapped to the number of rounds 10, 12, and 14, respectively.

[1]commit c01a15e1cdffac5c0555f42d9b4be0878cf0d31d [22].

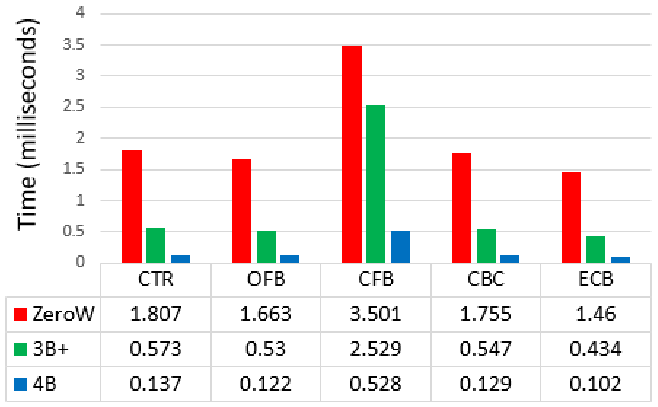
D. Impact of hardware resources on the execution time

In this set of experiments, plaintext size was fixed at 1,024 bytes and key size at 256 bits for all the crypto operations. We fixed the number of iterations at 30,000 for all experiments and reported the average values. Figures 9 and 10 show the encryption execution time and energy respectively of each cipher block mode of PyCryptodome crypto library running on each of the three Pi devices. The decryption execution times and energies are similar to their encryption counterparts; thus, we show only the encryption results due to the limited space.

The study documented in [23] and [24] have shown the dual benefits of higher security and lower computational overhead of CTR mode as compared to other block cipher modes, making it an ideal candidate for symmetric crypto operations on resource-constrained IoT devices. Since our target was to obtain maximum possible security along with maximum utilization of hardware resources, our results showed that this goal was achievable at a slightly higher expense of execution times/energy as compared to the remaining secure and efficient block cipher modes (OFB and CBC). On the contrary, our results showed that the benefits in smaller form factors of hardware configurations are reduced significantly by their higher execution times and energy consumption for the same test settings. On further investigation, we found Zero W, 3B+, and 4B models to operate at nearly the same voltages (4.9, 5.1, and 4.9 Volts, respectively). But their operating currents were different (0.3, 0.6, and 0.7 Amps, respectively).

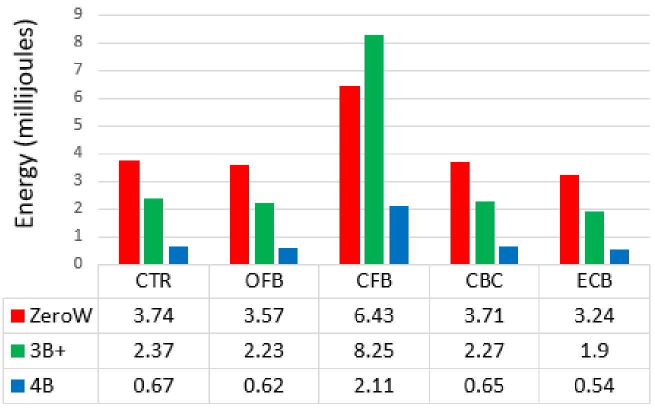
E. Impact of block cipher modes on ciphertext generation

The results shown in Table VII were obtained on an *Intel® Core™ i7-6700 CPU @ 3.40GHz × 8* desktop machine



	CTR	OFB	CFB	CBC	ECB
🟥 ZeroW	1.807	1.663	3.501	1.755	1.46
🟩 3B+	0.573	0.53	2.529	0.547	0.434
🟦 4B	0.137	0.122	0.528	0.129	0.102

Fig. 9. Encryption Times of AES-256-[Mode]



	CTR	OFB	CFB	CBC	ECB
🟥 ZeroW	3.74	3.57	6.43	3.71	3.24
🟩 3B+	2.37	2.23	8.25	2.27	1.9
🟦 4B	0.67	0.62	2.11	0.65	0.54

Fig. 10. Encryption Energies of AES-256-[Mode]

running the *Ubuntu 20.04.2 LTS* operating system. The reason for this switch was to overcome lower processing overheads on IoT devices and to focus only on the crypto libraries' behavior, which is independent of the underlying hardware target.

For a plaintext size of 16 bytes and key size of 16 bytes, we found different crypto libraries showing a variation in ciphertext lengths for a sample number of 3 consecutive iterations. Overall, the PyCryptodome library showed consistency and predictability across consecutive iterations for each block cipher mode of encryption. Excluding CBC mode, the ciphertext lengths for the remaining modes appear to be the least for the PyCryptodome library. This observation helped us to conclude that this library will consume the least energy during the propagation of encrypted data across a communication network. The ciphertext analysis of different block cipher modes according to [25] shows the PyCryptodome library to comply with their expected lengths unlike the other crypto libraries, thus increasing its reliability of implementation.

IV. Discussion and Future Work

In this paper, we confined our results to the microprocessor class of IoT devices and widely used C language-based crypto libraries. In follow-up work, we plan to run similar experiments on the microcontroller class of devices

TABLE VII
Ciphertext Variation w.r.t. Block Cipher Modes

Crypto Library	**Block Cipher Mode**				
	ECB	**CBC**	**CFB**	**OFB**	**CTR**
PyCryptodome	33,33,33	48,48,48	16,16,16	16,16,16	16,16,16
GnuPG	NA	NA	63,57,60	NA	NA
OpenSSL	33,33,33	33,33,39	21,27,24	24,21,24	21,21,21
wolfSSL	NA	27,27,33	NA	NA	24,21,21

using lightweight ciphers (LWC), RTOSs, and our preferred symmetric algorithm AES. As an initial step, we analyzed the features of a prominent NIST LWC Round 2 candidate *SAEAES* as listed in [26]. Some of its characteristics are well suited for the widely used microcontroller-based Arduino devices. This algorithm requires small ROM and RAM sizes. The same algorithm structure exists for the hash, encryption, and decryption functions. In addition, only one buffer of size 16 bytes is needed for storing the internal state. Authentication algorithms written as assembly language subroutines are callable from the C language. Two implementations exist for this algorithm: fast speed and small memory size. Also, it takes a constant number of cycles for fixed data length and memory consumption of the outer LWC routine is much smaller than that of the inner AES routine.

While surveying possible candidates for LWC algorithms for our future experiments, we intend to focus on algorithms that adhere to the guidelines prescribed in [27], [28], and [29]. In particular, we will focus on algorithms that tend to use a single S-Box architecture as their results show this step to be the most power-hungry step of the AES algorithm.

In summary, we plan to split our future characterization

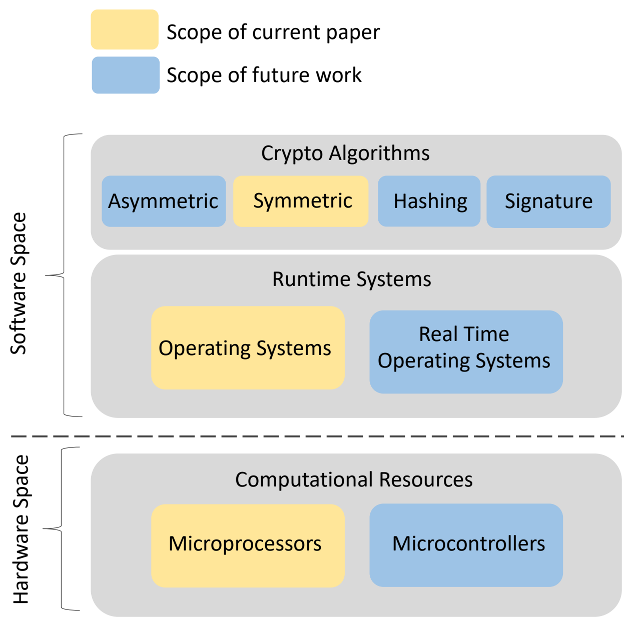


Fig. 11. Breakdown of our characterization process.

results across different levels, with each level being further split into different blocks as shown in Figure 11. Our ultimate goal is to evaluate both asymmetric and symmetric crypto algorithms on resource-constrained IoT devices since both have their own advantages and disadvantages and are usually combined in practice to transmit large chunks of information. Usually, an asymmetric algorithm is used to store or transmit the symmetric key, while the actual large-sized messages are encrypted or decrypted using a symmetric algorithm. Post completion of symmetric cryptography analysis w.r.t. AES, we plan to perform experiments along similar lines on a widely used asymmetric cipher RSA. We would then combine our results on symmetric and asymmetric ciphers to derive a hybrid solution following some of the guidelines meant for creating such customized solutions as stated in [30]. This would help us maximize energy and latency benefits in a diverse landscape of resource-constrained IoT devices.

V. Conclusion

From our experimental results, we conclude that the **Py-Cryptodome** crypto library proves to be a promising candidate in terms of execution time and energy on IoT devices capable of running any Linux-based operating system. The major drawback of this library appears to be the absence of a simplified user interface via the Linux command-line to allow users to readily test out various input combinations and verify results on the fly. This barrier was overcome by developing our custom wrapper software to interface with their GitHub code. In the future, we plan to open-source our implementation upon successful industrial deployment.

VI. acknowledgement

This material is based on research sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement number HR001120C0154. The views, opinions, and/or findings contained in this article are those of the author(s) and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

References

[1] M. A. Mahmud, K. Bates, T. Wood, A. Abdelgawad, and K. Yelamarthi, “A complete internet of things (IoT) platform for structural health monitoring (shm),” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, 2018, pp. 275–279.

[2] C. Coelho, D. Coelho, and M. Wolf, “An IoT smart home architecture for long-term care of people with special needs,” in *IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 626–627.

[3] C. Panait and D. Dragomir, “Measuring the performance and energy consumption of AES in wireless sensor networks,” in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2015, pp. 1261–1266.

[4] J. Raigoza and K. Jituri, “Evaluating performance of symmetric encryption algorithms,” in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1378–1379.

[5] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, “Analyzing the energy consumption of security protocols,” in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 30–35.

[6] M. M. N. Biasizzo, M. Mali, and F. Novak, “Hardware implementation of AES algorithm,” *Journal of Electrical Engineering*, vol. 56, no. 9-10, pp. 265–269, 2005.

[7] A. Hodjat and I. Verbauwhede, “Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors,” *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 366–372, 2006.

[8] J. Bahrami, V. B. Dang, A. Abdulgadir, K. N. Khasawneh, J.-P. Kaps, and K. Gaj, “Lightweight implementation of the lowmc block cipher protected against side-channel attacks,” in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, 2020, pp. 45–56.

[9] W. Diehl, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, “Comparison of hardware and software implementations of selected lightweight block ciphers,” in *27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[10] C. H. Lim and T. Korkishko, “mCrypton–a lightweight block cipher for security of low-cost RFID tags and sensors,” in *International workshop on information security applications*. Springer, 2005, pp. 243–258.

[11] I. K. Dutta, B. Ghosh, and M. Bayoumi, “Lightweight cryptography for internet of insecure things: A survey,” in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0475–0481.

[12] B. Buchanan, “Research team crack light-weight crypto,” 2018. [Online]. Available: https://www.linkedin.com/pulse/reseach-team-crack-light-weight-crypto-just-listening-prof-bill/

[13] K. Zhang, L. Ding, and J. Guan, “Cryptanalysis of Hummingbird-2,” *IACR Crypt. ePrint Arc*, vol. 2012, 2012.

[14] “Raspberry Pi Products.” [Online]. Available: https://www.raspberrypi.com/products

[15] “Raspberry Pi Comparison.” [Online]. Available: https://www.digikey.com/en/maker/blogs/2018/how-to-pick-the-right-raspberry-pi

[16] V. G. Reddy, “Neon technology introduction,” *ARM Corporation*, vol. 4, no. 1, 2008.

[17] “PyCryptodome.” [Online]. Available: https://www.pycryptodome.org/en/latest/index.html

[18] “GnuPG.” [Online]. Available: https://www.gnupg.org/index.html

[19] “OpenSSL.” [Online]. Available: https://www.openssl.org

[20] “wolfSSL.” [Online]. Available: https://www.wolfssl.com/

[21] “MakerHawk UM25C USB Tester.” [Online]. Available: https://www.makerhawk.com/

[22] “rework and fix evp crypto return value, fix heap overflow issue, add … 52.” [Online]. Available: https://github.com/wolfSSL/wolfCLU/pull/52

[23] D. Bujari and E. Aribas, “Comparative analysis of block cipher modes of operation,” in *International Advanced Researches & Engineering Congress*, 2017, pp. 1–4.

[24] H. Lipmaa, P. Rogaway, and D. A. Wagner, “Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption,” 2000.

[25] B. Schneier, “Applied cryptography, second edition : protocols, algorithms,and source code in c,” 2015.

[26] “NIST LWC.” [Online]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography

[27] A. Dogan, S. B. Ors, and G. Saldamli, “Analyzing and comparing the AES architectures for their power consumption,” *Journal of Intelligent Manufacturing*, vol. 25, no. 2, pp. 263–271, 2014.

[28] J.-P. E. Kaps, “Cryptography for ultra-low power devices,” Ph.D. dissertation, Worcester Polytechnic Institute, 2006.

[29] K. Gaj and P. Chodowiec, “FPGA and ASIC implementations of AES,” in *Cryptographic Engineering*. Springer, 2009, pp. 235–294.

[30] K. McKay, L. Bassham, M. Sönmez Turan, and N. Mouha, “Report on lightweight cryptography,” National Institute of Standards and Technology, Tech. Rep., 2016.