Lightweight Security Algorithms for Resource-constrained IoT-based Sensor Nodes

Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen and Tomi Westerlund

Turku Intelligent Embedded and Robotics Systems (TIERS),
Department of Future Technologies, University of Turku, Turku, Finland.
Email: {vikasar, tunggi, hatenhu, tovewe}@utu.fi,

***Abstract*—With the constant improvement of electronics and development by research community, professionals and enthusiasts around the world, Internet of Things (IoT) based devices have seen a massive increase. These devices are now connected to our daily life in multiple ways and facilitate smooth operation of large, autonomous and semi-autonomous systems in different sectors. The communication among these systems needs to be done in a secure manner. However, as most of the IoT devices have very limited processing capability and energy source, all cryptography algorithms are not able to run on all devices. In addition, depending on the required data performance, it can be desirable to use one specific type of algorithm over others. In this paper, we analyze popularly used lightweight algorithms in terms of operational latency by running them on multiple widely used embedded modules. In addition, we measure power consumption while running an algorithm to realize its impact on battery life as an example. Finally, we discuss design-time considerations to help designers to select an appropriate cryptography algorithm for different applications.**

***Index Terms*—Lightweight, Security, Algorithms, Internet of Things, IoT, Sensor, Nodes, Latency, Efficiency, AES, ChaCha, CTR, SHA, SHA3, BLAKE, SHAKE.**

I. Introduction

The Internet of Things (IoT) has essentially extended the use of Internet among smart, embedded and wide variety of cyber-physical systems. These are so widely used for data collection, monitoring and controlling purpose in numerous application fields that they are deeply integrated in our daily life [1]. Such devices communicate with local network or the Internet and also with each other. Since these often collect information which is sensitive and in some cases private, the transfer process of data must be secure [2]. In IoT applications, security is always one of the most important requirements because an unsecured IoT can cause serious consequences including but not limited to personal injury and damage of property due to unauthorized access [3]. Most importantly, sensitive data can fall into wrong hands resulting in breach of privacy [4]. For example, researchers demonstrated that some certain insulin pumps can be hacked by unauthorized third party to gain control with a bad intention [5]. In order to avoid such security related problems, end-to-end security must be implemented in IoT applications. In particular, architectural layers such as sensor layer, gateway layer, Cloud layer and application layer within the IoT-based system must incorporate security measures to protect data and secure the system [6]. For this purpose, cryptography algorithms are used to authenticate permitted users to provide access and encrypt data while transfer takes place.

The embedded devices and electronics involved in IoT are resource-constrained devices which are mostly limited in terms of processing capability, physical memory and power supply. Particularly, the sensor nodes at the lowest architectural level of the system cannot run the original variants of cryptography algorithms because those require heavy resources for complex computations [7]. Furthermore, the sensor nodes are often required to run for prolonged period of time to make it useful in particular applications such as health monitoring [8]. Considering these, it is not possible to run the original, processing-heavy algorithms on embedded devices and hence lightweight versions of cryptography algorithms were developed. These are optimized for the low-resource embedded systems to comply with the processing capability and latency requirements while fulfilling the security requirements. However, not all of them are equally suitable for the sensor layer [9]. Specifically, from one application to another, the suitability of a particular algorithm varies due to operational performance, complexity, processing and latency requirements and level of security it provides. In this paper, a systematic approach is taken to run widely used cryptography algorithms on preset data to measure operational latency. Additionally, we discuss the appropriateness of algorithms when system resources are limited and suggest considerations for optimal performance, energy-efficiency and application requirements. The specific contribution of this work is as follows:

- Evaluate run-time performance of widely used cryptography algorithms
- Discuss appropriateness of stated algorithms for different applications
- Provide guidelines and mention aspects to consider for optimized performance and energy efficiency

The rest of the document is arranged as follows: Section II gives an overview of our experimental methodology, Section III enumerates different lightweight cryptography algorithms used in IoT-based devices or sensor nodes and illustrates the test results, and Section IV discusses limitations and suggests possible improvisation methods. Finally, Section V concludes this paper and provides directions for future work.

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

II. Methodology

Many cryptography algorithms have been proposed for providing security, authenticating or encrypting payload data in a communication. However, not all of the algorithms can be applied in resource-constrained devices (e.g. sensor nodes in IoT systems). When employing cryptography algorithms, it is of utmost importance to know what a hardware platform offers and how well a specific algorithm runs on it. To test the performance and associated advantages and disadvantages of the lightweight cryptography algorithms, we have selected four widely used embedded devices which have been used as controllers for sensor nodes in many applications. These are Arduino Pro-Mini (Pro-mini), Arduino Uno (Uno), ESP-WROOM-32 (ESP32) and ESP8266MOD (ESP8266).

The first two are based on the Microchip *ATmega328P* [10] micro-controller unit (MCU) conforming to Harvard architecture. The modules support 8-bit operation and run at 8 MHz and 16 MHz clock frequency, and are supplied 3.3 V and 5 V, respectively. The ESP32 and the ESP8266 are 32-bit processing capable and are based on the Xtensa dual-core *LX6* microprocessor [11], and the *L106* RISC microprocessor core constructed upon the Tensilica Xtensa Diamond Standard *106Micro* [12], respectively. Although the *LX6* has two processing cores, a single core of ESP32 is utilized in our experiments because most of the lightweight cryptography algorithms are designed for a single core. The processor clock frequency, bus width, RAM and flash memory specifications for the aforementioned modules are listed in Table-I.

In the latency tests, an array of 16 bytes was used as the plain-text input. The algorithms are compiled in Arduino development environment based on the *Crypto* library [13]. In addition, for measuring the latency period, Arduino *micros()* function [14] is used which depends on the hardware timer. The measurement accuracy relies on the resolution of the underlying hardware timer in the MCU. For example, on an Arduino Uno running at 16 MHz, the resolution is 4 us. To get a more accurate latency measurement, each algorithm was run 500 times and the average latency values are presented here. In addition, we have conducted energy measurements for certain algorithms to provide insight on how those relate to variation in latency. During the tests, we have used a professional power monitor from Monsoon Inc. [15] which allows fine grained and accurate measurement of average current and power along with battery life estimation.

III. Experimental Results

This section briefly enumerates AES, CTR, ChaCha, ChaChaPoly, SHA and BLAKE2 algorithms and shows their run-time performance on our test modules. Also, the results, respective impacts and their causes are discussed, and possible considerations when using these are suggested.

A. Advanced Encryption Standard

Advanced Encryption Standard (AES) is one of the most widely used algorithms in embedded systems for encrypting

TABLE I
Specification of the embedded devices used in the experiments

Module / MCU	Bus Width (bit)	Clock (MHz)	RAM (KB)	Flash (KB)
Arduino Pro-Mini (Atmel ATmega328P)	8	8	2	32
Arduino Uno (Atmel ATmega328P)	8	16	2	32
ESP32 (ESP-WROOM-32) (Xtensa LX6 dual-core)	32	240	520	4096
ESP8266 (ESP8266MOD) (Xtensa Diamond 106Micro)	32	80	80	4096

blocks of data during transfer [16]. Among other AES variants, we have used the Electronic Codebook (ECB) in our experiment due to their linearity and reduced complexity. The AES operation happens in two stages; (1) the key setup and (2) the encryption/decryption. The key setup time for 128-bit, 192-bit and 256-bit keys on our test modules is given in Table-II. In addition, the time required for a single byte in the second stage of the AES algorithm, encryption/decryption is listed. It can be observed that the *ATmega328P* running at 8 MHz takes the maximum time and ESP32 takes the shortest time as expected. Although the ESP8266 runs at only 3 times slower clock than the ESP32, the latter takes significantly shorter time due to its dedicated cryptography hardware accelerator.

We can see close encrypt/decrypt timings among AES variants when run on the same module. In the setup phase, 32-bit modules dramatically outperform the 8-bit counterparts due to ample amount of available RAM. However, the difference between the 32-bit modules is noticeably smaller during encryption/decryption since it only involves static-length data during different rounds.

TABLE II
Key setup, Encryption and Decryption period of AES algorithm in *us*

Operation	Algorithm	Pro-mini	Uno	ESP32	ESP8266
Set Key	AES-128-ECB	317,14	158,57	0,52	35,03
	AES-192-ECB	330,47	165,24	0,54	33,69
	AES-256-ECB	411,91	211,45	0,57	44,36
Encrypt	AES-128-ECB	66,61	33,31	0,38	6,41
	AES-192-ECB	79,96	39,98	0,39	7,69
	AES-256-ECB	93,3	46,65	0,41	8,98
Decrypt	AES-128-ECB	144,3	72,15	0,38	9,16
	AES-192-ECB	174,89	87,44	0,39	11,05
	AES-256-ECB	205,48	102,74	0,41	12,94

B. CTR

Block ciphers ensure that when the amount of data is large, the produced cipher-text after encryption is different in each round of operation even if the original data or plain-text is the same. CTR is an operational mode of block ciphers in which a block cipher is essentially converted into a stream cipher using key-stream generated from successive values of a single or multi-step counter [17]. However, it is weak against manipulation attack and data cannot be recovered fully from

a corrupted stream of data due to chaining, i.e., dependency on previous data during key generation. Figure-1 shows the time CTR mode takes to complete. It can be noticed that the ESP32 performs significantly faster due to its dedicated AES block in the CPU core.

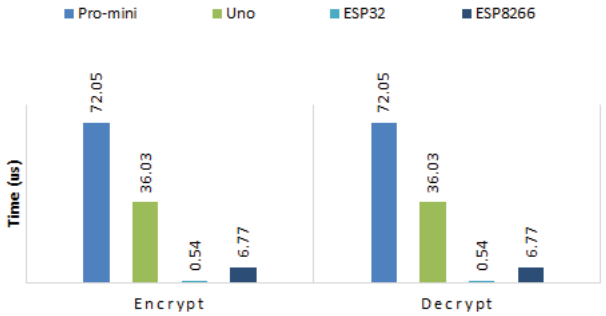

Fig. 1. Run-time performance of CTR algorithm.

TABLE III
KEY SETUP, ENCRYPTION AND DECRYPTION PERIOD OF CHACHA ALGORITHM IN *us*

Operation	Algorithm	Pro-mini	Uno	ESP32	ESP8266
SetKey	ChaCha8 128	86,51	43,08	1,26	8,7
	ChaCha8 256	84,64	42,13	1,09	4,16
	ChaCha12 128	86,51	43,09	1,26	8,7
	ChaCha12 256	84,62	42,13	1,09	4,16
	ChaCha20 128	86,38	43,08	1,26	8,71
	ChaCha20 256	84,63	42,13	1,09	4,16
Encrypt	ChaCha8 128	18,48	9,24	0,17	0,41
	ChaCha8 256	18,48	9,24	0,17	0,41
	ChaCha12 128	24	12	0,19	0,48
	ChaCha12 256	24	12	0,19	0,48
	ChaCha20 128	35,03	17,52	0,24	0,62
	ChaCha20 256	35,03	17,52	0,24	0,62
Decrypt	ChaCha8 128	18,49	9,25	0,17	0,42
	ChaCha8 256	18,49	9,25	0,17	0,42
	ChaCha12 128	24,01	12,01	0,19	0,48
	ChaCha12 256	24,01	12,01	0,19	0,48
	ChaCha20 128	35,05	17,52	0,24	0,62
	ChaCha20 256	35,05	17,52	0,24	0,62

C. ChaCha

In IoT systems when real-time data is transferred as a stream or as serial bytes of data, the overhead of the block ciphers can be problematic for ensuring smooth, flexible user experience. Consequently, the stream ciphers were developed to help reduce the total encryption and decryption latency. ChaCha is one of the most widely used stream ciphers due to its lower operational latency. Use of ChaCha20 [18] was proposed by *Google* as it provides better security to avoid cache-collision attacks and yields almost three times faster operation. ChaCha incorporates key stream which goes through X-OR operation with the plain-text data and is not vulnerable to timing based attacks. The numbers 8, 12 and 20 indicate the rounds of encryption operations run on a single data unit.

Table-III shows that ChaCha is approximately four times faster than similar AES variants in terms of key setup. Interestingly, ChaCha 256-bit variants runs faster on the 32-bit modules than the respective 128-bit ones due to fewer operations during the *quarter* round. Besides, there is very little variation in key setup times among the 128-bit and 256-bit variants of ChaCha8, ChaCha12 and ChaCha20. Table-III also lists the latency of encryption/decryption when running on our test modules. Although the timings are comparatively lower than the AES, the 32-bit modules perform the best. However, between two 32-bit modules, the performance difference is not on par with the 4 times faster ESP32 module.

D. ChaChaPoly

Modern IoT-based sensor nodes frequently send data to and receive control commands from a host directly or through another node. Consequently, large amounts of data packets are transmitted among nodes, especially when the number of nodes increases and data rate is high. It is critical to ensure the authenticity of such information packets and hence Authenticated Encryption with Associated Data (AEAD) algorithm ChaChaPoly was developed. It is widely used in transferring data to the Internet and is based on ChaCha20 and Poly1305 Message Authentication Code (MAC) algorithms [18]. Figure-2 shows the run-time duration of ChaChaPoly. It can be clearly seen that the 32-bit modules drastically outperform the resource-limited 8-bit ones. Unlike AES which uses look-up tables to compute *mixcolumn* operations, ChaCha uses built-in CPU instructions for encryption. This helps to achieve dramatically higher performance while being CPU-friendly and platform independent in general.

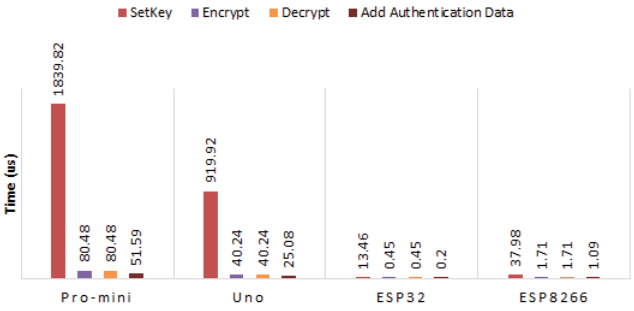

Fig. 2. Run-time performance of ChaChaPoly algorithm.

E. SHA

A cryptography hash function converts an arbitrary sized data into a predefined data set of fixed length. This is useful if there is a restricted or fixed-width payload capacity when data is transferred from one sensor node to another. Secure Hash Algorithm (SHA) is designed to perform this while hiding the actual data. A very small change in the original data can yield a dramatic change in the output [19]. In embedded systems, SHA algorithms are used to securely transfer and validate the integrity of data. SHA256 and SHA512 are computed using 32 and 64 bytes, and involve 64 and 80 rounds of operation on the data, respectively. Figure-3 and Figure-4 show the run-time latency of SHA256 and SHA512 algorithms, respectively. As the hash functions are originally targeted at

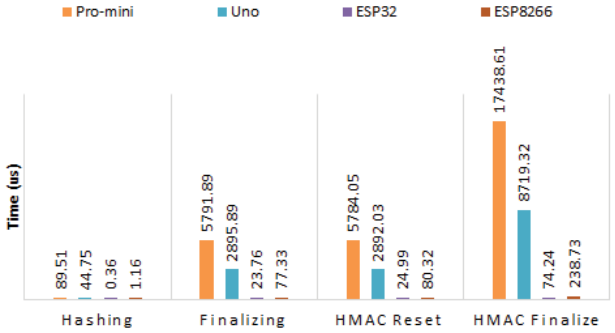

Fig. 3. Run-time performance of SHA256 algorithm

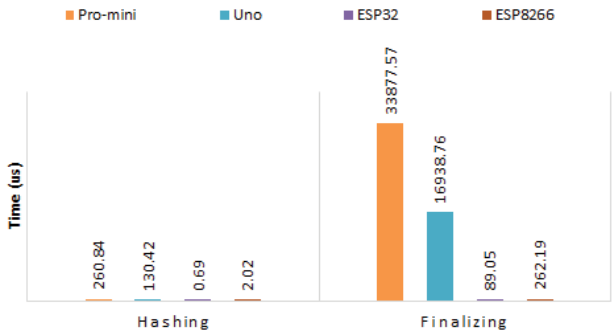

Fig. 4. Run-time performance of SHA-512 algorithm.

32-bit and 64-bit architectures, on 8-bit MCUs, these run at a drastically slower speed due to limited resources and division of operation yielding more cycles. In contrast, the 32-bit modules handle both hashing and finalizing operations well. However, the SHA512 takes longer time due to double hash size and extra rounds of operation than SHA256. Figure-3 also shows time taken for reset and finalizing operation by SHA256 variant of Hash-based Message Authentication Code (HMAC). HMAC depends on the underlying hash function to make an authentication token without actually encrypting the original message token. The receiver can verify the data integrity from hashing it and then matching with the additionally provided HMAC token by the sender. Interestingly, HMAC scales well with the clock frequency on both 8-bit and 32-bit modules in our test.

Although SHA-2 provides stronger encryption, however, it is vulnerable to length extension attack as it directly uses the hash generated from the last round to generate next hash token. For this reason, SHA-3 algorithm [20] was developed which uses modified hash for the consequent key generation and provides better performance on 32-bit and 64-bit processor architectures. Figure-5 shows the run-time performance of SHA3-256 and SHA3-512 algorithm. The time taken by SHA3-256 hashing process is approximately half of SHA3-512 due to key length. However, there is very little time variation during data finalizing rounds on the same module, respectively. The 8-bit modules suffer since data is read and written in chunks from the flash to the RAM, and vice-versa.

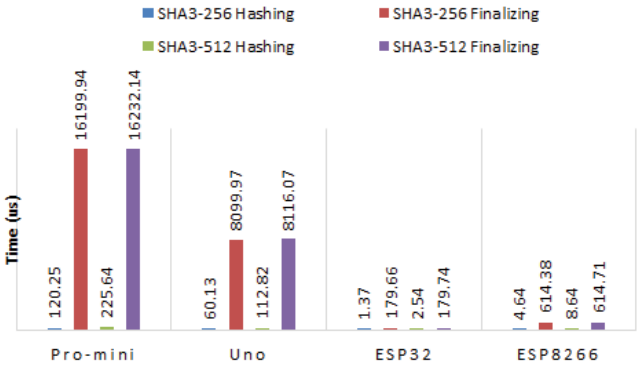

Fig. 5. Run-time performance of SHA3-256 and SHA3-512 algorithm.

F. BLAKE2

As cryptography functions involve many mathematical calculations, optimization of existing algorithms is desirable for improved performance and security. Cryptography hashing function BLAKE was developed ground-up from ChaCha by J-P. Aumasson *et al.* to outperform SHA algorithms. Later, an improved version called BLAKE2 [21] eliminated the security risks of the earlier compromised MD5 and SHA-1. On a 64-bit computing system BLAKE2b shows improved performance compared to SHA2 and SHA3 algorithms.

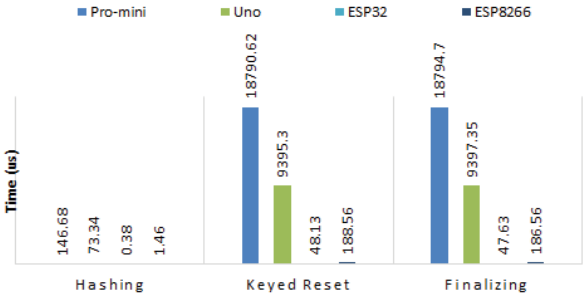

Fig. 6. Run-time performance of BLAKE2b algorithm.

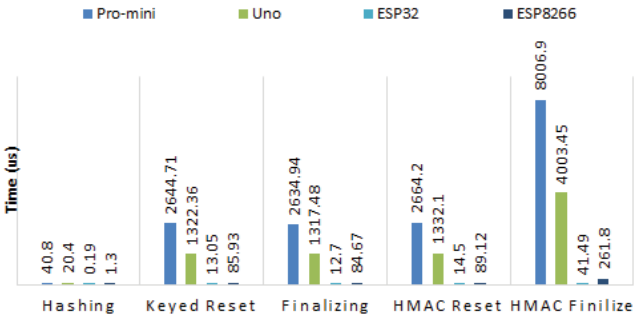

Fig. 7. Run-time performance of BLAKE2s algorithm.

In contrast, BLAKE2s is optimized for 8-bit and 32-bit processor platforms making it more suitable for embedded systems. Figure-6 and Figure-7 respectively show the run-time performance of BLAKE2b and BLAKE2s variants of

BLAKE2 on our test modules. From the figures, it can be seen that the BLAKE2s hashing, reset and finalizing process run significantly faster compared do BLAKE2b due to optimization by design. The ESP8266 and ESP32 modules are quicker due to higher clock and wider data width. Additionally, BLAKE2s support HMAC and ESP32 should be used if an application employs stricter latency budget.

G. SHAKE

Cryptography hashing functions work in a one-way manner and turn an arbitrary length of data to a fixed length content. While a specific algorithm uses a fixed, pre-defined length of data as input to next stage, the content length can vary among algorithms. Extendable-Output Functions (XOF) [20] are used to convert an arbitrary length of content to a dynamically sized digest output. This facilitates use of same data in multiple algorithms just by changing the output length of the hashing process and helps optimize the encryption or decryption process. Figure-8 shows run-time performance of two XOF- SHAKE128 and SHAKE256 algorithms. It can be observed that both run fast even on the slowest 8-bit module at 8 MHz, and would not add burden to the MCU.

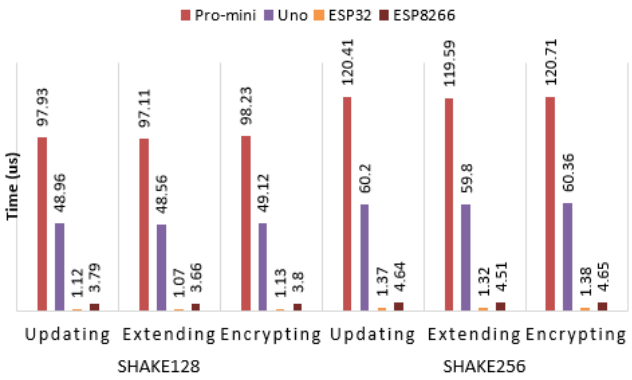

Fig. 8. Run-time performance of SHAKE128 and SHAKE256 algorithm.

IV. Discussion

Depending on the algorithm's usage of memory due to parameters such as number of processing rounds and key size, the run-time latency can vary dramatically. If the available RAM is low due to hardware design itself and memory used by other processes, cryptography algorithms can take long time to complete due to the slower operation of flash memory used instead to compensate the shortage of RAM. For example, the initial key setup time is high for AES, ChaCha and finalizing process in SHA algorithms running on the *ATmega328P*'s limited 2 KB RAM [10]. Therefore, to achieve optimum performance, it is highly recommended to consider having a MCU with sufficient RAM based on the algorithm and number of other tasks or processes which will occupy the RAM.

Many applications require sensor nodes to send varying amount of data to the host or to another node. One algorithm can be more efficient over another for a particular range of packet sizes for a message. Depending on the application and

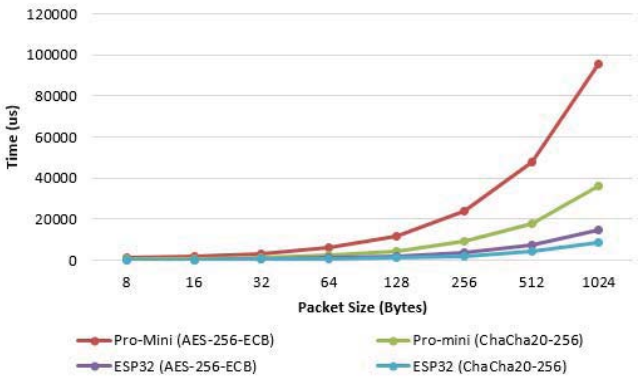

Fig. 9. Performance of AES and ChaCha with varying data packet size.

mode of power to the sensor node, it is important to consider how latency and size of data packets relate with each other. This can help to optimize total run-time of a node and to select an optimal algorithm. Figure-9 shows how AES-256-ECB and ChaCha20-256 compares in terms of time taken for encrypting data packet when number of bytes varies. During encryption, AES takes 16 byte blocks and ChaCha takes 64 bytes of plain text at a time. In addition, it is assumed that for each packet the encryption key is renewed. Each time the key is changed or renewed, it adds latency overhead to the total time taken for encryption process. It can be observed from Figure-9 that as packet size increases, time taken by AES increases at a much higher rate than ChaCha. Also, as the Pro-mini has limited RAM, the difference between AES and ChaCha is more visible when compared with ESP32. This implies that AES is better suited for infrequent and small amount of data, while ChaCha is more appropriate for streaming data when data packet size is sufficiently small. However, while frequent change or renewal of the key for each data packet provides superior security against brute-force attacks, if the application requirements allow, the key can also be renewed at a larger interval which can lower the total encryption latency.

On the other hand, time taken to encrypt or decrypt a single byte and initial key setup time can vary on 8-bit and 32-bit wide MCUs. For example, when the AES128, AES192, AES256 algorithm is run on the 8-bit *ATmega328P* and 32-bit *106Micro* processor, variations are observed. Since the former one is run at 8 MHz (in Pro-mini) and the latter one at 80 MHz, to have fair performance comparison, the architecture width and clock frequency is taken into account. For this reason, the latency of the *ATmega328P* is compared to 10 times slower and 4 times less architecture width of the *106Micro* processor which is shown in Figure-10. The graph shows that clock for clock, the 8-bit MCU takes less time to run the algorithm. According to Atmel, particular low-power systems can benefit from optimization of algorithms, dedicated hardware features and using an 8-bit MCU instead of a 32-bit variant [22].

Since cryptography algorithms involve complex and repetitive mathematical operations, it requires significant computation. Depending on the selected embedded module, the supply requirements can vary due to difference in run-time duration

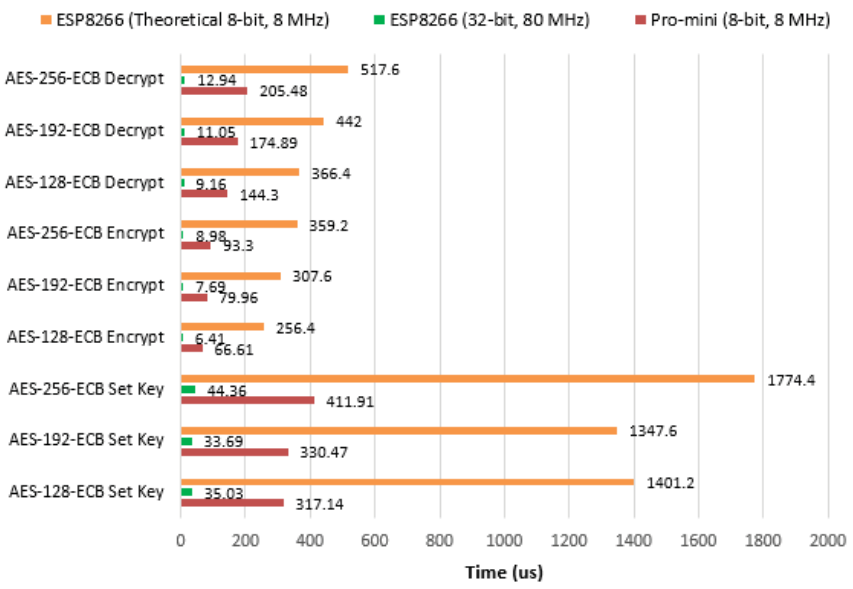

Fig. 10. Comparison of 8-bit Vs. 32-bit performance for AES algorithm.

while processing data. As an example, we have measured the energy requirement of the test devices used in this paper when stream cipher ChaCha is run. Table-IV enlists the supply voltage, average power consumption, average current draw and respective battery life. Since the modules are tested as-is and contain extra components on-board such as USB-TTL serial converter IC, it increases the values slightly. However, these give an overview of supply requirements when the algorithm is run. Designers must consider these parameters if a sensor node is run on battery power and is desired to work continuously for long time.

TABLE IV
POWER REQUIREMENTS OF TEST DEVICES WHEN CHACHA IS RUN

Module / Parameter	Pro-mini	Uno	ESP32	ESP8266
Supply Voltage (V)	3.3	5.0	3.3	3.3
Avg. Current (mA)	2.72	6.02	53.89	79.31
Avg. Power (mW)	12.23	30.94	177.2	260.7
Battery Life (hours) (1000 mAh capacity)	367.6	166.1	18.6	12.6

In recent years, the number of connected sensor nodes have been increasing in different application fields where real-time data is collected and then sent as a serial stream to the next level of the hierarchy such as a nearby gateway or another node connected to the Internet [23] [24]. Several factors need to be considered when designing a sensor node which requires security measures. In general, clock frequency, available memory (RAM), encryption latency, transmission bandwidth, confidentiality level of data, use of in-RAM data buffer, and utilization of independent hardware buffer for transmission play important roles and are related to each other. Figure-11

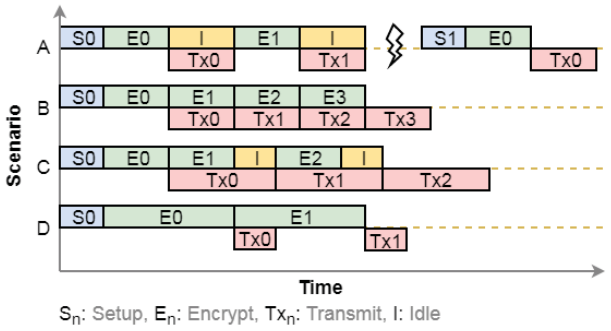

Fig. 11. Different operational scenarios in a sensor node when using cryptography algorithms and involving data transmission.

illustrates few time-wise operational scenarios (A, B, C and D) for a sensor node. In A and B, the per-byte encryption time is very close to the transmission period which is desired when choosing an encryption algorithm. The encryption key is setup at the beginning, then the data is encrypted and the CPU is idle while the transmission completes. This idle time can be utilized for other operations, or if there is none, the CPU can be put into sleep mode to save energy. After a certain interval, if required, the encryption key is updated and then the process repeats. This is optimal for low data rates and infrequent data transfer when the MCU resources are extremely limited. In B, the process is essentially similar except that the CPU can start encrypting the next byte after handing over the previous byte to the hardware serial buffer. This way the CPU is not idle and more data can be processed and sent in an interleaved manner. This is suitable for medium data rates. However, when considering other parameters such as energy efficiency, transmission bandwidth and sampling rate of data, often the transmission period and the selected encryption

method's latency is not the same. Embedded system designers must consider all parameters to achieve the best combination such that maximum data is processed upon minimum energy consumption. In C, the per-byte transmission period is longer than the per-byte encryption latency, while in D, it is the opposite. The CPU is idle (in C) after processing the next byte while the previous byte is being sent. In such cases, a hardware buffer, or if the system resources allow, a secure software buffer should be used. Consequently in D, when there is plenty of memory to temporarily store the encrypted data, a much higher baud-rate can be used so that more data can be sent in a single turn. This promotes energy efficiency due to much lower bit-transitional period. Finally, a proper choice of serial interface is also vital to achieve maximum energy-efficiency [25].

V. Conclusion

Cryptography algorithms are widely used in sensor nodes and IoT systems for ensuring security. However, the limited resources of the nodes' underlying embedded electronics require pertinent selection of security algorithms. Keeping this in mind, light-weight cryptography algorithms have been developed specifically targeting the low-power and resource-constrained embedded systems. However, the performance and processing requirements can render one algorithm most suited for a particular application while making it unusable for another. Furthermore, the computational requirements are inversely related to the total run-time of sensor nodes operating from limited battery power. Therefore, it is important to consider an appropriate algorithm for a target platform. In this paper, we have experimented with widely used cryptography algorithms on popular embedded modules revealing their operational latency on 8-bit and 32-bit platforms. We have analyzed the algorithms' suitability for specific scenarios to help system designers make better decisions when choosing a cryptography algorithm. Furthermore, to give an overview of probable battery life, we have also run a test to estimate the energy consumption of the modules while a specific algorithm is run. In future, we plan to investigate more hardware platforms, target specific applications, analyze performance and discuss optimization techniques when two or more lightweight cryptography algorithms are used in combination.

Acknowledgment

This work is supported by Academy of Finland (Grant No. 328755), and the Ulla Tuominen foundation, Finland.

References

[1] C. Gu *et al.* A Wireless Smart Sensor Network based on Multi-function Interferometric Radar Sensors for Structural Health Monitoring. In *IEEE Topical Conference on Wireless Sensors and Sensor Networks*, pages 33–36, Jan. 2012.

[2] D. Bastos *et al.* Internet of Things: A Survey of Technologies and Security Risks in Smart Home and City Environments. In *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–7, Mar. 2018.

[3] Q. F. Hassan. *Security Mechanisms and Technologies for Constrained IoT Devices*, chapter 8. IEEE, 2018.

[4] N. Neshenko *et al.* Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations. *IEEE Communications Surveys Tutorials*, 21(3):2702–2733, 2019.

[5] FDA warns patients and health care providers about potential cybersecurity concerns with certain Medtronic insulin pumps. [Online] Available: https://www.fda.gov/news-events/press-announcements/. Accessed: Aug. 10, 2019.

[6] S. J. Johnston *et al.* Recommendations for securing Internet of Things devices using commodity hardware. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 307–310, Dec. 2016.

[7] N. A. Gunathilake *et al.* Next Generation Lightweight Cryptography for Smart IoT Devices: : Implementation, Challenges and Applications. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 707–710, Apr. 2019.

[8] K. Tsuchiyama and A. Kajiwara. Accident Detection and Health-Monitoring UWB Sensor in Toilet. In *2019 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNet)*, pages 1–4, Jan. 2019.

[9] M. Suárez-Albela *et al.* A Practical Performance Comparison of ECC and RSA for Resource-Constrained IoT Devices. In *2018 Global Internet of Things Summit (GIoTS)*, pages 1–6, Jun. 2018.

[10] Microchip. ATmega328P. [Online] Available: https://www.microchip.com/wwwproducts/en/ATmega328p. Accessed: Jul. 20, 2019.

[11] Cadence Inc. Xtensa LX6 Customizable DPU. [Online] Available: https://mirrobo.ru/wp-content/uploads/2016/11/Cadence_Tensillica_Xtensa_LX6_ds.pdf. Accessed: Aug. 10, 2019.

[12] Cadence Inc. Tensilica Unveils Diamond Standard 106Micro Processor; Smallest Licensable 32-bit Core. [Online] Available: https://ip.cadence.com/news/243/330. Accessed: Aug. 10, 2019.

[13] R. Weatherley. Arduino Cryptography Library. [Online] Available: https://rweather.github.io/arduinolibs/crypto.html. Accessed: Jul. 25, 2019.

[14] Arduino. *micros()*. [Online] Available: https://www.arduino.cc/reference/en/language/functions/time/micros/. Accessed: Jul. 30, 2019.

[15] Power Monitor. https://www.msoon.com/high-voltage-power-monitor, Accessed: Aug. 20, 2019.

[16] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). [Online] Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. Accessed: Jul. 25, 2019.

[17] Network Working Group. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). [Online] Available: https://tools.ietf.org/html/rfc3686. Accessed: Jul. 25, 2019.

[18] Internet Research Task Force (IRTF). ChaCha20 and Poly1305 for IETF Protocols. [Online] Available: https://tools.ietf.org/html/rfc7539. Accessed: Jul. 25, 2019.

[19] Internet Engineering Task Force (IETF). US Secure Hash Algorithms. [Online] Available: https://tools.ietf.org/html/rfc6234. Accessed: Jul. 25, 2019.

[20] National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. [Online] Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. Accessed: Jul. 25, 2019.

[21] M-J. Saarinen and J-P. Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). [Online] Available: https://tools.ietf.org/html/rfc7693. Accessed: Jul. 25, 2019.

[22] K. Saether and I. Fredriksen. Introducing a New Breed of Microcontrollers for 8/16-bit Applications. [Online] Available: http://ww1.microchip.com/downloads/en/DeviceDoc/doc7926.pdf. Accessed: Jul. 25, 2019.

[23] V. K. Sarker *et al.* Portable Multipurpose Bio-signal Acquisition and Wireless Streaming Device for Wearables. In *2017 IEEE Sensors Applications Symposium (SAS)*, pages 1–6, Mar. 2017.

[24] R. Grodi *et al.* Smart Parking: Parking Occupancy Monitoring and Visualization System for Smart Cities. In *SoutheastCon 2016*, pages 1–5, Mar. 2016.

[25] K. Mikhaylov and J. Tervonen. Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers. In *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, May 2012.