Hindawi
Security and Communication Networks
Volume 2017, Article ID 2046735, 16 pages
https://doi.org/10.1155/2017/2046735

Research Article

Performance Evaluation of Cryptographic Algorithms over IoT Platforms and Operating Systems

Geovandro C. C. F. Pereira, Renan C. A. Alves, Felipe L. da Silva, Roberto M. Azevedo, Bruno C. Albertini, and Cíntia B. Margi

Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brazil

Correspondence should be addressed to Geovandro C. C. F. Pereira; geovandro@larc.usp.br

Received 1 May 2017; Accepted 17 July 2017; Published 23 August 2017

Academic Editor: Qing Yang

The deployment of security services over Wireless Sensor Networks (WSN) and IoT devices brings significant processing and energy consumption overheads. These overheads are mainly determined by algorithmic efficiency, quality of implementation, and operating system. Benchmarks of symmetric primitives exist in the literature for WSN platforms but they are mostly focused on single platforms or single operating systems. Moreover, they are not up to date with respect to implementations and/or operating systems versions which had significant progress. Herein, we provide time and energy benchmarks of reference implementations for different platforms and operating systems and analyze their impact. Moreover, we not only give the first benchmark results of symmetric cryptography for the Intel Edison IoT platform but also describe a methodology of how to measure energy consumption on that platform.

1. Introduction

The progressive growth of IoT applications has been broadening the spectrum of transmitted data, bringing an increasing demand of security services like data confidentiality, integrity, and source authentication. However, the attempt to employ security mechanisms that are typical of conventional networks is likely to cause undesirable effects due to hardware-related resource limitations. The most relevant overheads relate to energy consumption and/or increase of communication delays. Another concern is the relatively higher memory consumption that might be aggravated by the device's available memory and the amount of applications running on it. Therefore, one of the main challenges for deploying security mechanisms over WSN and IoT is to minimize the conflict between resource consumption and the desired security [1, 2].

In addition, with the advent of Software Defined Networking (SDN), heterogeneous networking devices can be remotely reconfigured by a central SDN controller on the fly and thus a new trend of a wider range of platforms and communication technologies has emerged, like smart homes, autonomous cars, and many others [3]. A new requirement is thus introduced, and security solutions should take those richer environments into account while still offering acceptable performance and energy footprints among all devices within a network.

It is worth mentioning that many works in the literature concentrate their efforts either only on the evaluation of symmetric cryptographic primitives over a single microcontroller (e.g., the MSP430 embedded in TelosB motes [4, 5] or the ATmega128L embedded in MICAz motes [6]) or even on a single OS [5]. In addition, the performance evaluation is usually done by direct compilation to the target platform without an underlying OS. Even though that approach is interesting for providing a reliable benchmarking in an isolated environment, in real-world applications, an OS is required to implement its own TCP-IP protocol stack where TLS-based solutions are built on top of [7]. In practice, each OS has different energy and memory consumption footprints in addition to the delays introduced in the network communication. In the context of WSN, experiments in the literature were conducted over older versions of the operating systems [8] but many important core features have been

changed since then, and thus those results do not reflect the current state of the art in terms of energy consumption among other performance metrics. One relevant example is the ContikiOS, which has been drastically reformulated recently; its 3.0 version was launched in August 2015 [9]. For example, in our experiments, typical AES operations using the SUPERCOOP implementation [10] over both most recent TinyOS 2.1.2 and ContikiOS 3.0 are 2x faster and 2x less energy consuming than the results reported by Casado and Tsigas [8].

In addition to the operating systems, many algorithm parameters and implementations were improved through time; thus an evaluation and analysis of the combined effect of them are crucial for implementors to take better decisions when security services are required.

In this work, we perform an extensive experimental evaluation of reference implementations of many cryptographic primitives for different security services over real sensor platforms. We also evaluate the influence of the most recent versions of two popular operating systems for WSN, TinyOS 2.1.2 and ContikiOS 3.0. We provide a performance and energy analysis, including a methodology, for important cryptographic operations deployed on the TI TelosB [4] and the Intel Edison [11] platforms. As far as we know, this is the first extensive benchmark of symmetric primitives for the Intel Edison platform. The implementations were obtained mostly from the original authors when possible. We preferred 16-bit implementations for TelosB and 32-bit implementations for the Intel Edison.

The paper is organized as follows. Section 2 discusses related work; Section 3 describes the scenario, platforms, and cryptographic algorithms. In Section 4 the experimental setup and methodology for the benchmarks are detailed. Section 5 introduces the results for the algorithms running over TinyOS on TelosB platform, while Section 6 presents the same evaluation for the ContikiOS. We then perform the analysis for the Intel Edison platform in Section 7. Section 8 concludes this paper and suggests future work.

2. Related Work

In 2006, Law et al. [5] evaluated eight different block ciphers and four modes of encryption over a constrained WSN platform, the Texas Instruments 16-bit RISC-based MSP430F149. They focused only on encryption operations and did not analyze primitives for other security services such as authentication, which is usually required by most of modern applications. In addition, if we think about an IoT application, an operating system will also be running over a microcontroller (like ContikiOS) and it would be interesting to measure the behavior of primitives working along with the operating system. Also, the ciphers and libraries analyzed in [5] are completely outdated, since the work is from almost a decade ago. Today sensor motes became much more energy efficient like the underlying TelosB microcontroller, that is, the MSP430F1611 which consumes a 330 μA current in active mode, while the MSP430F149 draws a nominal current of 420 μA as mentioned in [5].

In 2011, Hyncica et al. [12] evaluated the performance of 15 different block ciphers on three different microcontrollers (of 32-bit, 16-bit, and 8-bit instructions). They used the TomCrypt LTC library version 1.16 which is a general purpose cryptographic library for 32-bit platforms and thus not optimized for constrained microcontrollers. They adopted the ECB encryption mode, which allows for a clear distinction of the performance of the plain block ciphers when compared to other modes of operation, for example, CTR, CBC, and CCM. On the other hand, the ECB mode only targets confidentiality service and is considered insecure in practice today [13]. In addition to confidentiality, modern real-world secure applications must include more advanced security services such as authentication. Therefore, a performance evaluation over constrained platforms of more sophisticated modes like authenticated encryption is relevant since they present significant impact on energy consumption compared to ECB. Additionally, Hyncica et al. [12] do not provide energy measurements, which is very important in the context of battery-powered devices. They also do not address the performance behavior when operating systems also play a role.

In 2010, Margi et al. analyzed the impact of the operating systems on security applications for a single platform, that is, the TelosB. Their work is based on TinyOS 2.0.2 and Contiki 2.3. In 2013, Simplicio Jr. et al. [14] provided a comparison of message authentication code algorithms for the TelosB platform using TinyOS. They implemented the algorithms themselves and did not specify what TinyOS version was used for. The compilations were performed with GNU mspgcc version 3.2.3 and they preferred to use *-Os* flag to get a memory optimized result.

Regarding existing full-fledge security frameworks for WSN, the TinySec [15] and MiniSec [16] were designed only with TinyOS [17] in mind and each of them was implemented for a specific sensor device. Another popular one, ContikiSec, was planned and implemented particularly for the ContikiOS. Such differences make it impractical to compare cryptographic algorithms. Moreover, the structures within each framework can also adopt different project and implementation approaches, using different policies of processing or memory usage. Modifications in operating system components (in the case of MiniSec) are an even more complex issue.

Another difficulty is that in recent years the devices and operating systems have been updated a number of times, while the security structures have not. In this way, the structures have become unusable for a number of platforms, such as the cases of TinySec and MiniSec, which are no longer usable for series 2 of the versions of TinyOS (TinyOS2x).

In addition, the drivers are sometimes unavailable for the operating system. This was the case in the IEEE 802.15.4 security specification, which was available in certain devices, like the TelosB [4] and the MICAz [6], without certain functions required to activate the security resources in the operating systems, which made it very difficult to utilize the security structure.

All those frameworks, as explained, are highly bounded to specific platforms or operating systems, and therefore we do not evaluate them in this work.

Table 1: Architecture comparison.

Class	Device		Energy (mW)	
	Name	Architecture	Idle	Processing
SBC	Intel Edison	Atom (x86@500 MHz dual-core)	88	340
	Intel Galileo	Quark (x86@400 MHz)	520	550
	BeagleBone Black	Cortex-A8 (ARM@1 GHz)	310	400
	Raspberry Pi A	Broadcom (ARM@700 MHz)	130	180
	Raspberry Pi B	Broadcom (ARM@700 MHz)	380	410
WSN	MICAz	ATmega128L (RISC@7.3 MHz)	26	0.025
	TelosB	TI MSP430 (RISC@8 MHz)	4.8	0.035
	Arduino Yun	ATmega32U4 (RISC@16 MHz) + Atheros (MIPS@400 MHz)	240	280

3. Scenario

Given related work discussed, the scenarios we consider for our performance evaluation must be platforms or operating systems independent. Next we present the platforms and operating systems, as well as the cryptographic algorithms we selected.

3.1. Platforms. This section presents an overview of the selected embedded devices, which are typically used in IoT and WSN applications.

Table 1 depicts a group of embedded architectures. We selected some popular embedded devices found on literature according to main usage: SBC (Single Board Computer) and WSN (Wireless Sensor Network). SBC class includes devices with general purpose computing power, for gateway or sink, and WSN includes only platforms targeting low-power communication, typically used as nodes. Data were compiled from [18–21] and updated with [22].

Considering the goal of this work, we choose Intel Edison and TelosB as representative of their respective classes. We expect that the performance behavior remain the same for a device of the same class, except for Arduino Yun, whose consumption is far from WSN nodes (built for low energy) but does not achieve enough performance to be compared with SBC class. Platforms that do not include any type of IoT communication without external support (e.g., Arduino UNO) were excluded from our evaluation.

3.1.1. TelosB. MEMSICs TelosB Mote is an open-source platform based on a low-power MSP430 16-bit 8 MHz microcontroller with 10 KB RAM and 48 KB program flash memory. TelosB has a low current consumption and is powered by two AA batteries, but it can be plugged in the USB so power is provided from the host computer [4]. TelosB has an IEEE 802.15.4 compliant RF transceiver and runs TinyOS 1.1.11 or higher and Contiki OS.

3.1.2. Intel® Edison. The Intel Edison is a low-power 32-bit x86 IoT platform, which contains a core system processing and connectivity elements: a dual-core, dual-threaded Intel Atom CPU at 500 MHz, and a 32-bit Intel Quark microcontroller at 100 MHz processor; 1 GB RAM; 4 GB eMMC internal storage; and IEEE 802.11 and Bluetooth 4.0. But it is not a self-contained, standalone device. It relies on the end-user support of input power (i.e., the computing module does not control battery recharging) [11]. The Linux distribution Yocto is its default operating system (OS), but its x86 architecture can enable a variety of OSs.

3.2. Operating Systems. This section presents a brief overview of the main operating systems involved in the performance analysis, that is, the TinyOS, ContikiOS, and Yocto.

3.2.1. TinyOS. TinyOS is an open-source operating system designed for low-power wireless devices, such as sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters [23]. According to its authors [24], TinyOS is a tiny (less than 400 bytes), flexible operating system built from a set of reusable components that are assembled into an application-specific system and supports an event-driven concurrency model based on split-phase interfaces, asynchronous events, and deferred computation called tasks. TinyOS is implemented in the NesC language, which supports the TinyOS component and concurrency model as well as extensive cross-component optimizations and compile-time race detection. It is a programming framework and set of components for embedded systems that enable building an application-specific OS into each application [24].

TinyOS became the de facto operating system for WSN until 2012, because it combines an efficient memory footprint with an easy-to-use interface for small WSN devices like MICAz and TelosB.

3.2.2. Contiki. Contiki [25] is an open-source operating system for the Internet of Things that connects tiny low-cost, low-power microcontrollers to the Internet.

Contiki applications are written in standard C mainly using structures called *protothreads*. Applications for Contiki can be simulated using Cooja simulator to evaluate networks behavior before deployment into the hardware. ContikiOS 3.0 supports the recently standardized IETF protocols for low-power IPv6 networking, including the 6lowpan adaptation layer [26], the RPL IPv6 multihop routing protocol [27], and the CoAP RESTful application-layer protocol [25, 28, 29].

Contiki runs on a wide range of tiny platforms, ranging from 8051-powered systems on a chip through the MSP430 and the AVR to a variety of ARM devices.

3.2.3. Yocto. The Yocto Project is an open-source collaboration project that provides templates, tools, and methods to help creating custom Linux-based systems for embedded products regardless of the hardware architecture [30].

Its capabilities include a Yocto Project kernel which can cover many profiles across multiple architectures including ARM, PPC, MIPS, x86, and x86-64. It is the default OS for the Intel Edison and it is adopted in our benchmarks.

3.3. Cryptographic Algorithms. The algorithms described in this section were chosen among many other cryptographic algorithms in the literature. This choice has been made about electing algorithms that were designed to be lightweight on resource constrained platforms, where memory and processing resources are relatively scarce, but at same time providing the minimum desired security level.

3.3.1. Symmetric Ciphers. AES [31] algorithm is a symmetric block cipher capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data using blocks of 128 bits.

Curupira is a special-purpose block cipher tailored for platforms where power consumption and processing time are very constrained resources, such as sensor and mobile networks, or systems heavily dependent on tokens or smart cards. Curupira is an instance of the Wide Trail family of algorithms, which includes the AES cipher, and displays involutional structure, in the sense that the encryption and decryption modes differ only in the key schedule. Its first version, named Curupira-1 [32, 33], also presents a cyclic key schedule, meaning that the original key is recovered after a certain number of rounds, whereby allowing them to be computed in-place in any order.

Curupira-2 [34] adopts the same round structure as Curupira-1 but takes a less conservative (with a slower diffusion) key scheduling algorithm. This results in a higher performance when the round keys are computed on demand, a common situation in resource constrained networks.

Trivium is a stream cipher that takes a stream of plaintext, a secret key, and an IV as input and then operates on the plaintext with key stream generated by the key and IV, typically bit by bit. It is designed for constrained devices to generate up to 2^{64} bits of key stream from an 80-bit secret key and an 80-bit initial value (IV) [35].

Grain is another stream cipher submitted to eSTREAM [36]. This algorithm has been selected for the final eSTREAM portfolio by the eSTREAM project and is designed primarily for restricted hardware.

3.3.2. Cryptographic Hashing. Blake2 is an extremely fast hash function, yet there are no known security issues in this algorithm. Blake2 has different versions that are suitable for different situations. We chose Blake2s, because it is suitable for resource constrained platforms [37].

Keccak is the winner of SHA3 competition [38]. It consists of a family of sponge functions that hashes message texts with a high level of security. The Keccak Code Package [39] provides the implementation of the Keccak sponge function that we used. For our benchmark purposes we used the version with rate of 1088, capacity of 512, and hash output length of 256 bits.

3.3.3. Message Authentication Codes (MACs). Keyed-Hash Message Authentication Code (HMAC) is standardized in FIPS 198-1, a mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function, in combination with a shared secret key. Additional applications of keyed-hash functions include their use in challenge-response identification protocols for computing responses, which are a function of both a secret key and a challenge message. An HMAC function is used by the message sender to produce a value (the MAC) that is formed by condensing the secret key and the message input. The MAC is typically sent to the message receiver along with the message. The receiver computes the MAC on the received message using the same key and HMAC function as used by the sender and compares the result computed with the received MAC. If the two values match, the message has been correctly received, and the receiver is ensured that the sender is a member of the community of users that share the key [40].

Marvin message authentication code [41] was proposed by Simplicio et al. and was exactly designed for resource constrained devices. Marvin explores the structure of an underlying block cipher to provide security at a small cost in terms of memory needs. Also, Marvin can be used as an authentication-only function or in an authenticated encryption with associated data (AEAD) scheme. AES, Curupira-1, or Curupira-2 can be adopted as the underlying block cipher.

3.3.4. Authenticated Encryption with Associated Data (AEAD). Counter mode (CTR) is a mode of operation that turns a block cipher into a stream cipher so it used for achieving confidentiality [42]. It generates the next keystream block by encrypting successive values of a "counter." The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment by one counter is the simplest and most popular. CTR mode is well suited to operate on a multiprocessor machine where blocks can be encrypted in parallel. If the IV/nonce is random, then they can be combined together with the counter using any lossless operation (concatenation, addition, or XOR) to produce the actual unique counter block for encryption. In case of a nonrandom nonce (such as a packet counter), the nonce and counter should be concatenated (e.g., storing nonce in upper 64 bits and the counter in lower 64 bits). Notice that simply adding or XORing of the nonce and counter into a single value would completely break the security under a chosen-plaintext attack.

Counter with Cipher Block Chaining-Message Authentication Code (CCM) is a two-pass patent-free AEAD. CCM is based on an approved symmetric key block cipher algorithm with 128-bit block size, such as the Advanced Encryption Standard (AES) algorithm currently specified in FIPS 197. CCM can be considered a mode of operation of the block cipher algorithm. As with other modes of operation, a single key to the block cipher must be established beforehand

among the parties to the data; thus, CCM should be implemented within a well-designed key management structure. The security properties of CCM depend, at a minimum, on the secrecy of the key [43].

Offset Codebook Mode (OCB) is a one-pass authenticated encryption scheme designed by Rogaway et al. [44]. What makes OCB remarkable is that it achieves authenticated encryption in almost the same time as the fastest conventional mode, CTR mode, achieves privacy alone. This results in lower computational cost compared to using separate encryption and authentication functions. OCB performance overhead is minimal compared to classical, nonauthenticating modes like CBC. OCB does not require the nonce to be random; a counter, say, will work fine. Unlike some modes, the plaintext provided to OCB can be of any length, as well as the associated data, and OCB will encrypt the plaintext without padding it to some convenient-length string, an approach that would yield a longer ciphertext. If one is encrypting and MACing in a conventional way, like CTR-mode encryption and the CBC MAC, the cost for privacy and authenticity is going to be twice the cost for privacy alone, just counting block cipher calls. Unfortunately, OCB is patented for commercial use in the USA [45]. There are now free licenses available for OCB with unusual restrictions, which some implementors have expressed concerns over (e.g., one license only applies only to open-source software; another allows only use of OCB in OpenSSL implementation).

EAX is a two-pass AEAD; that is, encryption and authentication are done in separate operations. This makes it much slower than OCB, though unlike CCM it is "online." Still, EAX has three advantages: first, it is patent-free. Second, it is pretty easy to implement. Third, it uses only the encipher direction of the block cipher, meaning that one could technically fit it into an implementation with a very constrained code size, if that sort of thing is a concern. One possible drawback is that EAX is not entirely parallelizable due to the use of CMAC [46].

LetterSoup is a parallelizable two-pass AEAD scheme based on the Marvin for message authentication [41]. One of the main interests of using Marvin is that it follows the ALRED construction [47], meaning that each block of the message blocks is processed using a few unkeyed rounds of an underlying block cipher (the so-called Square Complete Transform, SCT) instead of a full encryption as in CMAC. LetterSoup was designed with constrained platforms in mind, one advantage being that its encryption function (the LFSRC mode) is involutional; that is, applying it twice recovers the plaintext, so the code size gets reduced since only one direction is needed. Its official implementation employs Curupira-2 as the underlying block cipher [48]. Another advantage is that the security of LetterSoup is formally analyzed giving more confidence for the algorithm. In addition, the message and the header can be processed in any desired order, and the tag can be verified before the decryption process takes place. The IVs used must be nonrepeating [41]. In the figures and tables we will refer to LetterSoup as LTS. Moreover, in our analysis the LTS implementation by Simplicio [49] utilizes Curupira-2 [50] as its underlying block cipher.

Ketje is composed of two authenticated encryption schemes with support for associated data and was designed by the inventors of Keccak, the winner of the SHA3 hash function. Ketje is a nonce-based AEAD, meaning that the encryption function is deterministic and needs a different nonce for each invocation to be secure. It supports two security levels, that is, 96 bits and 128 bits. Ketje builds on round-reduced versions of the Keccak-f permutation as primitives, that is, the Keccak-f[400] and Keccak-f[200], allowing for code reuse when Keccak is already being used. It also offers considerable side-channel protection. On the other hand, it is more memory consuming compared to the above-mentioned AEAD schemes [51]. The implementation employed Ketje v1 in this work along with Keccak-f[400] for 128-bit security. The code was obtained from the Keccak Code Package [39].

4. Experimental Setup and Methodology

As discussed in the previous section, the evaluation covers algorithms from four categories. Next we address the implementation and configuration used for each of them.

4.1. Symmetric Ciphers. The reference implementations of the underlying block ciphers Curupira and Trivium are designed for 8-bit platforms. For the case of AES operations, we used the highly optimized implementation for 16-bit CPUs with 128-bits keys (and 128-bit IVs) available at [10]. For the Trivium synchronous stream cipher [35], we used the implementation published in [52], configured with key size of 112 bits and IV size of 96 bits. For the Curupira [34] block cipher we have adopted the reference implementation [50]. This implementation is optimized for key size of 96 bits.

4.2. Cryptographic Hashing. We selected the official Blake2s implementation [53]. For the Keccak hash function we adopted the Keccak Code Package [39] which provides the implementation of the Keccak sponge function.

4.3. Message Authentication Codes (MAC). MAC algorithms can be built based on hashing or symmetric cipher algorithms. We chose one algorithm from each category.

HMAC uses a cryptographic hash function to define a MAC. Any hash function could be used by this algorithm. We used the implementation published in [54], considering the following parameters: SHA-256 hash function, 128-bit key size, and 64-byte block size.

Marvin [41] uses a symmetric cipher algorithm to ensure authenticity and integrity of a received message. It can use AES or Curupira-2 as the base cipher. We used the implementation published in [55], configured with the following parameters: Curupira-2 cipher, 96-bit key size, and 12-byte tag size.

4.4. Authenticated Encryption with Associated Data (AEAD). For EAX [56] we used the implementation published in [49], and the Curupira-2 was adopted as the underlying block cipher. For the LetterSoup [41], the default parametrization uses Curupira-2 as block cipher, which is also used in the implementation published in [48]. LetterSoup will be referred to as LTS in the next graphs and tables. OCB [57] is a patented

Table 2: Operations of interest for each primitive.

Symmetric ciphers	Hash	MAC	AEADs
Init	*Init*	*Init*	*Init*
Encryption	*Update*	*Update*	*Encryption*
Decryption	*Final*	*Final*	*Decryption*

mode of operation. Our implementation for this algorithm [49] also uses Curupira-2 as cipher algorithm. Ketje is an AEAD algorithm based on Keccak. It is designed for resource constrained platforms, and the implementation used in this report is the one at [39].

It is worth pointing out that a usual good approach for comparing performance of different algorithms is that the same implementor implements all of them since the same amount of tricks is more evenly applied. On the other hand, we think a single implementor producing tens of new nontested implementations for the different platforms is not reasonable. Actually, our goal in here is to evaluate existing already deployed and better tested implementations and their behavior in the selected platforms.

In order to evaluate algorithms for each category *(symmetric cipher*, *hash*, *MAC*, *AEAD)*, we calculate the run time and energy consumption of each relevant operation. The procedures considered are listed in Table 2.

The run times and energy consumption are extracted using a python script wrote by the authors, which reads the output file from the LabView setup explained next. The measurement setup for current consumption consists of an Agilent 34401A digital multimeter [58] that reads the drained current and communicates with a computer via GPIB, running LabView. The current sampling is limited to 500 Hz. The mote is powered by a fixed voltage source at 3 V. Since it is ineffective to compare cryptographic algorithms using a fixed message size, we compared all algorithms varying the sizes. Messages up to roughly 100 bytes were considered, according to the following rules:

(i) Symmetric block ciphers: Curupira-2 and AES data sizes D will be multiples of the block size

(ii) Hash: multiples of the tag size

(iii) MAC: multiples of the underlying cipher (or hash) block size

(iv) AEAD: multiples of the underlying cipher (or hash) block size

Five independent measurements are conducted in order to calculate the average, standard deviation, and confidence interval of 95%. Each operation runs N times, and N is determined by the time t of a single run of the operation. It should satisfy $N \leq (100/t)$. This is due to a limitation of a LabView internal buffer size which becomes fully loaded after 100 seconds of measurements using the maximum sampling rate of our setup.

For the *TelosB platform* the number N is defined according to the algorithm as follows:

(i) AES, Curupira-2, LetterSoup, OCB, Blake2s, Marvin, HMAC, and Trivium: $N = 100$.

(ii) EAX and Keccak: $N = 80$.

(iii) Ketje: $N = 5$.

For the Intel Edison with Yocto OS the following N is adopted for each algorithm:

(i) For execution time experiments:

(a) Symmetric ciphers: $N = 200000$.
(b) Hash: $N = 200000$.
(c) MAC: $N = 100000$.
(d) AEAD: $N = 30000$.

(ii) For energy consumption experiments:

(a) Symmetric ciphers: $N = 250000$.
(b) Hash: $N = 80000$.
(c) MAC: $N = 500000$.
(d) AEAD: $N = 40000$.

Before any test is performed, a script that stops unnecessary OS processes is executed on Yocto OS. Moreover, the run time acquisition in this case is conducted by means of the *gettimeofday* function which provides a microsecond resolution [59]. Because symmetric algorithms are very fast, one might think that *gettimeofday* may not be enough to get a precise time taking, but since we are taking a large amount of executions of the same operation we are able to get precise result, and yet the precision of microseconds is fair enough for our experiments since the fastest operation we have measured is more than 1 microsecond.

All procedures were executed N times in order to allow a statistical analysis of the experiment and defined an error margin, which was calculated with a standard error.

The compiler used for TelosB was the GNU msp430-gcc LTS 20120406 and the *-O3* optimization option was set, which means optimizing for speed, since applications usually want to minimize energy consumption for the target IoT and WSN scenarios. We used GCC 5.1 for the Intel Edison.

5. Benchmarks and Results on the TelosB Platform, TinyOS

We first collect the block cipher performance results over TelosB for the *encryption operation* in Figure 1.

It is clear from the graphs in Figure 1 that run time and energy consumption are proportional in the TelosB platform. This can be explained by the fact that TinyOS is event-oriented and very lightweight operating system. When a task is to run, the operating system simply yields all the processing resources to the task. The consequence is that practically all energy consumption is due to the running algorithm in the task and energy consumption gets proportional to the running time of the algorithm.

When we compare the results in Figure 1 with the ones reported by Casado and Tsigas [8] for the AES on 16-bit MSP microcontrollers, we observe that the AES implementation SUPERCOOP combined with most recent TinyOS 2.1.2 over an MSP430 microcontroller performs 2x faster for both

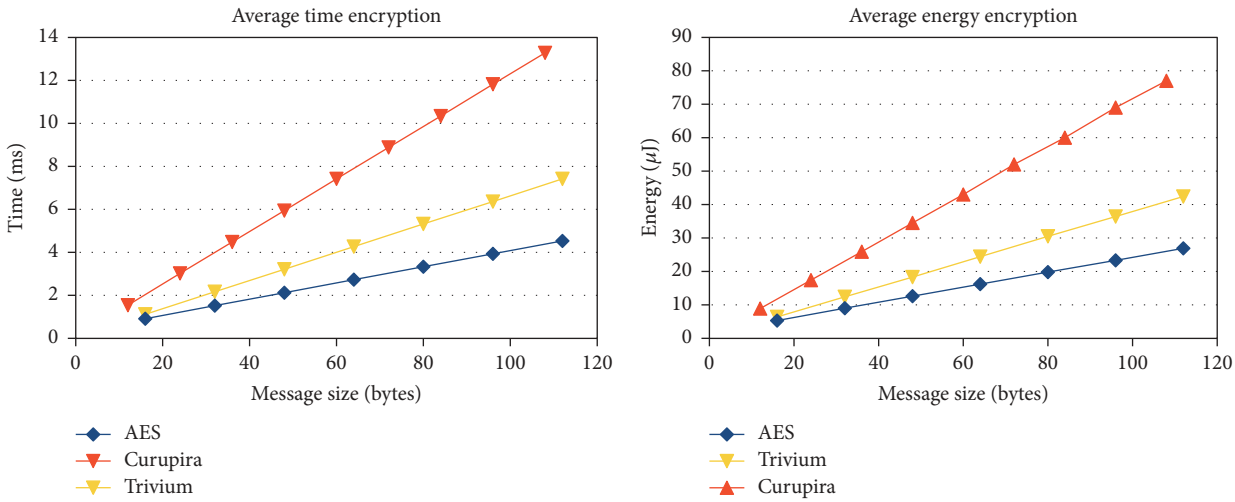

Figure 1: TinyOS/TelosB: average time and energy for the *encryption* operation for each block cipher.

encryption/decryption operations (1 ms per block versus >2 ms per block) and consequently spends 2x less energy for these computations. This observation gives evidence that many of the previous reports in the literature do not reflect precisely the most up-to-date energy footprints of combined best implementations and OSs for WSN.

For all the next operations with block ciphers, hash, and AEAD, energy consumption is proportional to running times; thus we provide only the time comparison for some of the next benchmarks since we can expect the same behavior for the energy comparison.

It is also worth pointing out that from Figure 1 Curupira-2 performed slower when compared to other ciphers although having smaller block size and being tailored for constrained devices. We asked the original authors about this apparent paradox and they clarified that although Curupira-2 implementation was developed with small word sizes in mind they aimed at correctness over high performance. On the other hand, a much better performance is expected if properly optimized. Moreover, from our results, the relative decryption behavior for the three algorithms is the same; thus we do not provide the detailed figures herein for conciseness.

In practice, block ciphers are used along with modes of operation. We provide the benchmarks of this combination for the AES block cipher next.

5.1. AES-CTR and AES-CBC Encryption Modes. The AES in CTR mode follows its standard specifications. For the CBC (Cipher Block Chaining) mode, the following parameters (experiments follow the same described setup) are adopted:

(i) Key size is 128 bits.

(ii) Block size is 16 bytes.

(iii) Plaintext size is 16 bytes.

(iv) Number of rounds is 10.

(v) Precomputed tables are used in AES MixColumns for run time optimization.

The run time and energy consumption for different operations from CTR and CBC modes are shown in Table 3.

Table 3: Time in ms for one execution in different mode AES of *init*, *encryption*, and *decryption* tasks, with message size = 16 bytes.

Mode	Init	Encryption	Decryption
CBC	1,28 ± 0,01	2,69 ± 0,02	2,82 ± 0,04
CTR	0,47 ± 0,01	0,91 ± 0,01	0,91 ± 0,01

Table 4: Time to execute one *init* operation for each hash algorithm. Time is in ms.

Blake2s	Keccak
0,96 ± 0,00	6,96 ± 0,02

From the results in Table 3, we notice that CTR mode is about 3x faster and less energy consuming for the encryption/decryption operations when compared to CBC.

We now provide the benchmark results for the *hash* functions.

The run time and energy consumption of the *init* operation for the different hash algorithms are relatively cheap compared to other hash operations and are shown in Table 4. But it is worth mentioning that Blake2's initialization is 7x faster than Keccak's. This operation has a roughly constant execution time for all data sizes.

Tables 5 and 6 show the comparison and the numerical differences between hash algorithms for the *update* operation.

One can check that Blake2's performance surpasses Keccak's. A better illustration of the *update* operation is provided in Figure 2.

From the results of Tables 4, 5, and 6 and Figure 2 we clearly see an advantage of Blake2s compared to Keccak. The main reason for this behavior is that Blake2s was particularly designed to run faster in smaller word processors (TelosB is embedded with a MSP430 microcontroller with 16-bit words).

Finally, Table 7 shows the execution time for the *final* operation, which is roughly constant for any data size ($12 \leq D \leq 108$), thus the time for only one input size is shown. Keccak is faster than Blake2 in this operation.

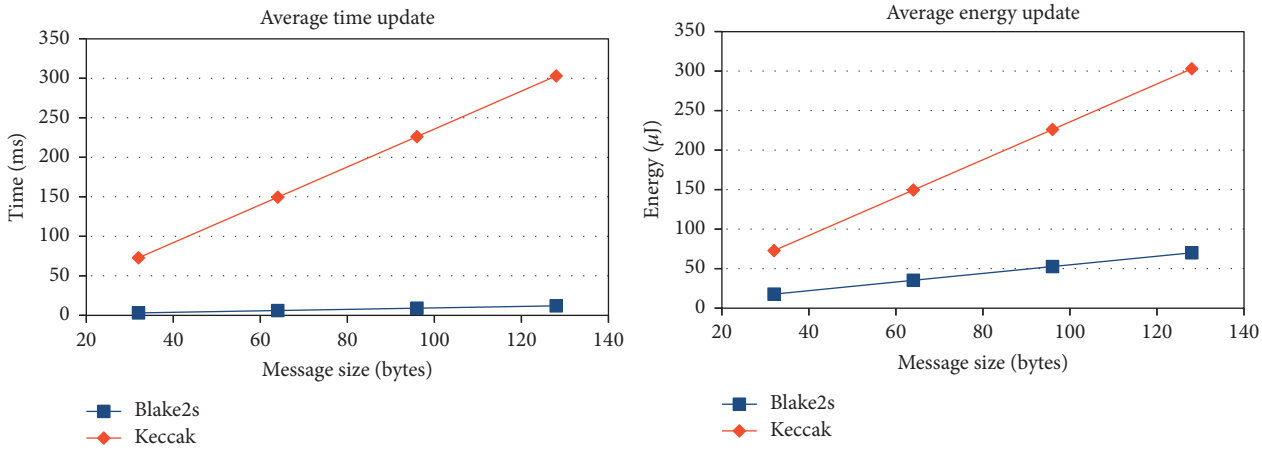

Figure 2: TinyOS/TelosB: average time and energy consumption of *update* operation for each hash algorithm.

Table 5: Time to execute one *update* operation for each hash algorithm. Time is in ms and the data size D is in bytes.

D	Blake2s	Keccak
32	3,03 ± 0,01	72,8 ± 0,2
64	6,00 ± 0,02	149,5 ± 0,4
96	8,98 ± 0,02	226,1 ± 0,6
128	11,94 ± 0,04	303,0 ± 1,0

Table 6: Energy to execute one *update* operation for each hash algorithm. Energy is in μJ and the data size D is in bytes.

D	Blake2s	Keccak
32	17,7 ± 0,2	410 ± 20
64	35,2 ± 0,3	840 ± 50
96	52,6 ± 0,5	1270 ± 70
128	70,0 ± 0,6	1700 ± 100

Table 7: Time to execute one *final* operation for each hash algorithm. Time is in ms and D is the data size in bytes.

D	Blake2s	Keccak
128	6,27 ± 0,02	4,1 ± 0,02

Notice that here we compare two *MAC* algorithms, Marvin and HMAC, which operate with different sizes of blocks. Marvin does not have exact blocks of sizes 64 bytes and 128 bytes, but it offers a range of sizes varying from 12 bytes to any multiple of it. On the other hand, HMAC works with blocks multiple of 64 bytes. For the case of 64-byte block in HMAC and 72-byte block in Marvin, Marvin is still faster and more energy efficient even when operating with large blocks.

Table 8 shows the execution time of *init* operation for each MAC algorithm. The execution time is roughly constant for every data size, and Marvin takes longer to initialize than HMAC.

Table 9 shows the execution time and energy consumption for the *update* operation for each MAC algorithm.

Table 8: Time to execute one *init* operation for each MAC algorithm. Time is expressed in ms.

Marvin	HMAC
5,62 ± 0,01	28,95 ± 0,04

Table 9: Time to execute one update operation for each MAC algorithm. Time is in ms and D is the data size in bytes.

D	Marvin	HMAC
12	1,68 ± 0,01	
36	4,88 ± 0,01	
60	8,08 ± 0,02	
84	11,28 ± 0,02	
108	14,50 ± 0,03	
128		28,37 ± 0,04

Table 10: Time to execute one *final* operation for each MAC algorithm. Time is expressed in ms and D is the data size in bytes.

D	Marvin	HMAC
108	7,04 ± 0,01	
128		35,43 ± 0,05

Table 11: Time to execute one *init* operation. Time is expressed in ms.

EAX	OCB	LetterSoup	Ketje
7,67 ± 0,01	4,88 ± 0,01	3,04 ± 0,01	18,4 ± 0,2

Benchmarks for the *update* operation are also shown in Figure 3.

Finally, Table 10 shows the time and energy performance for the *final* operation. The execution time for Marvin is roughly constant for all data sizes and thus we only plot the result for the 108-byte input size. Marvin performs about 5x faster than HMAC for the *final* operation.

Table 11 shows the execution time and energy consumption of the *init* operation for different *AEAD* algorithms.

Table 12: Time to execute one *encryption* operation for each AEAD algorithm (associated data size = 0). Time is expressed in ms and D is the data size in bytes.

D	EAX	OCB	LetterSoup	Ketje
12	7,85 ± 0,02	6,58 ± 0,01	4,94 ± 0,01	93,2 ± 0,2
36	15,43 ± 0,04	11,10 ± 0,01	8,66 ± 0,01	167,6 ± 0,2
60	23,00 ± 0,04	15,60 ± 0,02	12,38 ± 0,03	242,4 ± 0,2
84	30,80 ± 0,04	20,12 ± 0,03	16,08 ± 0,05	316,6 ± 0,3
108	38,18 ± 0,06	24,65 ± 0,05	19,78 ± 0,06	391,2 ± 0,3

Table 13: Time to execute one *encryption* operation for each AEAD algorithm (associated data size = message size). Time is expressed in ms and D is the data size in bytes.

D	EAX	OCB	LetterSoup	Ketje
12	9,86 ± 0,03	8,76 ± 0,02	4,96 ± 0,01	118,4 ± 0,2
36	21,33 ± 0,04	17,50 ± 0,03	10,30 ± 0,02	267,4 ± 0,3
60	32,8 ± 0,1	26,24 ± 0,05	14,86 ± 0,04	416,0 ± 0,6
84	44,2 ± 0,2	34,98 ± 0,06	19,40 ± 0,06	566,0 ± 3,0
108	55,7 ± 0,1	43,68 ± 0,08	23,96 ± 0,07	714,8 ± 0,7

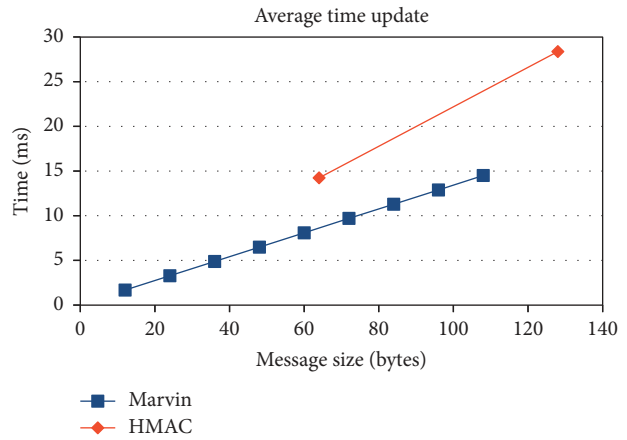

Figure 3: TinyOS/TelosB: average time consumption of *update* operation for each MAC algorithm.

LetterSoup initialization is the fastest AEAD with a small difference compared to OCB. Ketje has the slowest initialization which comes from the fact of managing with larger states. On the other hand, LetterSoup's better performance is influenced by the smaller state but at a penalty of a slightly smaller security level of 96 bits.

Table 12 shows the run time and energy consumption of the *encryption* operation without associated data. Note that Ketje's performance is significantly worse. Although its implementation is intended for 16-bit CPUs, the design works with 50-byte internal states which are at least 3x larger than states from other algorithms, which explains the slower behavior.

Table 13 shows the execution time of the *authenticated encryption with associated data* operation. Ketje remains the slowest algorithm in this experiment by one order of magnitude compared to others. Recall that a decision to select one of these algorithms depends on the tradeoff performance and the additional features of each algorithm. Recall that LetterSoup is faster but comes with a slightly smaller security level and also is not standardized. EAX and OCB are standardized, and OCB has some patent issues. Ketje in turn could be reusing Keccak's code and save code memory in case a hash function is also desired in the same application.

Figure 4 shows the execution time and energy of the *encryption with associated data* operation. LetterSoup presents a better performance than others when there is associated data. Therefore it presents a better option when the combined services are desired.

The operation *decryption with associated data* presents similar performance as the encryption counterpart and is thus omitted here.

According to the results described in this section, the best performances in terms of speed and energy on TelosB with TinyOS are achieved by the following:

(i) Encryption mode: *AES-CTR*

(ii) Hash: *Blake2*

(iii) MAC: *Marvin*

(iv) AEAD: *LetterSoup*

It is worth pointing out that, for each type above, speed might not be the priority and the decision should be made taking into account all the desired features for each algorithm to have.

6. Benchmarks and Results on the TelosB Platform, ContikiOS

We first perform benchmarks for each symmetric algorithm running it along with ContikiOS 3.0 over the TelosB mote. The only exception is the Blake2s hash function. Its reference implementation consumes a large amount of flash (code) memory compared to the other analyzed algorithms and cannot be compiled along with ContikiOS 3.0 which is high code memory consuming as well. In this case, the Blake2s

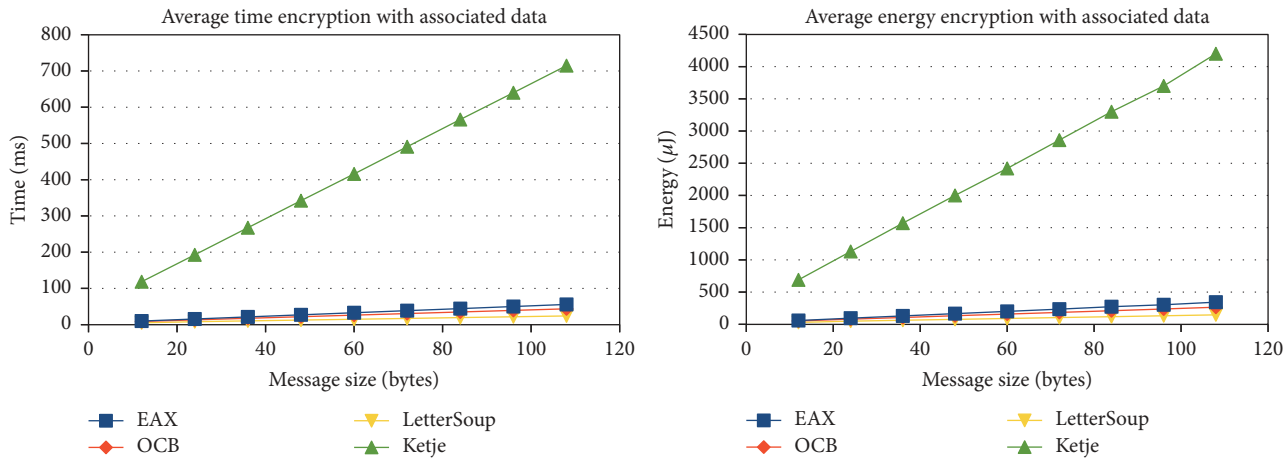

Figure 4: TinyOS/TelosB: average time and energy to execute one *encryption* operation for each AEAD algorithm with associated data.

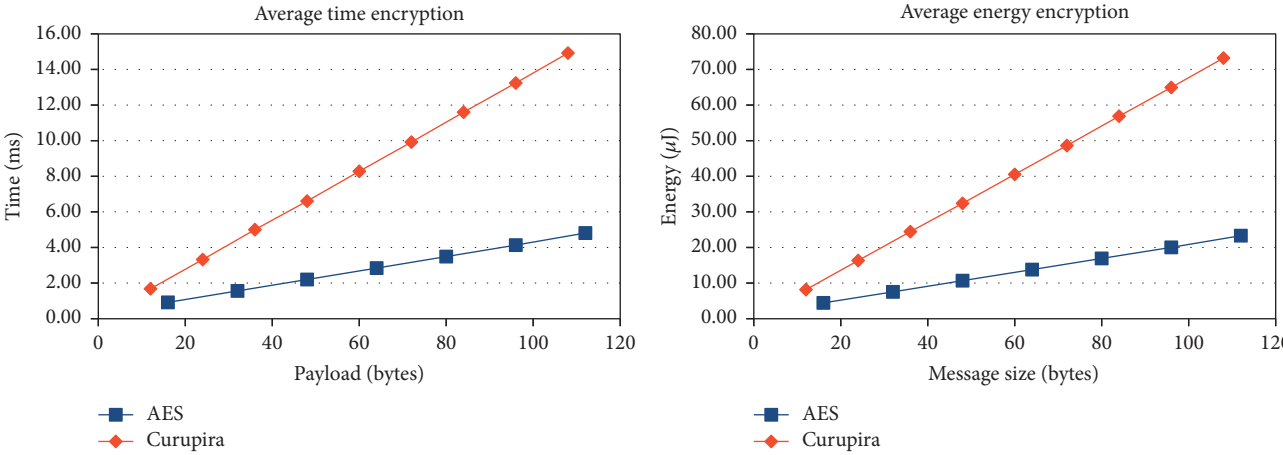

Figure 5: ContikiOS/TelosB: average time and energy to execute one *encryption* operation for each block cipher.

benchmarks are performed along with ContikiOS 2.7 which is less flash memory consuming. It is worth mentioning that ContikiOS 2.7 introduces some drawbacks like more current spikes which potentially leads to more energy consumption. Fortunately we were able to filter that overhead by estimating the average current of operation and considering only the area below this average current in the current versus time graph, which gives the average charge.

In the present experiments, the same methodology used for TinyOS is adopted. Same implementations are also used and same ranges of parameters are defined.

Figure 5 compares the run time of the *encryption* operation for different block ciphers. We can see a similar outcome when compared to the behavior of the ciphers on TelosB. AES is still faster than Curupira-2. It is important to point out that ContikiOS presents run times and energy consumption very close to the ones observed for TinyOS in Figure 1.

Figure 6 presents the *decryption* operation for different block ciphers.

The benchmarks for the *update* operation of MAC algorithms are shown in Figure 7 while the benchmarks for the *update* operation of *hash algorithms* are shown in Figure 8.

The *authenticated encryption with associated data* operation is shown in Figure 9.

The *decryption with associated data* displays similar behavior and thus is omitted here.

6.1. Discussion. In comparison, the performance results on TinyOS and ContikiOS are very close to one another, meaning that the overhead of those operating systems on processing-only operations is similar. This corroborates the conclusions by Margi et al. [60]. On the other hand, we also showed that even though the relative performance id comparable, the magnitude of the results of time and energy is very different from other works such as the comparison with the results in [8] already discussed in Section 5.

7. Benchmarks and Results on Intel Edison with Yocto

7.1. Ciphers. In this section we present the performance and energy results on Intel Edison platform based on the Yocto OS. Tables 14 and 15 show the execution time and energy consumption for one *init* operation. This operation has a

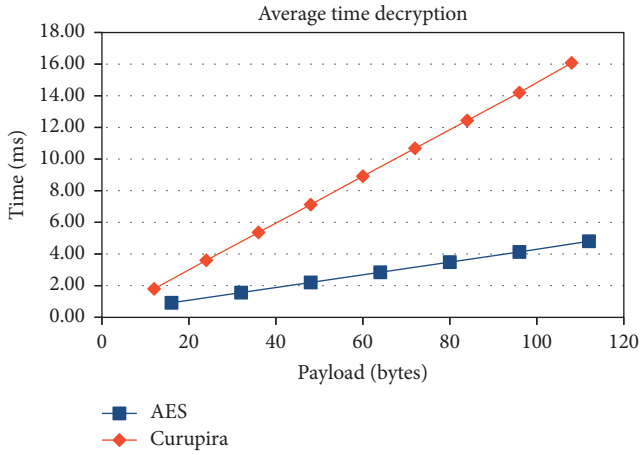

Figure 6: ContikiOS/TelosB: average time to execute one *decryption* operation for each block cipher.

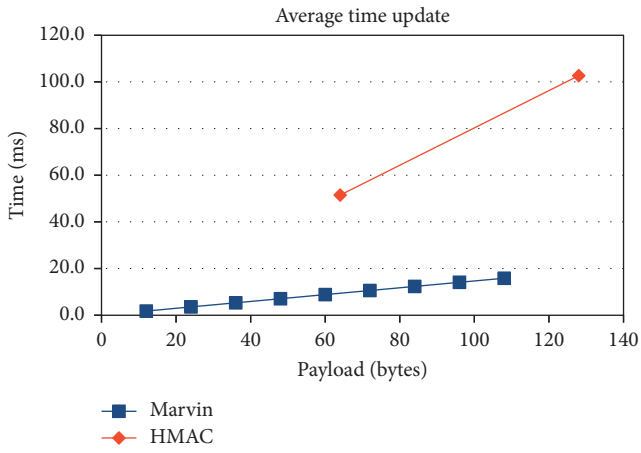

Figure 7: ContikiOS/TelosB: average time to execute one *update* for each MAC algorithm.

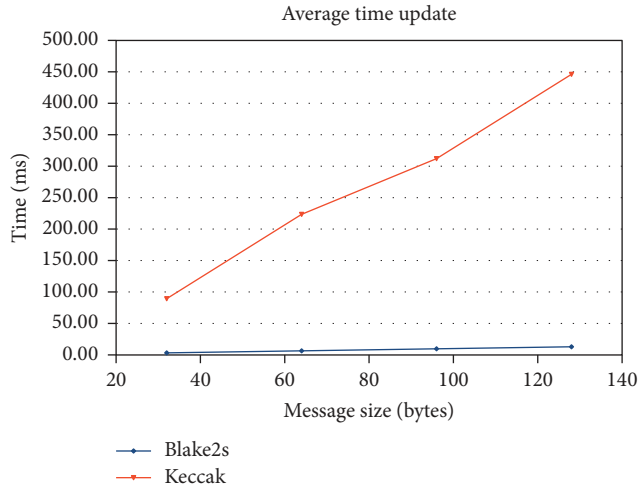

Figure 8: ContikiOS/TelosB: average time to execute one *update* for each hash algorithm.

Table 14: Time to execute one *init* operation for each block cipher. Time is in μs and D is the data size in bytes.

D	AES	Curupira	Trivium	Grain
12		13.90 ± 0.01		
16	7.33 ± 0.01		11.32 ± 0.01	8.58 ± 0.01

Table 15: Energy consumption to execute N *init* operations for each block cipher. The consumption is expressed in J.

AES	Curupira	Trivium	Grain
0.49 ± 0.00	1.12 ± 0.00	0.89 ± 0.00	0.62 ± 0.00

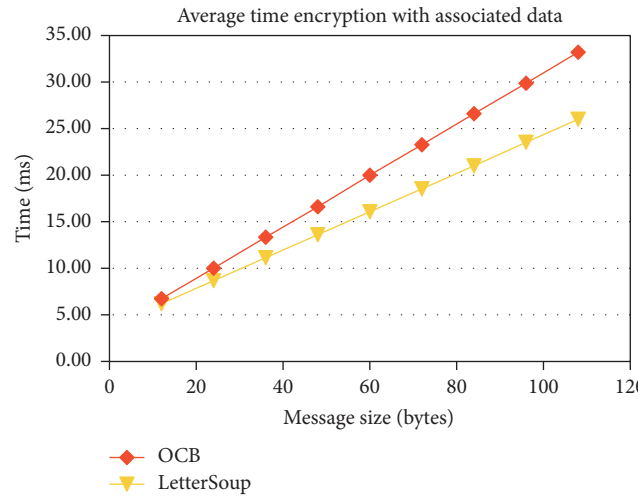

Figure 9: ContikiOS/TelosB: average time to execute one *authenticated encryption* with associated data operation for each AEAD algorithm.

(roughly) constant execution time that is not affected by the data size. Even though the speed of this operation cannot solely define the best algorithm, we can see that the fastest *init* operation is performed by AES, while Curupira is the slowest.

Figure 10 fully specifies the performance of the *encryption* operation for each block cipher algorithm. From the figures one can note that AES is the fastest algorithm for data size up to 32 bytes. For larger data sizes, AES is surpassed by Trivium and for even the larger messages sizes (more than ≈95 bytes) also by Grain.

The AES performance is expected to be more sensitive to message sizes than Trivium and Grain, since AES is a block cipher (this is also true for Curupira). Curupira in turn presented the worst performance for all data sizes, but this is due the nonfully optimized implementation.

The performance behavior observed for decryption is identical to encryption as expected and therefore the actual data is omitted.

Tables 16 and 17 show the execution time and energy consumption by the *init* operation for *hash algorithms*. Analyzing only this operation is not very meaningful for defining the best algorithm, but Blake2's initialization is much

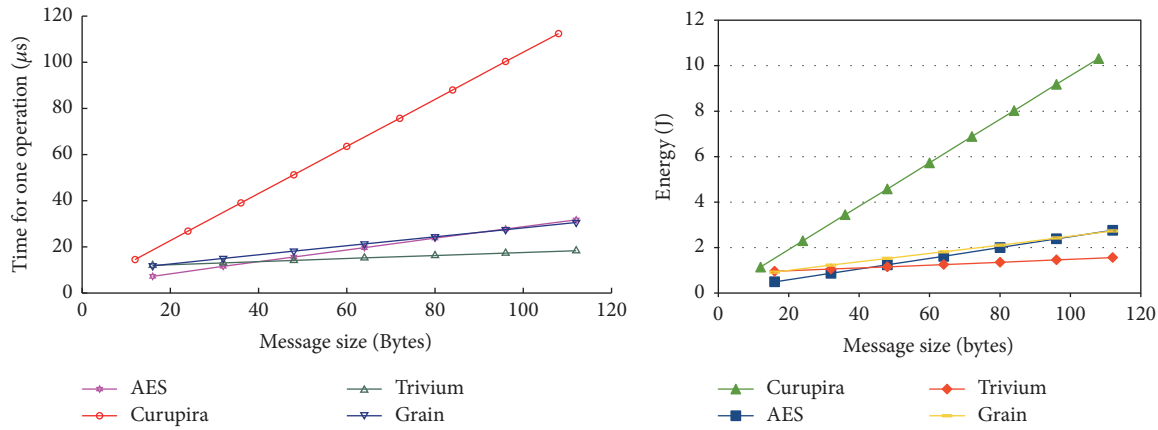

Figure 10: Average time and energy to execute one *encryption* operation for each block cipher.

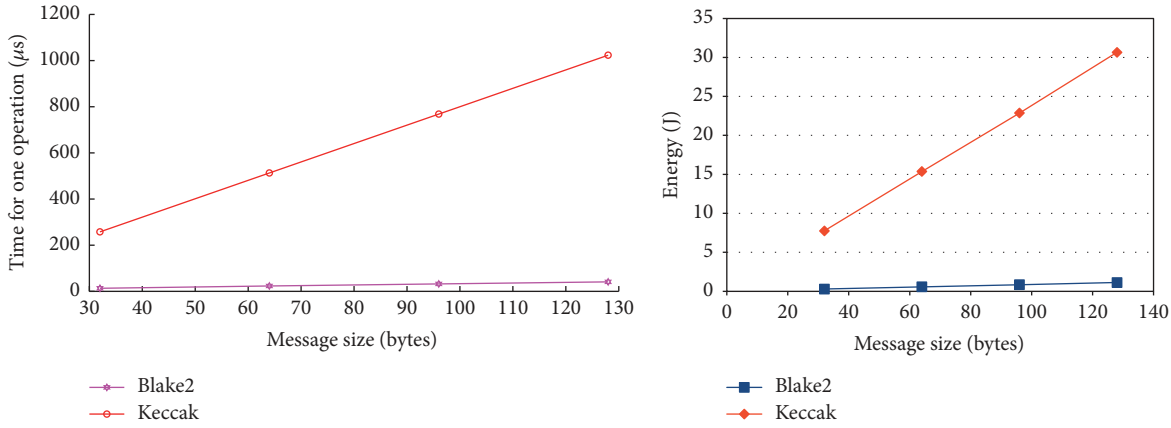

Figure 11: Average time and energy to execute *update* operation for each hash algorithm.

Table 16: Time to execute one *init* operation for each hash algorithm. Time is expressed in μs and D is the data size in bytes.

D	Blake2	Keccak
128	5.95 ± 0.01	27.23 ± 0.01

Table 17: Energy consumption to execute N *init* operations for each hash algorithm. The consumption is expressed in J.

Blake2	Keccak
0.12 ± 0.00	0.79 ± 0.01

Table 18: Time of *final* operation for each hash algorithm expressed in μs. D is the data size in bytes.

D	Blake2	Keccak
128	24.09 ± 0.01	2.13 ± 0.01

Table 19: Energy consumption for N *final* operations for the hash algorithms expressed in J.

Blake2	Keccak
0.59 ± 0.00	0.01 ± 0.00

faster than Keccak's. This operation has a roughly constant execution time for all data sizes.

Figure 11 illustrates the execution time and energy consumption comparisons for the *update* operation between two hash algorithms. Blake2 presents performance one order of magnitude (or even more depending on the message sizes) better than Keccak.

Finally, Tables 18 and 19 show the execution time for the *final* operation which is roughly constant for all data sizes and only results for one input size are shown. Keccak is about 11x faster and less energy consuming than Blake2 for the analyzed operation.

It is worth observing that for smaller input sizes (<32 bytes) both algorithms have similar performance considering the combination *update* + *final*, but as the input length grows the Blake2 relative performance is progressively improved.

Tables 20 and 21 show the execution time of *init* operation for each *MAC algorithm*. The Marvin execution time is roughly constant for data sizes $12 \leq D \leq 108$ and only figures for $D = 108$ are shown. Marvin takes ≈2x longer to initialize than HMAC.

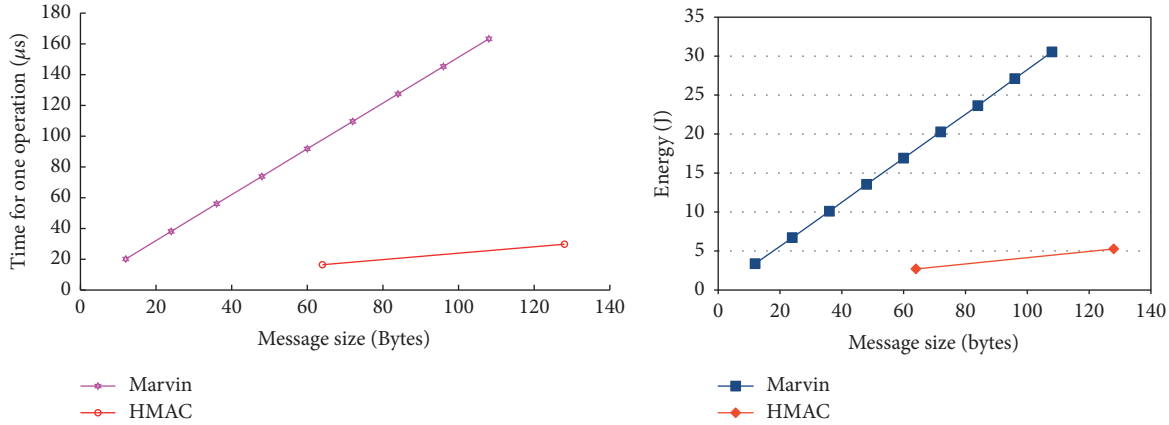

Figure 12: Average time and energy of the *update* operation for each MAC algorithm.

Table 20: Time of *init* operation for each MAC algorithm expressed in μs. D is the data size in bytes.

D	Marvin	HMAC
108	62.84 ± 0.20	
128		32.90 ± 0.02

Table 21: Energy consumption in J to execute N *init* operations for each MAC algorithm.

Marvin	HMAC
11.32 ± 0.02	5.82 ± 0.01

Table 22: Time of *final* operation for each MAC algorithm expressed in μs. D is the data size in bytes.

D	Marvin	HMAC
108	80.26 ± 0.03	
128		39.15 ± 0.05

Table 23: Energy consumption for N *final* operations for each MAC algorithm expressed in J.

Marvin	HMAC
14.77 ± 0.02	7.06 ± 0.01

Figure 12 compiles the benchmarks for the *update* operation for the MAC algorithms. HMAC presents a much better performance, executing this procedure for 64 bytes faster than Marvin with 12 bytes.

Finally, Tables 22 and 23 present the performance of MAC *final* operation. Marvin execution time is roughly constant for all data sizes $12 \leq D \leq 108$, so only the result for $D = 108$ is shown. HMAC performs 2x faster than Marvin for the *final* operation.

Tables 24 and 25 show the execution time and energy consumption of the *init* procedure for *AEAD algorithms*. LetterSoup initialization is the fastest one with a slight difference comparing to OCB. Ketje has the slowest initialization.

Table 24: Time to execute one *init* operation for each AEAD algorithm. Time is expressed in μs and D is the data size in bytes.

D	EAX	OCB	LTS	Ketje
108	33.25 ± 0.03	20.98 ± 0.03	20.30 ± 0.03	70.98 ± 0.05

Table 25: Energy consumption to execute N *init* operations for each AEAD algorithm. The consumption is in J.

EAX	OCB	LTS	Ketje
0.46 ± 0.00	0.28 ± 0.00	0.27 ± 0.00	1.01 ± 0.01

Figure 13 illustrates the execution time and energy consumption of the *encryption with associated data* operation. Ketje remains the slowest algorithm in this experiment. It shows that LetterSoup presents a better performance than OCB when there is associated data.

Since the decryption operation displays similar performance for time and energy, the detailed results are also omitted here.

According to the results described in this section, the run time and energy consumption experiments agree with each other. Let D be the data size; the recommendation of cryptographic algorithms for the Intel Edison is as follows:

(i) Symmetric block cipher: *AES* if $D \leq 32$, and *Trivium* if $D > 32$

(ii) Hash: *Blake2*

(iii) MAC: *HMAC*

(iv) AEAD: *OCB* when there is no associated data, and *LetterSoup* when there is associated data

8. Conclusions

We provided a detailed evaluation of symmetric cryptographic primitives providing different security services in relevant real-world platforms and operating systems, typical of IoT and WSN. We observed that some previous results in the literature only considered the (relatively) old implementations over a single platform or a single operating system.

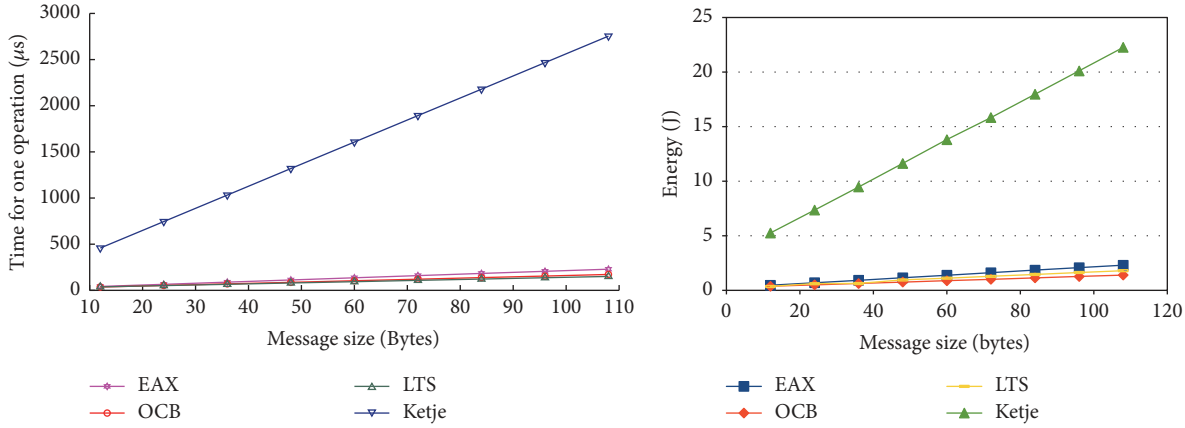

FIGURE 13: Average time and energy to execute one *encryption* operation for each AEAD algorithm with associated data.

We give some potential recommendations of algorithms depending on input data sizes. This work also provided for the first time a detailed benchmark methodology and a significant set of experiments for the Intel Edison board, a 32-bit IoT power-efficient IoT platform.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by Fundação para o Desenvolvimento Tecnológico da Engenharia (FDTE) under Grant no. 1450. Cíntia B. Margi is supported by CNPq Research Fellowship no. 307304/2015-9.

References

[1] F. Hu and N. K. Sharma, "Security considerations in ad hoc sensor networks," *Ad Hoc Networks*, vol. 3, no. 1, pp. 69–89, 2005.

[2] J. Lee, K. Kapitanova, and S. H. Son, "The price of security in wireless sensor networks," *Computer Networks*, vol. 54, no. 17, pp. 2967–2978, 2010.

[3] C. B. Margi, R. C. A. Alves, and J. Sepulveda, "Sensing as a service: secure wireless sensor network infrastructure sharing for the internet of things," in *Proceedings of theProceedings of the International Workshop on Very Large Internet of Things (VLIoT 2017) in conjunction with the VLDB 2017*, vol. 3, Munich, Germany, 2017, https://www.ronpub.com/OJIOT_2017v3i1n08_Margi.pdf.

[4] S. Hu, X. Zhang, H. Yao, and C. She, "An Android Terminal in TelosB Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Computer and Information Applications (ICCIA '12)*, December 2012.

[5] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 65–93, 2006.

[6] W. Su and M. Alzaghal, "Channel propagation characteristics of wireless MICAz sensor nodes," *Ad Hoc Networks*, vol. 7, no. 6, pp. 1183–1193, 2009.

[7] H. J. Ban, J. Choi, and N. Kang, "Fine-grained support of security services for resource constrained internet of things," *International Journal of Distributed Sensor Networks*, vol. 2016, Article ID 7824686, 2016.

[8] L. Casado and P. Tsigas, "ContikiSec: a secure network layer for wireless sensor networks under the Contiki operating system," in *Identity and Privacy in the Internet Age*, vol. 5838 of *Lecture Notes in Computer Science*, pp. 133–147, Springer, Berlin, Germany, 2009.

[9] A. Dunkels, "Contiki 3.0 released, new hardware from texas instruments, zolertia," The Official Contiki OS Blog, 2015, http://contiki-os.blogspot.ca/2015/08/contiki-30-released-new-hardware-from.html.

[10] SUPERCOP, "Mirror of SUPERCOP: System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives," 2015, https://github.com/floodyberry/supercop.

[11] I. Corporation, "Intel edison product brief," 2015, http://download.intel.com/support/edison/sb/edison_pb_331179001.pdf.

[12] O. Hyncica, P. Kucera, P. Honzik, and P. Fiedler, "Performance evaluation of symmetric cryptography in embedded systems," in *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2011*, pp. 277–282, September 2011.

[13] Electronic codebook (ecb), 2016, https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Electronic_Codebook_.28ECB.29.

[14] M. A. Simplicio Jr., B. T. De Oliveira, C. B. Margi, P. S. L. M. Barreto, T. C. M. B. Carvalho, and M. Näslund, "Survey and comparison of message authentication solutions on wireless sensor networks," *Ad Hoc Networks*, vol. 11, no. 3, pp. 1221–1236, 2013.

[15] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 162–175, November 2004.

[16] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: a secure sensor network communication architecture," in *Proceedings of the 2007 6th International Symposium on Information Processing in Sensor Networks*, pp. 479–488, New York, NY, USA, April 2007.

[17] J. Hill, R. Szewczyk, W. Alec, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, 2000.

[18] P. Serrano, A. Garcia-Saavedra, G. Bianchi, A. Banchs, and A. Azcorra, "Per-frame energy consumption in 802.11 devices and its implication on modeling and design," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1243–1256, 2015.

[19] F. Kaup, P. Gottschling, and D. Hausheer, "PowerPi: measuring and modeling the power consumption of the raspberry Pi," in *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks, (LCN '14)*, pp. 236–243, September 2014.

[20] G. De Meulenaer, F. Gosset, F. Standaert, and O. Pereira, "On the energy cost of communication and cryptography in wireless sensor networks," in *Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communication (WiMob '08)*, pp. 580–585, Avignon, France, October 2008.

[21] "Embedded linux board comparison," 2014, https://learn.adafruit.com/embedded-linux-board-comparison/power-usage.

[22] "Measured power consumption of intel edison," 2016, https://www.scivision.co/measured-power-consumption-of-intel-edison/.

[23] TinyOS, 2015, http://www.tinyos.net/.

[24] P. Levis, S. Madden, J. Polastre et al., "TinyOS: an operating system for sensor networks," in *Ambient Intelligence*, pp. 115–148, Springer, Berlin, Germany, 2005.

[25] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki—a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th IEEE Annual International Conference on Local Computer Networks (LCN '04)*, pp. 455–462, November 2004.

[26] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals," RFC 4919 4919, IETF, 2007, http://www.ietf.org/rfc/rfc4919.txt.

[27] T. Winter, P. Thubert, A. Brandt et al., "RPL: IPv6 routing protocol for low-power and lossy networks," RFC 6550 6550, 2012, http://www.ietf.org/rfc/rfc6550.txt.

[28] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: an application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[29] A. Dunkels, O. Schmidt, and T. Voigt, "Using protothreads for sensor node programming in," in *In Proceedings of the REALWSN 2005 Workshop on RealWorld Wireless Sensor Networks*, 2005.

[30] The yocto project, 2015, https://www.yoctoproject.org/.

[31] N. F. Standard, "Announcing the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197 NIST FIPS 197, 2001.

[32] P. Barreto and M. Simplicio, "Curupira, a block cipher for constrained platforms, in: Anais do 25o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'07, 2007".

[33] S. Panasenko and S. Smagin, "Lightweight cryptography: underlying principles and approaches," *International Journal of Computer Theory and Engineering*, pp. 516–520, 2011.

[34] M. Simplicio, P. Barreto, T. Carvalho, C. Margi, and M. Naslund, "The Curupira-2 block cipher for constrained platforms: Specification and benchmarking," in *Proceedings of the 1st International Workshop on Privacy in Location-Based Applications - 13th European Symposium on Research in Computer Security (ESORICS '08)*, vol. 397, 2008, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-397.

[35] C. De Cannière and B. Preneel, "Trivium," in *New Stream Cipher Designs*, vol. 4986 of *Lecture Notes in Computer Science*, pp. 244–266, Springer, Berlin, Germany, 2008.

[36] M. Hell, T. Johansson, and W. Meier, "Grain: a stream cipher for constrained environments," *International Journal of Wireless and Mobile Computing*, vol. 2, no. 1, pp. 86–93, 2007.

[37] J. P. Aumasson, S. Neves, Z. Wilcox-OΓHearn, and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," *Applied Cryptography and Network Security*, pp. 119–135, 2013.

[38] NIST, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family," Tech. Rep., Department of Commerce, 2007, http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.

[39] "gvanas, Keccak code package," 2014, https://github.com/gvanas/KeccakCodePackage.

[40] P. FIPS, "The Keyed-Hash Message Authentication Code (HMAC)," FIPS PUB 198-1 NIST FIPS 198-1, National Institute of Standards and Technology, 2008.

[41] M. A. Simplicio, B. Pedro Aquino, P. S. L. M. Barreto, T. C. M. B. Carvalho, and C. B. Margi, "The Marvin message authentication code and the LetterSoup authenticated encryption scheme," *Security and Communication Networks*, vol. 2, no. 2, pp. 165–180, 2009.

[42] H. Lipmaa, P. Rogaway, and D. Wagner, "CTR-mode encryption," *First NIST Workshop on Modes of Operation, Citeseer*, 2000.

[43] M. Dworkin, "NIST Special Publication 800-38C: The CCM Mode for Authentication and Confidentiality," US National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf.

[44] P. Rogaway, M. Bellare, and R. S. Ferguson, "OCB: a block-cipher mode of operation for efficient authenticated encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[45] P. Rogaway, "OCB: Background," 2015, http://web.cs.ucdavis.edu/rogaway/ocb/ocb-faq.htm.

[46] M. J. Dworkin, "Nist special publication 800-38b," Recommendation for Block Cipher Modes of Operation: The cmac mode for authentication NIST SP 800-38b, 2016.

[47] M. A. Simplicio Jr., B. T. De Oliveira, P. S. L. M. Barreto, C. B. Margi, T. C. M. B. Carvalho, and M. Naslund, "Comparison of authenticated-encryption schemes in wireless sensor networks," in *Proceedings of the 36th Annual IEEE Conference on Local Computer Networks, (LCN '11)*, pp. 450–457, October 2011.

[48] M. Simplicio, LetterSoup implementation, 2015, http://www.larc.usp.br/mjunior/files/algs/8%20bits/LetterSoup/LetterSoup.c.

[49] M. Simplicio, "AEAD implementations," 2015, http://www.larc.usp.br/mjunior/files/algs/8%20bits/algs8bits.zip.

[50] M. Simplicio, "Curupira-2 implementation," 2015, http://www.larc.usp.br/mjunior/files/algs/8%20bits/Curupira-2/Curupira-2.zip.

[51] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, CAESAR submission: Ketje v1, 2014, http://competitions.cr.yp.to/round1/ketjev11.pdf.

[52] S. Pelissier, "Application using trivium with 16-bit microcontroller," 2009, https://github.com/tyll/tinyos-2.x-contrib/tree/master/crypto/apps.

[53] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7954, pp. 119–135, 2013.

[54] O. Gay, "HMAC implementation," 2015, https://github.com/ogay/hmac.

[55] M. Simplicio, Marvin implementation, 2015, http://www.larc.usp.br/mjunior/files/algs/8%20bits/Marvin/Marvin.c.

[56] M. Bellare, P. Rogaway, and D. Wagner, "The EAX mode of operation," in *Fast Software Encryption: 11th International Workshop (FSE 2004)*, B. Roy and W. Meier, Eds., pp. 389–407, Springer, Berlin, Germany, 2004.

[57] T. Krovetz and P. Rogaway, "The OCB authenticated-encryption algorithm," RFC Editor RFC7253, 2014, http://www.cs.ucdavis.edu/rogaway/papers/ocb-id.htm.

[58] Agilent, "Agilent 34401A Multimeter," 2007, http://cp.literature.agilent.com/litweb/pdf/5968-0162EN.pdf.

[59] L. P. Manual and gettimeofday., 2015, http://man7.org/linux/man-pages/man2/gettimeofday.2.html.

[60] C. B. Margi, B. T. De Oliveira, G. T. De Sousa et al., "Impact of operating systems on Wireless Sensor Networks (security) applications and testbeds," in *Proceedings of the 2010 19th International Conference on Computer Communications and Networks, (ICCCN '10)*, August 2010.