Analyzing the Resource Utilization of AES Encryption on IoT Devices

Pedro Sanchez Munoz*, Nam Tran†, Brandon Craig‡, Behnam Dezfouli§, and Yuhong Liu¶
Internet of Things Research Lab, Department of Computer Engineering, Santa Clara University, USA
*psanchezmunoz@scu.edu, †nvtran@scu.edu, ‡bcraig@scu.edu, §bdezfouli@scu.edu, ¶yhliu@scu.edu

***Abstract*—With the prosperity of the Internet of Things (IoT), the implied security issues demand heightened attention. However, conventional cryptographic security solutions often lead to heavy computations and high energy costs, which can hardly be afforded by resource-constrained IoT edge devices. Therefore, it is essential to retrieve real testing data and investigate the trade-off between information security and its resource consumption on IoT edge devices. This paper explores the duration and energy consumption of the Advanced Encryption Standard (AES), implemented through both software and hardware with various key and buffer size settings on two resource-constrained IoT edge devices. In particular, we observe that (1) compared to software, the hardware implementation is more sensitive to buffer size settings and only consumes lower overall time and energy when the buffer size is sufficiently large; (2) the security premium provided by an increase in key size leads to increased resource consumption in all cases; and (3) comparing the two IoT boards, the CYW board, which is designed with faster default CPU clock rate and more memory, consumes fewer resources overall than the BCM board. These observations not only help advance the understanding of the trade-off between IoT devices' security needs and resource consumption, but also shed light on future improvement of light-weight security designs.**

I. Introduction

As of 2016, there were approximately 6 billion connected Internet of Things (IoT) edge devices world-wide and there will be 20 billion connected by 2020 [1]. These IoT edge devices can be found in a variety of "smart" objects such as smart heaters, fridges, cars, lights, doors, and other everyday products. The IoT edge devices which typically host a variety of sensors for temperature, pressure, humidity, light, motion and acceleration are frequently resource-constrained in a multitude of ways. For example, many of them are battery-powered. Their processing power is often designed to be just sufficient for the task at hand so that they can be mass-produced while minimizing costs. They often heavily rely on threading on a real-time operating system to extract all possible performance out of their hardware. On the other hand, the amount of data produced by these sensors at this scale is staggering. This data is valuable for analysis and predictions, but the security implications are worrisome. As time passes, these devices will increasingly be used as interfaces to the real world, such as medical monitoring on a patient or navigation systems in a car [2]. The security risks are suddenly made real and potentially life-threatening.

There are a variety of challenges to be addressed with respect to securing the IoT world. One of the most critical security challenges is the pervasive trade-off between data security and device performance, especially given that the typical IoT edge device is resource-constrained. Security implementations must share the processing power, scheduling time, and energy available to the board. The more security demanded, the more time and energy required by the device. It is of utmost importance to understand and compare the time and energy consumption of existing security solutions in order to streamline their use in the IoT market.

Although research has been conducted to evaluate the trade-off between different security choices and their resource consumption on IoT boards [3]–[8], many of these studies focus on more powerful devices such as mobile phones or custom FPGA implementations. In addition, the studies using resource-constrained IoT devices are often ad hoc and only focus on specific testing platforms, making it difficult to be generalized.

This paper seeks to evaluate the time and energy efficiency required to perform AES encryption on two state-of-the-art platforms with various settings. These are modern, resource -constrained, IoT edge devices. By analyzing the real data collected from our testing platform, this work brings an update to the field's understanding of the performance metrics for modern IoT edge devices running the AES algorithm through either software or hardware implementations. Note that our hardware acceleration is provided by a Cypress Semiconductor "cryptography core" available on certain production IoT platforms, specifically the CYW board that we use, in contrast to the implementations in other work that use FPGA devices, such as one by Mali et al. [7]. Furthermore, this paper will shed light on further improvement of lightweight security designs for resource-constrained IoT edge devices in the future.

The rest of this paper is organized as follows. Section II outlines an overview of previous works that establishes a baseline for comparison with our results. Section III briefly introduces the basics of AES. In Section IV, the evaluation methodology is discussed in detail, which includes the characteristics of the two testing platforms, the customized energy measurement tool developed by a previous work, and the design choices made to obtain clear and reliable data. The experimental data is analyzed in Section V, followed by the conclusion in Section VI.

II. Related Work

The study of cryptographic performance on IoT devices has been previously explored on other platforms. For example,

Potlapally et al. [5] focus on battery life, a critical constraint of devices. Their study provides a comprehensive analysis of Security Sockets Layer (SSL) and the underlying cryptographic algorithms. They demonstrate that cryptographic algorithms have significant impact on battery life and memory consumption. Though outdated, the conclusions provide an idea of where cryptographic algorithms stood with respect to performance and thus serve as a reference point to measure their improvement. A relevant conclusion drawn is that AESs energy consumption increases with the chosen key size. Similarly, Salama et al. [4] also examine an 8% increase of energy and encryption duration between the 128-bit and 192-bit key size options, and a 16% increase between 128-bit and 256-bit key sizes.

There are some studies comparing the performance of hardware and software implementations of AES. For example, Mali et al. [7] test a hardware implementation of AES on an FPGA versus a software implementation. They conclude that hardware is faster with respect to encryption and decryption, 23 and 29 times respectively. However, they also find that software's key expansion is 1.88 times faster than hardware and that the most costly operation in using hardware acceleration is the Direct Memory Access (DMA) transfer.

Diehl et al. [6] investigate AES's "lightweight" software versus hardware cryptography across a diverse set of platforms and specifications. A conclusion is reached that software implementations typically use 10% less energy than their corresponding hardware versions during operation. However, the increased throughput of custom hardware allows it to have a reduced overall energy requirement and be the ultimate best option for situations where long-term energy consumption is constrained.

Panait et al. [3] measure the performance of an AES implementation on the ATmega128RFA1 microcontroller. The microcontroller features a 16 MHz CPU that is equipped with a hardware accelerated cryptography engine that performs AES 128-bit encryption. The ATmega128RFA1 device is purposely chosen to obtain a benchmark for wireless sensor nodes, low-power devices that operate independently. These nodes often operate without an external power source, so accomplishing device longevity makes minimal energy consumption a priority. Energy and time metrics are obtained on the ATmega128RFA1 for the CFB AES mode in both software and hardware AES-128 implementations. A noteworthy observation for our evaluation is that average power consumed is independent of the plaintext size for both styles of implementation. Also, hardware and software AES implementations on the ATmega128RFA1 consume nearly an equal amount of energy.

Hodjat et al. [8] propose a variety of design decisions to maximize throughput/area for a hardware AES implementation. The main decision is to implement pipelining independently for both key expansion and the encryption rounds. While their contributions with respect to minimizing area are valuable given the small space on IoT edge devices, they also provide optimization with respect to power consumption. Specifically with regards to the key expansion, they point out that the round keys do not change as frequently as data. Therefore, their design only calculates the round keys for each session rather than for each encryption.

As a summary, there are few studies focusing on the resource consumption of cryptography algorithms on resource-constrained IoT edge devices. In addition, the studies using resource-constrained IoT edge devices are either outdated or often ad hoc and platform specific, resulting in inconsistencies in some of their conclusions, such as the comparison between hardware and software implementations.

Therefore, in this work, we focus on providing an updated set of measurements with respect to energy and encryption duration on two popular IoT platforms. Specifically, to ensure the generality of the observations made from this work, we choose one of the platforms as best suited for computation-heavy workloads, and the other for low-cost and scalable applications. The technical differences between the two are explored more thoroughly in section IV. Furthermore, we also test how design differences influence AES performance by examining different key sizes as 128/192/256 bits and buffer sizes as 16/128/512/2048 bytes. The experiment results quantify the pervasive trade off between security and resources. In addition, we involve both hardware and software implementations of AES to compare their performance. The ultimate goal is to help future designs of lightweight security solutions by providing real performance data for resource-constrained IoT devices.

III. An Overview of Advanced Encryption Standard

Currently, the Advanced Encryption Standard (AES) is one of the most widely used symmetric key cryptosystems in IoT edge devices. Given its widespread usage, it is critical for AES to be fast and energy-efficient. In this section, a brief introduction on AES is provided with major focuses on its resource consumption and security benefits[1]. AES is composed of two main parts: key schedule and round transformation.

A. Key Schedule

Key schedule is a process that derives round keys from the cipher key. In particular, the cipher key, with size N_k, is the key that the user interacts with. These round keys are used in round transformations in order to perform AES encryption/decryption. Note that the increasing key size requires more rounds of key scheduling and thus greater resource consumption.

The key schedule contains four major components as: (1) WordSub (i.e., f_W), which maps an input word (4 bytes) to an output word, (2) RotWord (i.e., f_R), which returns 4 bytes permuted cyclically such that the bytes (a, b, c, d) would become (b, c, d, a), (3) round constants (i.e., R_c) which are defined such that the i^{th} round constant is a word containing $2^{\text{i-1}}$ followed by zeros, and (4) dependencies over past round keys. Note that i starts at 1, not 0.

[1]For more details, the interested reader is encouraged to refer to [9], [10].

The first round key is exactly the cipher key. Let the next round key be $w[i]$. In general, the equation used to get the next round key is:

$$w[i] = w[i-1] \oplus w[i-N_k] \tag{1}$$

For words in positions that are multiples of N_k, $w[i-1]$ is replaced in equation 1 by:

$$w[i-1] = R_c[i] \oplus f_W(f_R(w[i-1])) \tag{2}$$

B. Round Transformations

The round transformation process uses round keys to perform AES encryption/decryption. A complete encryption is defined as the final state after round transformations are performed for each round key. As a result, an increase in the number of round keys will cause more round transformations and increase the resources consumed for a complete encryption.

The round transformations are composed of four functions: ByteSub, ShiftRow, MixColumn and AddKey. These functions operate on the state, which is a rectangular array of four rows and N_k columns, for which each element is a byte.

The ByteSub function returns a byte from a corresponding input byte. The ShiftRow Function cyclically shifts rows of the state over different offsets. Specifically, the first row (i.e., r_0) is never shifted. The remaining rows are shifted cyclically based on N_k where the last row (i.e., r_3) will be shifted the most. The MixColumn function provides a mapping over which the state's columns will be "mixed" by application of an XOR. Note that in the final round, the MixColumn step is excluded in order to better support hardware acceleration. The AddKey function simply returns the current round key XOR with the state.

AES provides certain desirable properties that contribute to its security. The ByteSub/WordSub functions provide non-linearity with their use of a multiplicative inverse over a finite field. Although it is often done using an S-Box (lookup table), the same property is achieved. The ShiftRow function provides diffusion across rows and the MixColumn function provides diffusion across columns. In combination and over multiple rounds, these functions provide full diffusion such that each output bit is dependent on every input bit. The AddKey function provides secrecy/key dependency over the corresponding round keys. These properties provide security against a variety of attacks including linear cryptanalysis [11]. Its keyspace is considered too large for brute-force attacks to be practical. It is also simple, which allows for the use of hardware acceleration and also mitigates the risk of implementation vulnerabilities [9].

IV. Methodology

In this section, we discuss the IoT edge devices that are used as our testing platforms, our customized energy measurement platform, and the evaluation parameters in detail.

TABLE I
Testing Platforms and Features

Device	CYW43907 (CYW)	BCM4343 (BCM)
MCU	ARM Cortex R4	ARM Cortex M4
Word Size	32-bit	32-bit
Clock Frequency	320 MHz	100 MHz
SRAM	2 MB	128 KB

A. Testing Platforms

The features of the two IoT edge devices employed in this work are summarized in Table I. In particular, we use CYW43907 (CYW) [12] as the first testing platform. It is an embedded wireless system-on-a-chip (SoC) manufactured by Cypress Semiconductor. Featuring an ARM Cortex-R4 applications processor, it is optimized for IoT computation-heavy applications. In addition, we also use BCM4343 (BCM) [13] as the second testing platform, which is another SoC proposed for IoT but offers less processing power from its ARM Cortex-M4 processor.

It should be noted that the CYW board's price is higher than that of the BCM board as it has a faster default CPU clock rate, more memory, more I/O, and an on-board cryptography engine with support for hardware-accelerated AES. The BCM does not boast such an engine and therefore does not support hardware-accelerated AES. There are use cases for both boards. The CYW board is better suited for computation-heavy tasks due to its extra resources. The BCM board is better for large-scale deployments due to its reduced cost. This contrast between devices allows us to analyze AES performance across both computation-heavy and scalable IoT scenarios.

B. Energy Measurement Tool

This subsection explores the high-performance Energy Measurement Platform for Wireless IoT Devices (EMPIOT) proposed in a previous work [14]. Featuring a sampling rate of approximately 1000 Hz, EMPIOT is accurate to 0.4 μW in its energy measurement. This tool can run on a variable power source without compromising accuracy. Most impressively, the EMPIOT platform is known to boast less than 3% of energy measurement error for IoT devices using 802.15.4 or 802.11 wireless standards. This capable and accurate measurement platform is used to collect all energy data presented in this work.

Fig. 1 and Fig. 2 illustrate EMPIOT's components and interconnections with IoT boards. The RaspberryPi controlling EMPIOT provides user interface through a monitor, with keyboard and mouse inputs similar to a computer. In particular, the SoC being measured interfaces with the RaspberryPi. The SoC is connected via two general-purpose input/output (GPIO) pins. The output signals from the SoC GPIO pins act as triggers to the measurement sequence on the Raspberry Pi. Once activated by a starting trigger, the device measures the values of current (resolution of 100 μA) and voltage (resolution of 4 mV) through the tested SoC.

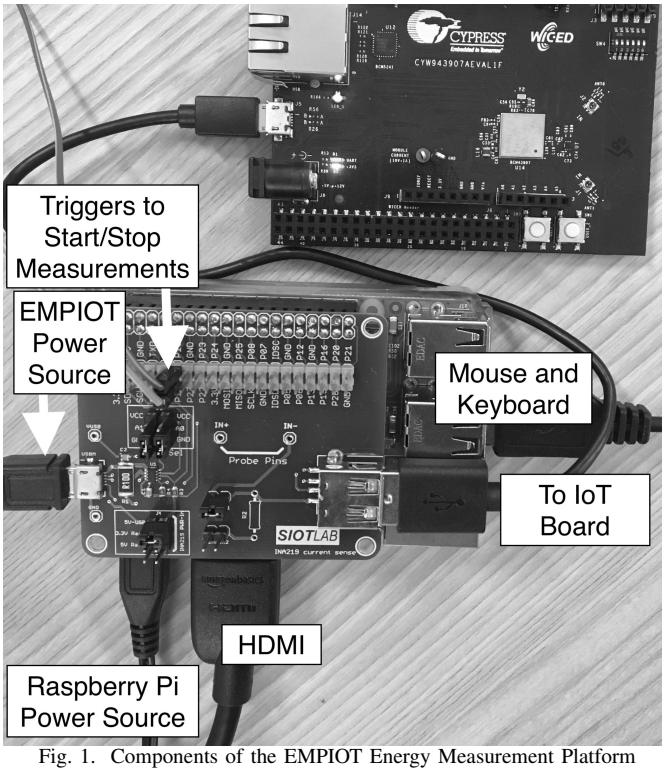

Fig. 1. Components of the EMPIOT Energy Measurement Platform

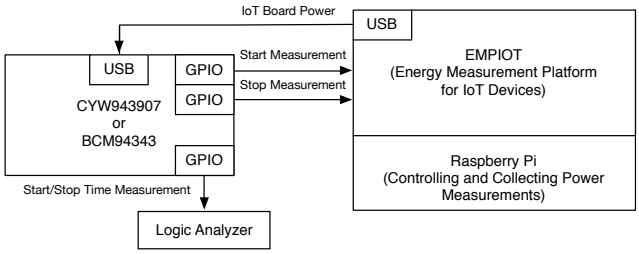

Fig. 2. Interconnection with the EMPIOT Energy Measurement Platform

It should be noted that the limitations of EMPIOT are also considered. Although EMPIOT can run on a variable power source without compromising accuracy, to ensure consistent sampling, the wall outlet connecting to the energy measurement platform remains unchanged throughout the tests. Also, it has been shown that during warm-up, the first 5 samples measured after initialization are not accurate. As such, there is a 3 millisecond delay implemented between the initialization of the EMPIOT platform and the collection of data. All collected data from a session is stored into a text file stored within the onboard memory that can be retrieved using a USB drive.

C. AES Implementation

Cypress Semiconductor's WICED SDK provides a library of cryptographic implementations [15]. The experiments are conducted using software and hardware implementations of AES-CBC from the WICED security library version 6.0.1. To provide its AES functionality, the WICED security library uses the mbedtls' free, open source library. Cipher block-chaining mode (CBC) is used in all experiments. This mode is popular since it prevents a given plaintext mapping to a single ciphertext, while only requiring a single public initialization vector (IV).

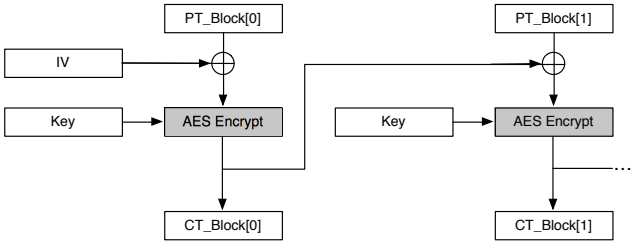

Fig. 3. AES-CBC Encryption

Fig. 3 depicts how AES-CBC encryption works. The implementation requires the use of a context structure to hold the round keys and buffers for the IV, plaintext blocks (i.e., PT_Block) and resulting ciphertext blocks (i.e., CT_Block). First, a buffer holding the key is copied into the context structure. Then, the IV and plaintext buffers are prepared. Note that the buffers are passed in by address. Finally, these are used in a function call as arguments to the `crypt_cbc()` function which will write the ciphertext at the address provided. It is found that the only affected variables are the IV, since it is used in each block encryption, and the ciphertext buffer, which now holds the corresponding ciphertext.

D. Evaluation Process

In this work, the evaluations focus on a single-threaded scenario. In a single thread, the performance of AES is tested across the following parameters: hardware/software implementations, different key sizes and buffer sizes. The evaluation follows three steps:

1) Initialization: Before each experiment, an AES-CBC key size parameter is set to the desired length as 128, 192, or 256 bits, and a plaintext buffer size parameter is set to the desired size as 16, 128, 512, or 2048 bytes. A library function completes the key schedule for all trials.

2) Starting Energy Measurement: The first general GPIO trigger is enabled at time T_s, signaling the EMPIOT to begin energy measurement. Using a WICED software or hardware AES function, AES-CBC encryptions of the same plaintext buffer are repeated for *N* times. Please note that although the overhead of the `for` loop structure and a linear time C Standard Library `memcpy` function that resets the AES initialization vector (IV) is included, the measurements show that the time taken by these operations is negligible.

3) Completing Energy Measurement: Following the conclusion of the Nth encryption at time T_e, a second GPIO pin is enabled, signaling the EMPIOT to conclude energy measurement. Over the interval $[T_s, T_e]$, the EMPIOT collects a record of the board's instantaneous current and voltage at approximately every 1/1000 seconds. Representing the total number of records taken over the interval as M, the following

equation is used to obtain total energy consumption E_{total} (J) over the interval:

$$E_{total} = \sum_{i=1}^{M} \frac{(I_i V_i + I_{i-1} V_{i-1})}{2} (t_i - t_{i-1}), \quad (3)$$

where I and V represent current and voltage respectively. The product in the summation represents a trapezoid under the IV versus time curve, whose area is the energy consumed since the previous sample was taken. For each specific key size and buffer size, the encryption process is repeated by N = 400,000 times, and the energy consumption per encryption with unit as μJ/block is calculated as follows:

$$\bar{E} = \frac{E_{total}}{N \times B} \quad (4)$$

where B is the number of blocks in the buffer per encryption. B is calculated by dividing the corresponding buffer size (in bytes) by 16 bytes (size of a standard AES block).

Please note that in the experiments the above procedure has been repeated k times and the mean and standard deviation of $\bar{E}$ are calculated. For the CYW and the BCM boards, the software implementations of AES with three different key sizes are tested by repeating the above process. In addition, as the CYW43907's ARM R4 processor contains an on-chip cryptography core supporting AES, the process is repeated for each key size using the hardware AES implementation.

V. Experimental Results

In this section, the performance evaluation results are presented. First, a discussion is provided with respect to energy consumption across the platforms. Then, the duration data is discussed similarly.

A. Energy Consumption

As discussed before, we have collected the energy consumption of three AES implementations (i.e., BCM software, CYW software and CYW hardware) with different key sizes and buffer sizes. The results are demonstrated in Fig. 4, Fig. 5 and Fig. 6.

In particular, Fig. 4, Fig. 5 and Fig. 6 represent energy consumption (measured in microjoules μJ) for AES with key size as 128, 192, and 256 bits respectively. Each figure has three groups of bars from left to right, representing three AES implementations as BCM software, CYW hardware and CYW software. Each group contains four bars representing buffer sizes as 16, 128, 512, and 2048 bytes respectively. It is important to note that when the energy data is gathered, some of the results include slight variations across experiments with the same parameters. The greatest standard deviation is seen in the energy data for the 2048-byte buffer size implementation of 256-bit key variant, as shown in Fig. 6, which is 0.44 μJ between experiments. For measurements over other buffer size/key size implementations, the variations are on the order of 10^{-3} μJ to 10^{-5} μJ, which are negligible. We have shown the standard deviations through error bars on these figures.

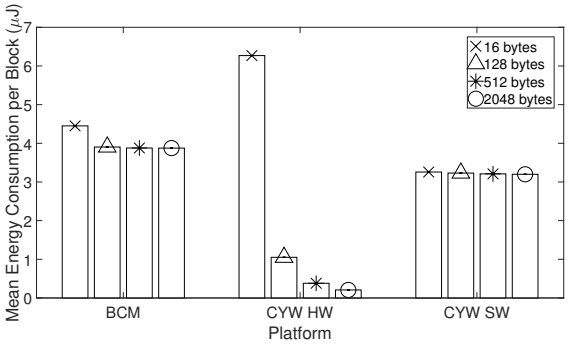

Fig. 4. This graph depicts buffer size (bytes) versus mean energy consumption per block (μJ) for 128-bit key size. As buffer size increases, CYW HW's energy consumption becomes much lower than the other platforms'. BCM SW sees a slight reduction in cost at the smallest buffer sizes while the rest of the implementations remain relatively unchanged.

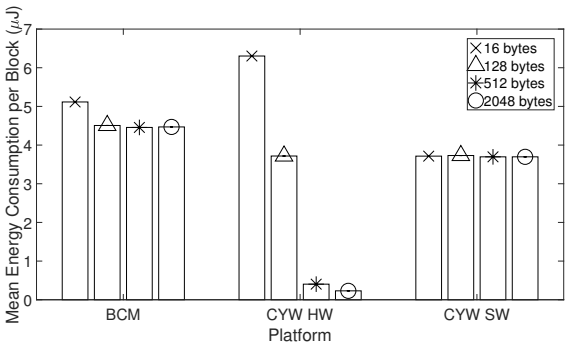

Fig. 5. This graph depicts buffer size (bytes) versus mean energy consumption per block (μJ) for 192-bit key size. As buffer size increases, CYW HW's energy consumption becomes much lower than the other platforms'. BCM SW sees a slight reduction in cost at the smallest buffer sizes while the rest of the implementations remain relatively unchanged.

However, most of them can hardly be seen due to their small values.

Based on these results, we can make the following observations. First, from Fig. 4 we observe that for the 16 byte buffer size, the two software implementations outperform the

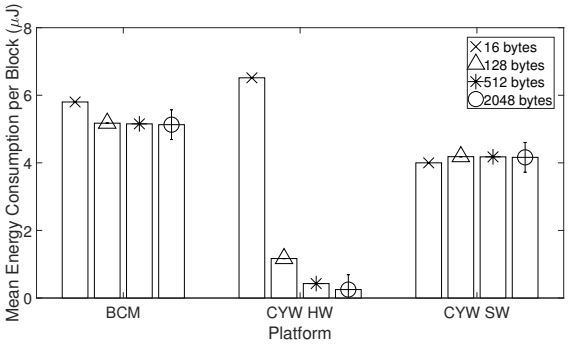

Fig. 6. This graph depicts buffer size (bytes) versus mean energy consumption per block (μJ) for 256-bit key size. As buffer size increases, CYW HW's energy consumption becomes much lower than the other platforms'. BCM SW sees a slight reduction in cost at the smallest buffer sizes while the rest of the implementations remain relatively unchanged.

hardware implementation in terms of using less energy. When the buffer size increases, the average energy consumption drops for all the three implementations, but the drop is especially large for the CYW hardware implementation. The same observation can be made from Fig. 5 and Fig. 6 as well. The reasons are as follows. The overall energy costs contain two parts: (1) the energy for performing encryption over the buffer, and (2) the energy overhead for initial setup, which is required for each encryption.

In CYW hardware implementation, as the CPU is required to program the DMA controller directly for every buffer regardless of buffer size, a high initial setup cost is required to prepare the cryptography core to perform encryption. The cost for performing the AES round transformations over the buffers is very small in comparison. In contrast, the software implementation incurs very little constant setup cost besides the function call overhead and loading parameters into memory; still it faces a relatively higher linear cost when performing round transformations over the buffers. Therefore, when the buffer size is very small, software implementations cost less energy overall, due to the relatively low setup cost. However, as the buffer size increases, the high setup cost required by the hardware implementation becomes less relevant, and the relatively lower cost of performing round transformations over the buffer becomes the main factor.

Second, comparing the software AES implementation on the BCM and the CYW boards, the CYW board requires less energy regardless of the key sizes and buffer sizes, which suggests it is particularly designed for energy efficiency when performing computation-heavy tasks. In contrast, BCM falls behind quite drastically across all scenarios. It consumes more energy and, as the next subsection demonstrates, takes more time in most scenarios. From this, one can conclude that, despite the lower up-front cost of the BCM platform, the cost of operation with respect to energy will be higher. Depending on the use case and intensity, for a sufficiently long deployment, BCM may consume more energy overall and thus cost more overall as well. BCM would significantly benefit from a cryptography core in order to support energy-efficient encryption, although this would likely interfere with its pursuit for market share within the low-cost platform market.

Third, comparing Fig. 4, Fig. 5 and Fig. 6, we observe that as the key size increases, an increase in energy consumption occurs for all three implementations. For example, based on the percentage difference between the 128-bit energy total and the 192-bit energy total, the software implementations face a higher energy trade-off for an increase in security. When the AES key length is increased from 128 bits to 192 bits and then 256 bits, 14.0% and 22.8% increases in energy consumption are observed, respectively, for the CYW software. The BCM shares a similar result, with increases of 14.9% and 30.4% respectively. The CYW hardware implementation experiences a lesser premium for security, with respective increases of 0.6% and 4.0%. This observation aligns with the working mechanism of AES as a longer key size requires more rounds of key scheduling and round transformations. For example, the 128-bit key variant will consume less energy than the 256-bit key variant, since the former runs 10 rounds of round key scheduling/transformations, whereas the latter will run 14 rounds of round key scheduling/transformations.

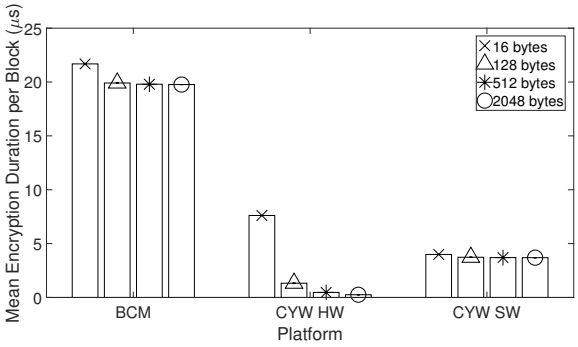

Fig. 7. This graph depicts buffer size (bytes) versus mean encryption duration per block (μs) for 128-bit key size. As buffer size increases, CYW HW's encryption duration becomes much lower than the other platforms'. BCM SW sees a slight reduction in duration and all other variants remain relatively unchanged.

B. Duration

Similar to energy consumption, we have also gathered the encryption duration of the three AES implementations with different key and buffer sizes. The results are shown in Fig. 7, Fig. 8 , and Fig. 9. In particular, Fig. 7, Fig. 8 and Fig. 9 represent encryption duration (measured in microseconds μs) for AES with key size as 128, 192, and 256 bits respectively. Each figure has three groups of bars from left to right, representing three AES implementations as BCM software, CYW hardware and CYW software. Each group contains four bars representing buffer sizes as 16, 128, 512, and 2048 bytes respectively.

It is important to note that when the timing data is gathered, some of the results include slight variations across experiments with the same parameters. For measurements over software implementations, there is consistently no variation between experiments. For measurements over hardware implementations, it is found that the standard deviations are on the order of 10^{-4} μs between experiments, which is negligible. We have also marked the standard deviations through error bars on these figures. But most of them can hardly be seen due to their small values.

From Fig. 7, Fig. 8 , and Fig. 9, we can observe that as expected, the CYW cryptography core allows the hardware implementation to outperform its competitors in terms of time. As buffer size increases, CYW hardware implementation's encryption duration decreases drastically and becomes much lower than that of other platforms. The high initial setup cost that has been discussed in the previous subsection on energy manifests itself again here. As buffer size increases, the CYW hardware implementation's setup cost becomes so negligible that it consumes significantly less time when compared to the remaining platforms.

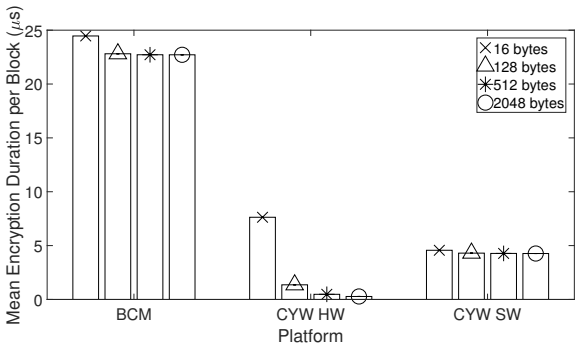

Fig. 8. This graph depicts buffer size (bytes) versus mean encryption duration per block (μs) for 192-bit key size. As buffer size increases, CYW HW's encryption duration becomes much lower than the other platforms'. BCM SW sees a slight reduction in duration and all other variants remain relatively unchanged.

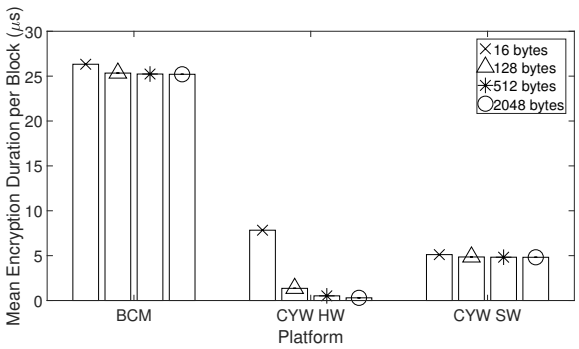

Fig. 9. This graph depicts buffer size (bytes) versus mean encryption duration per block (μs) for 256-bit key size. As buffer size increases, CYW HW's encryption duration becomes much lower than the other platforms'. BCM SW sees a slight reduction in duration and all other variants remain relatively unchanged.

The BCM's significantly longer elapsed time reflects its inferior processor speed compared to the CYW's. Again when examining the percentage increase that comes with each of the larger key sizes, we observe that both software implementations experience a 10-15% increase from 128 bits to 192 bits, and a 20-30% increase from 128 bits to 256 bits. As hinted by the significantly small energy increase discussed earlier, the hardware implementation incurs a smaller magnitude time penalty when using an increased key size.

Recall Dieh et. al's work finds that the energy consumed by a hardware chip is significantly greater than the software implementation, but the hardware chip ultimately achieves a lower overall energy consumption by virtue of being much faster with respect to time. In contrast, the CYW hardware implementation is able to achieve both less energy consumption and less execution time. The experiment results show that given sufficiently large buffer sizes the CYW hardware implementation can outperform its software competitors in both energy and time. The exact buffer size varies by key size. Based on the data, the buffer size at which CYW hardware implementation will perform better than software is roughly 64 bytes or four AES blocks.

As can be seen in Hodjat et. al [8], there are major design choices that must be made that could lead to different results with respect to hardware chip performance. It is now clear that, given the technological advances and the results gathered from this particular implementation of a cryptography core, it is now possible to take advantage of hardware-accelerated AES encryption while consuming both less energy and less time.

VI. Conclusion

This paper serves to provide an in-depth analysis of three different implementations of AES on two different IoT boards with respect to their energy and encryption duration across available key sizes and buffer sizes. Security algorithms must share resources with other computational work or I/O, so it is imperative that these algorithms be efficient to minimize the overall cost of security such that they can be effectively used alongside other functionality.

The energy measurements show that in most cases, the BCM board consumes the most energy, followed by CYW's software implementation. Finally, CYW's hardware implementation consumes the least energy. However, if buffer size is too small (i.e., 16 bytes), CYW's hardware implementation consumes more energy to encrypt a given block. This is due to the high constant cost of DMA programming required to run the cryptography core. So, CYW's cryptography core can, for a large enough buffer size, encrypt a message using AES in less time while consuming less energy than the software variants. One can use the cryptography core as a pseudo-thread capable of providing fast, energy-efficient encryption that will not use application memory and only a constant amount of CPU time, leaving valuable resources available for other tasks. Although the BCM platform is clearly trying to minimize up-front monetary cost, the energy data suggests that, given a sufficiently long lifespan, it will ultimately cost the user more in operating energy costs.

Although this work mainly focuses on the analysis of three implementations across two platforms, the trends shown in the comparison between different software implementations and between software and hardware implementations will likely remain the same with respect to boards with similar implementations and specifications. To further encourage the use of security algorithms, these algorithms must be efficient in order to share resources with other important tasks. This work shows that hardware acceleration provides a method by which one can achieve high-efficiency security algorithms. A future work, similar to Hodjat et. al [8] with a focus on increasing throughput and minimizing energy consumption, would be an important contribution towards this goal.

Measurements are only provided for AES encryption in this work. Key scheduling is discussed only for the reader to understand its implications on resource consumption. However, there are many aspects of the algorithm that are not analyzed with respect to resource consumption in this work, including but not limited to: key scheduling, modes of operation, parallelism, and choice of cryptographic algorithm. A potential next step would be implementing a multi-threaded scheme of

the same cryptographic functions. Further work with analysis over any combination of these would provide further data-driven arguments for minimizing the resource consumption of security algorithm while maintaining high levels of security.

ACKNOWLEDGMENT

This work has been partially supported by a research grant from Cypress Semiconductor Corporation (Grant No. CYP-001).

REFERENCES

[1] R. van der Meulen, "Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016," Feb 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[2] B. Dezfouli, M. Radi, and O. Chipara, "REWIMO: A real-time and reliable low-power wireless mobile network," *ACM Transactions on Sensor Networks (TOSN)*, vol. 13, no. 3, p. 17, 2017.

[3] C. Panait and D. Dragomir, "Measuring the performance and energy consumption of aes in wireless sensor networks," *Proceedings of the Federated Conference on Computer Science and Information Systems*, vol. 5, pp. 1261–1266, 2015.

[4] D. S. A. Elminaam, H. M. Abdual-Kader, and M. M. Hadhoud, "Evaluating the performance of symmetric encryption algorithms." *IJ Network Security*, vol. 10, no. 3, pp. 216–222, 2010.

[5] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the energy consumption of security protocols," in *Proceedings of the 2003 international symposium on Low power electronics and design*. ACM, 2003, pp. 30–35.

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

[7] A. Biasizzo, M. Mali, and F. Novak, "Hardware implementation of aes algorithm," *Journal of Electrical Engineering*, vol. 56, no. 9-10, pp. 265–269, 2005.

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

[9] V. Rijmen and J. Daemen, "Advanced encryption standard," *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[10] M. A. Musa, E. F. Schaefer, and S. Wedig, "A simplified aes algorithm and its linear and differential cryptanalyses," *Cryptologia*, vol. 27, no. 2, pp. 148–177, 2003.

[11] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *Journal of CRYPTOLOGY*, vol. 4, no. 1, pp. 3–72, 1991.

[12] Cypress Semiconductor, "CYW943907AEVAL1F Evaluation Kit." [Online]. Available: http://www.cypress.com/documentation/development-kitsboards/cyw943907aeval1f-evaluation-kit

[13] ——, "Avnet BCM4343W IoT Starter Kit." [Online]. Available: http://cloudconnectkits.org/product/avnet-bcm4343w-iot-starter-kit

[14] B. Dezfouli, I. Amirtharaj, and C.-C. Li, "EMPIOT: An Energy Measurement Platform for Wireless IoT Devices," *Journal of Network and Computer Applications*, 2018.

[15] Cypress Semiconductor, "WICED Studio." [Online]. Available: http://www.cypress.com/products/wiced-software