IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 18, 2020, accepted July 2, 2020, date of publication July 6, 2020, date of current version July 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3007536

Security and Performance in IoT: A Balancing Act

LUKE E. KANE[1], JIAMING JAMES CHEN[2], REBECCA THOMAS[2], VICKY LIU[2], AND MATTHEW MCKAGUE[2]
[1]Cyber Security Cooperative Research Centre, Science and Engineering Faculty, Queensland University of Technology (QUT), Brisbane, QLD 4000, Australia
[2]Science and Engineering Faculty, Queensland University of Technology (QUT), Brisbane, QLD 4000, Australia

Corresponding author: Luke Kane (le.kane@qut.edu.au)

This work was supported in part by the Cyber Security Research Centre Ltd., funded by the Australian Government's Cooperative Research Centres Program.

ABSTRACT With predictions suggesting there will be 18 billion Internet of Things (IoT) devices live by 2022, performance of these low powered devices, as well as security is of utmost importance. Managing security and performance is a balancing act. Achieving this balance will always continue to be a challenge. This research presents two main contributions to this area. The first contribution is a framework to measure cryptographic performance of IoT devices. The areas of measurement are power consumption, time cost, energy cost, random access memory (RAM) usage and flash usage. The second contribution is an insightful comparison of the performance of the ATmega328, STM32F103C8T6 and ESP8266 low powered microcontroller devices. Experiments were conducted on these devices running various cryptographic operations. The measured operations are from three encryption algorithms: Advanced Encryption Standard (AES), ChaCha and Acorn. The proposed methods from this research are real-world in nature rather than simulated, and can be used by others wishing to conduct their own IoT performance testing. The results show that the ATmega328 has the lowest overall power consumption. The ESP8266 was generally the fastest performing device. ChaCha outperformed AES in both time cost and energy cost. Both algorithms outperformed Acorn in these metrics. The STM32F103C8T6 device displayed the best overall energy cost, while still performing well in terms of time. The results from the experiments conducted in this study can be used by network designers, developers and others to make appropriate decisions in IoT deployments with regards to balancing performance and security.

INDEX TERMS AES, Acorn, ChaCha, ciphers, cryptography, the Internet of Things (IoT), lightweight encryption, power consumption.

I. INTRODUCTION

The Internet of Things (IoT) is growing at an incredible rate. In 2011, the number of Internet connected devices exceeded the entire world's population [1]. According to Ericsson, by 2022 there will be approximately 29 billion Internet enabled devices with approximately 18 billion of these being IoT devices [2]. IoT and related communications technologies underpin the development and implementation of smart cities, with IoT already being used in numerous applications including wireless sensor networks (WSN), smart homes and in critical infrastructure such as power and water [3]. With such wide adoption and integration into society, it is important that these devices can operate in a secure manner. IoT devices are often low powered devices with limited resources available [4], which means providing a sufficient standard of security needs to be balanced with the physical constraints of the device such as being battery powered, having low processing power and/or limited memory capacity. This balance of security versus performance forms the motivation of this research. Our contribution is a proposed methodology to measure the performance of common IoT devices performing cryptographic operations in the key areas of power consumption, energy cost and time cost. In addition to these main performance areas, a methodology to measure random access memory (RAM) and flash memory utilisation is also presented. The research aims were to firstly use an experimental approach rather than a simulated approach when designing our proposed methodology and secondly through the testing of cryptographic operations on common IoT devices, present and discuss recommendations on appropriate device and algorithm combinations for various scenarios.

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Esposito.

The scope of this study is the measurement of performance of cryptographic algorithms running on commonly available microcontrollers that are typical in IoT deployments. Transmission from device to device was excluded from the scope. Only operations running locally on the devices were measured. The tested operations include encryption, decryption and set key of ChaCha, Acorn and Advanced Encryption Standard (AES) Electronic Code-Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter (CTR) modes. All algorithms selected were operated with 128-bit encryption. AES was selected due to its widespread use [5]. The ChaCha algorithm was selected due to its promising performance results when compared with AES [6]. Acorn is a newer cipher and was selected due to being designed to operate on resource-constrained devices [7]. The microcontrollers selected for the experiments were the ATmega328 [8], the STM32F103C8T6 [9], [10] and the ESP8266 WIFI Witty Cloud Development Board [11]–[13]. These microcontrollers were selected due to their low cost, their accessibility, their common place usage in IoT deployments and the fact they represent three distinct architectures being AVR, ARM and Tensilica.

This research first presents a literature review which discusses some relevant background information on the AES, Acorn and ChaCha ciphers and a thorough review of relevant previous work. The microcontrollers and other resources used in the experiments are then covered. The proposed performance analysis framework is then presented and the methodology is explained in detail. Finally, the conclusion is presented with a summary and possible future work suggestions.

II. BACKGROUND

A. AES

The AES symmetric encryption algorithm was developed by two Belgian cryptographers - Daemen and Rijmen [5]. It was created in response to a request made by the United States NIST (National Institute of Standards and Technology) for a new encryption algorithm to replace the then standard Data Encryption Standard (DES) encryption algorithm, which by then was known to be vulnerable to brute force attacks [5]. In 2001, AES was published as the FIPS (Federal Information Processing Standards Publication) 197 standard [14] and subsequently was approved for use by the US federal government for protection of sensitive electronic data. Since then, its use has been widespread. AES is flexible enough to be used in applications with high security requirements such as e-commerce, as well as applications with fast processing requirements, such as those involving image or video processing [15].

The AES encryption algorithm is an iterative algorithm which processes data in blocks of 128 bits [5]. It conducts a set number of operations on each data block for a fixed number of iterations that is determined by the chosen key size. As shown in Table 1, AES supports key sizes of 128, 192 and 256 bits, and based on the key size chosen, it conducts 10, 12, or 14 operations respectively [5]. Given AES's popularity and widespread use, it was selected as one of the algorithms to be measured in this study.

TABLE 1. Number of rounds for each encryption algorithm [5].

Encryption Algorithm	Key Length (bits)	Block Size (bits)	Number of rounds
AES-128	128	128	10
AES-192	192	192	12
AES-256	256	256	14

1) MODES OF OPERATION

Cipher block modes of operations are used to dictate how to apply an encryption or decryption algorithm to a large number of data blocks [16]. As per NIST publication [16], there are over a dozen block cipher modes in AES. As mentioned in Section I, the scope of this study is limited to ECB, CBC, CFB, OFB and CTR modes. A brief comparison of these modes is provided in Table 2.

2) ECB MODE

ECB mode has the simplest implementation of the five modes of operations discussed in this study. For each encryption process as shown in Formula (1) [16], the encryption algorithm in ECB mode receives a key K and a plain-text block P_t as input and produces a cipher-text block C_t as output [16]. The decryption process as demonstrated in Formula (2) [16] occurs similarly, with the decryption algorithm receiving a key K and a cipher-text block C_t as input and produces a plain-text block P_t as output [16].

$$C_t = E(K.P_t), \quad t = 1 \ldots .N \tag{1}$$
$$P_t = D(K.C_t), \quad t = 1 \ldots .N \tag{2}$$

One of the primary drawbacks of ECB mode is that for a given key, a plain-text block will always encrypt to the same cipher-text [16]. This makes it vulnerable to known plain-text attacks like the code-book attack, where an attacker could keep track of plain-text blocks and corresponding cipher-text blocks and then use this information to their advantage to modify cipher-text blocks in transmission. It is recommended to use ECB mode only for short messages like an AES key, as it may be insecure to use with lengthy messages [5], [16]. One advantage of ECB mode is that since each data block is operated on independently, and the blocks do not influence each other, multiple data blocks can be processed at the same time. With parallel encryption and decryption, the overall processing times can be reduced. Furthermore, with ECB mode, an error in a cipher-text block will result in a deciphering error in the corresponding plain-text block only.

TABLE 2. Comparison between different modes of operations [16].

Mode of Operation	Padding Required	Error Propagation	Block/Stream	Parallel Encryption/ Decryption	Pre-computable	Self-synchronous	IV Required
ECB	Yes	Errors propagate within the current block	Block	Yes, Yes	No	Yes	No
CBC	Yes	Errors propagate within the current block and into specific bits of the next block	Block	No, Yes	No	Yes	Yes (Should be random)
CFB	No	Errors propagate to the next block and into specific bits of the current block	Block/Stream	No, Yes	No	Yes	Yes (Should be random)
OFB	No	Errors propagate into specific bits of the current block	Block/Stream	No, No	Keystream is pre-computable	Yes	Yes (Should be unique)
CTR	No	Errors propagate into specific bits of the current block	Block/Stream	Yes, Yes	Yes	Yes	Yes (Nonce should be unique)

3) CBC MODE

In CBC mode, for the encryption process demonstrated by Formula (3) [16], the plain-text block P_t is first XORed with the previous cipher-text block C_{t-1}. The output is then encrypted using a key K to produce the cipher-text block C_t. For the first encryption, C_0 is the initialisation vector (IV), which is a randomly chosen number that is sent along with the cipher-text blocks [16]. The decryption process is shown in Formula (4) [16].

$$C_t = E(P_t \oplus C_{t-1}, K), \quad t = 1....N \text{ and } C_0 = IV \qquad (3)$$
$$P_t = D(C_t, K) \oplus C_{t-1}, \quad t = 1....N \text{ and } C_0 = IV \qquad (4)$$

The addition of an IV and the involvement of the previous cipher-text block in the encryption process ensures that attackers cannot create a code-book, because for a given key, a plain-text block does not always encrypt to the same cipher-text block [16]. One of the applications of CBC mode is to provide integrity assurance. This is achieved by using the last block of a file to create a message authentication code (MAC), which is referred to as CBC-MAC [17]. The CBC-MAC is used for integrity checking and is sent along with the encrypted file to the receiver.

4) CFB MODE

Unlike the previous cipher block modes, the plain-text block is not directly fed as input to the encryption algorithm in CFB mode. Instead, to produce the cipher-text block C_t, the previous cipher-text block C_{t-1} is encrypted using key K, and the output of this is XORed with the plain-text block P_t. For the first encryption process as shown in Formula (5), the IV is used instead of the previous cipher-text block C_{t-1} [16]. The decryption process is demonstrated in Formula (6).

$$C_t = E(C_{t-1}, K) \oplus P_t, \quad t = 1....N \qquad (5)$$
$$P_t = E(C_{t-1}, K) \oplus C_t, \quad t = 1....N \qquad (6)$$

In CFB mode, the encryption algorithm itself can be used for the decryption process, in contrast to ECB and CBC modes, where a separate decryption operation is required [16]. Since the operation is the same, the resulting encryption and decryption times would be similar. A key advantage of CFB mode is its ability to self-synchronise. For instance, if a cipher-text block C_t is lost in transmission, when the receiver decrypts the next cipher-text block C_{t+1}, it would decrypt as $E(C_t, K) \oplus C_{t+1}$, resulting in an incorrect plain-text block P_t. For the next cipher-text block C_{t+2} however, the receiver would decrypt it as $E(C_{t+1}, K) \oplus C_{t+2}$, which would result in the correct plain-text block P_{t+2} and the cipher would once again be back in sync [16].

5) OFB MODE

Like CFB mode, the plain-text block is not directly fed as an input to the encryption algorithm in OFB mode. Instead, to produce the cipher-text block C_t, the output of the encryption of O_t using key K is XORed with the plain-text block P_t, where O_t is the output of the previous encryption process [16].

For the encryption of the first plain-text block P_1, O_0 refers to the IV, which is chosen at random. Thus, OFB mode is in effect a synchronous stream cipher, where O_t is a keystream that is XORed with plain-text P_t to produce cipher-text C_t during encryption, and XORed with cipher-text C_t during decryption, to produce plain-text P_t. Encryption and decryption are demonstrated by Formulas (7) and (8) respectively [16].

$$\begin{aligned} O_t &= E(O_{t-1}, K) \\ C_t &= O_t \oplus P_t, \quad t = 1....N \end{aligned} \qquad (7)$$
$$\begin{aligned} O_t &= E(O_{t-1}, K) \\ P_t &= O_t \oplus C_t, \quad t = 1....N \end{aligned} \qquad (8)$$

Like CFB mode, OFB mode also does not require a separate decryption algorithm to decrypt cipher-text. Instead the encryption algorithm itself can be used. Since the operations are the same, the encryption and decryption times are also likely to be similar.

The output of the encryption process in OFB mode can be used to produce key streams. Furthermore, unlike in CFB mode, the output of the encryption process depends only on the IV and key and is independent of the plain-text blocks being encrypted [16]. This introduces an added advantage, where certain operations in OFB mode can be pre-computed even before receiving the cipher-text or plain-text. [16].

6) CTR MODE

Like OFB mode, CTR mode is in effect a synchronous stream cipher which involves a generated keystream, a cipher-text and a plain-text. In CTR mode, the keystream O_t is generated by encrypting a concatenation of a nonce N and a counter value t that is incremented for each subsequent encryption. The nonce N used would be a randomly chosen number [16].

For both the encryption and decryption process, which can be seen in Formulas (9) and (10) respectively [16], the same counter value must be used. As best practice, it is recommended to ensure different nonce values are used for different plain-text messages to ensure the confidentiality of the plain-text is upheld [16].

$$\begin{aligned} O_t &= E(T_t, K) \quad \text{where } T_t = N||t \\ C_t &= O_t \oplus P_t \end{aligned} \tag{9}$$

$$\begin{aligned} O_t &= E(T_t, K), \quad \text{where } T_t = N||t \\ P_t &= O_t \oplus C_t \end{aligned} \tag{10}$$

Counter mode has several advantages. Firstly, its ability to encrypt or decrypt different blocks simultaneously. This is because the encryption and decryption process of a block does not depend on any input from previous encryption or decryption stages [16]. Secondly, in CTR mode the output of the encryption or decryption process can be pre-computed even before the receiver receives the plain-text or cipher-text blocks [16]. Thirdly, in CTR mode a random plain-text or cipher-text block can be encrypted or decrypted without having to process prior blocks first [16]. Finally, unlike in ECB and CBC modes, CTR mode does not require a separate decryption algorithm to decrypt. Like OFB and CFB modes, the encryption algorithm itself can be used for decryption which would in turn lead to similar decryption and encryption processing times.

B. ChaCha

In 2005, ChaCha20 was introduced by Bernstein [6] as a candidate for the eSTREAM project. ChaCha20 is a stream cipher that is based on the Salsa cipher, which according to Bernstein has been consistently faster than AES [6]. ChaCha8, ChaCha12 and ChaCha20 have 8, 12 and 20 rounds, and each of them are based on the Salsa8, Salsa12 and Salsa20 respectively. The changes incorporated into ChaCha (compared to Salsa) can be attributed to its improved diffusion per round, which in turn contributes to its stronger resistance to cryptanalysis compared to Salsa, whilst still preserving the time it takes per round [6].

The 256-bit stream cipher's function accepts a 256-bit key, a 64-bit block counter and a 64-bit nonce as inputs. Each round of the cipher consists of 16 XORs, 16 additions and 16 constant-distance 32-bit word rotations. The cipher's round function is split into two functions that alternate between even rounds for column-round functions and odd rounds for row-round functions. The row-round function rotates the rows of the state matrix right and then rotates the columns upwards [18]. The number of rotations performed would depend on the column and row position. The consolidation of row and column round functions is called the double round function [18].

Depending on what is being prioritised, maximum security, maximum speed or a balance between the security and speed, one can choose between ChaCha20, ChaCha8 and ChaCha12 accordingly [6]. Some of ChaCha's most notable applications include OpenSSL and NSS. Google adopted ChaCha20 in March 2013 to enable symmetric encryption in OpenSSL and NSS [19]. Given its significant applications, ChaCha is one of the ciphers chosen for analysis in this study.

C. ACORN

Acorn is an authenticated cipher, which means its single algorithmic construct provides three primary cryptographic services - integrity, confidentiality and authentication [7]. Though these services can be collectively provided through schemes that combine encryption algorithms (stream or block ciphers) with message authentication schemes (keyed hash functions), having a dedicated authenticated encryption design usually has the advantage of improved efficiency and performance, particularly in devices with limited resources such as IoT devices [7]. Acorn has this advantage and has a fast software implementation, as 32 of its steps can be simultaneously computed [7], [20].

The Acorn-128 authenticated cipher requires a 128-bit key and a 128-bit IV. It takes up to 1792 steps for its initialisation process. The plain-text and associated data lengths in this authenticated cipher should be a maximum of 264 bits and the authentication tag should be a maximum of 128 bits. Associated data refers to unencrypted data that is authenticated [7].

Acorn involves the concatenation of six linear-feedback shift registers and has three primary functions. The first function generates a key stream bit from the state. The second function generates the overall feedback bit. The final function updates the state. Finally, the cipher generates a tag, after all the plain-text bits are processed. The decryption process for the cipher is similar [7], [20].

In 2013, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was started with the intent of providing a platform for developing authenticated ciphers with improved capabilities compared to the standard authenticated cipher AES-GCM. One of the use cases specified by the CAESAR committee during the evaluation of the round 3 candidates was lightweight applications. This meant the authenticated ciphers had to exhibit improved performance on resource-constrained devices while being able to exhibit resistance to side channel attacks. This kind of attack is often used to target cryptographic implementations on resource-constrained IoT devices deployed in remote locations with little to no physical protection [21]. Thus, in addition to being efficient, authenticated ciphers are also expected to be resistant to such attacks.

In order to incorporate variety in the ciphers tested, the decision was made to include Acorn, an authenticated cipher that uses stream encryption. Furthermore, with Acorn being one of the finalists of the previously discussed prestigious CAESAR competition 2018, it was deemed worthy and relevant for comparison with the other more established algorithms tested in this study, such as AES. Finally, Acorn is a relatively new authenticated cipher, and thus there was not any prominent applications for it at the time of this study.

D. PREVIOUS WORK

A comparison of the most relevant previous work, highlighting the innovation of this work, can be seen in Table 3. Most existing research on power and energy consumption measurements of low-resource devices are based on Wireless Sensor Network (WSN) nodes, which are low-powered devices that can communicate wirelessly [22]. One such work includes Dezfouli *et al.* [23] who proposed Energy Measurement Platform for Wireless IoT (EMPIOT), a power measurement platform for IoT devices. Using EMPIOT, they measured the energy and power consumption of five IoT devices with 4 varying types of workloads. EMPIOT consists of two primary components - a base board that runs a data collection and controlling software, and a shield board with an energy monitoring chip (INA219) that handles both voltage and current measurements. However, the power measuring platform has a limitation, where it is incapable of capturing low current transitions between 1μA, 10μA and 100μA low powered states.

Other studies employ specialised equipment and software to enable power and energy consumption measurements of low-powered devices. Huth *et al.* [24] in their research, measure and compare the energy consumption of two key agreements scheme implementations, Elliptic Curve Diffie-Hellman (ECDH) and Channel Based Key Agreement (CBKA), on a 32-bit ARM Cortex M3-based IoT platform. To measure the energy consumption, they make use of the EFM32GG-STK3700 Giant Gecko, a starter kit from Silicon Labs.

Zhang *et al.* [25], in their research, test for memory requirements, execution times and energy consumption of the MICAz sensor node, whilst considering only different implementations of AES. To measure the power consumption and execution time, they make use of the Agilent 14565B and 66321D equipment. To measure the RAM and ROM usage of the AES implementations, they use functions provided by the operating system. Pereira *et al.* [26], in their research, test the performance of symmetric ciphers, HMACs (hash-based message authentication code), authenticated encryption with associated data (AEAD) and hashing algorithms on the Intel Edison - a low-powered 32-bit IoT platform, with a 32-bit Intel Quark microcontroller. They test for performance in terms of execution times and energy consumption. To do this, they extract the energy consumption and run time values from the output file of the LabView software setup and obtain current consumption values using the Agilent 33401A digital multimeter. The multimeter communicates with the LabView software on the computer. These studies employ specialised equipment and software to measure energy consumption and execution times. In the method adopted by this research, commonly available non-specialised equipment is used, such as an oscilloscope to enable measurements of power and energy consumption. By comparison, the methods used to gather the performance data in this research are more accessible to those that do not have such a specialised test bed available.

There are other works that make use of an oscilloscope to enable measurements of execution time, current flow, power consumption and energy cost. Tung *et al.* [27] in their research measure only the current consumption of the Intel Edison module in three different operation modes - active, idle and sleep. To do this, they make use of an AC/DC current measurement system to measure the current waveforms of the different operation states. They use an oscilloscope to measure the up time of the Intel Edison module in sniff/sleep mode. In Ledwaba *et al.* [28] the performance of four ARM architecture chips are considered against the algorithms AES-128 in CTR mode, SHA-256 and Elliptical Curve Digital Signature Algorithm (ECDSA). This study has expanded on Ledwaba *et al.* [28] by introducing more algorithms and a wider array of microcontroller devices representative of different architectures.

Guimaraes *et al.* [29] used an oscilloscope to obtain time intervals. In their research, they calculate the energy consumption of various encryption algorithms on the MICA2 wireless sensor, which uses an ATmega128L microcontroller. They attributed the main factor for difference in energy consumption to be processing times. Thus, to measure the time intervals, they make use of an oscilloscope connected to a pin on the ATmega128. Other studies [32]–[34], use the oscilloscope to measure energy consumption of two commercial WSN nodes implementing cryptographic algorithms. They make use of the PicoScope 3206 oscilloscope and 2 resistors of 0.1Ω and 10Ω. To eliminate noise, they make use of an instrumentation amplifier. However, of the four cryptographic algorithms they tested, SHA-1, RC5, DES-CBC and AES, two are no longer in use, that is SHA-1 and DES and as such their relevance today could be questioned.

Of the research works reviewed, some [30], [31] use a similar test bench to this research. The current sensing resistors used in the research by Lee *et al.* [30] and Panait and Dragomir [31], differ from the one employed in this research being (1Ω). Lee *et al.* [30] and Panait and Dragomir [31] make use of 10.1Ω and 4.99Ω resistors respectively. In addition to this, Panait and Dragomir [31] test the energy consumption of only AES implementations in different modes of operations. This work has a broader scope, as it measures

TABLE 3. Summary of the most relevant previous work to highlight the differences and innovation of this study.

Previous Work	Brief Summary of Previous Work	Innovation of this Work by Comparison
Zhang et al. [25]	Zhang et al. conduct an analysis of three implementations of AES running on MICAz. The work examines energy consumption, memory usage and execution time. All tests are conducted using a battery simulator and associated software.	Zhang et al. only test the performance of AES in a single mode. In contrast, this study provides a structured framework and method for measuring the resource cost of encryption and decryption on resource-constrained devices. It examines AES operating in 5 modes, along with two emerging encryption algorithms – ChaCha and Acorn. The equipment used by Zhang et al. is more specialised and subsequently more expensive than the implementation this study proposed. This work tests the performance of newer microcontrollers with a more cost-effective test bench, and thus is more accessible to others, particularly in developing nations.
Pereira et al. [26]	Pereira et al. conduct performance analysis of two platforms – The Intel Edison and the TelosB. The work measures the energy use and execution time of a variety of symmetric ciphers, hashing, HMAC, AEAD. All testing implementations have an underlying OS running on the microcontroller.	This work is more focused on encryption algorithms and as such, tests and analyses a greater number of ciphers. Pereira et al. do not consider the energy usage of the key setup phase, in contrast to this work which measures the set key and set IV operations. The method adopted by Pereira et al. requires specialised software to implement in contrast to this work. All their devices are running an operating system. This study tests on the bare metal environment to produce more accurate measurements without the overhead of additional processes running in the background.
Ledwaba et al. [28]	Ledwaba et al. study the performance of AES128-CTR, SHA-256 and Elliptical Curve Digital Signature Algorithm (ECDSA) on a variety of ARM Cortex-M devices. The study measures execution time, power consumption, RAM and flash memory usage.	The test bed adopted by Ledwaba et al. incorporates a different design to that used in this study and utilises 2 channels of the oscilloscope rather than the 3 used in this study. Ledwaba et al. only consider the ARM architecture in contrast to this work which considers ARM, AVR and Tensilica. This study takes a more granular approach where possible in the algorithm analysis by measuring the set IV operation separately from the set key operation wherever possible. The scope of architectures and algorithms employed in this study is greater and thus of interest to a broader audience.
Guimaraes et al. [29]	Guimaraes et al. conduct performance assessments of various ciphers executing operations on a single device - the MICA2. The execution time is measured, and the energy consumed is calculated using the time taken for completion of the operation and the devices rated current and voltage data from the manufacturer.	In this study, using the proposed framework and practical test bench, actual power consumption was measured rather than calculated, offering a greater level of accuracy and precision. The measured algorithms were more up to date and contained more fine grain measurements by capturing the performance of key setup, setting of the IV where possible, encryption and decryption operations.
Lee et al. [30]	Lee et al. evaluate the power consumption of various encryption algorithms running on two devices - the TelosB and MICAz.	In this work, up to date devices and algorithms are used to provide a more current and meaningful analysis for real-world applications. This work also details a framework that is device and algorithm agnostic, that can be used by other researchers and IoT network designers to gain insight into other combinations of devices and ciphers in future works. In this research a much smaller resistance is used for the current sensing resistor to ensure less wasted power, striving for a higher precision result.
Panait et al. [31]	Panait et al. measure a device based on the ATmega128RFA1 microcontroller. They conduct measurements on current and execution time of AES in ECB, CBC, CFB and CTR modes.	This study covers a wider variety of devices and architectures as well as a wider variety of algorithms with the addition of AES-OFB, ChaCha and Acorn. This study measures encryption, decryption, key setup and setting of the IV, thereby offering greater granularity in the analysis in contrast to Panait et al. which consider encryption and decryption only. Panait et al. are more focused on the difference between software implementations of AES vs. hardware accelerated implementations which is not in the scope of this research. In this research a smaller resistance for the current sensing resistor is used to ensure less wasted power, to obtain a higher precision more accurate result.

power consumption, time cost, energy cost, RAM, and flash memory usage of symmetric encryption algorithms, namely AES in ECB, CBC, CFB, OFB and CTR modes of operation, ChaCha8, ChaCha12, ChaCha20 and Acorn. This study examines more up to date algorithms and devices when compared to Lee *et al.* [30]. The method in this work has also been adopted into a framework for other researchers, IoT network designers and others to follow and implement for their own testing purposes. This contrasts with Lee *et al.* [30] where the methodology is not as well defined and reproducible.

Wander *et al.* [35] presents a comparison of the energy consumption of two public-key algorithms and of implementations of mutual authentication and key exchanges between two nodes. They do this by estimating the energy consumption based on the current drawn from the power supply. In contrast, this study examines more relevant algorithms using a more sophisticated and accurate test bench to measure the actual energy usage when compared to Wander *et al.* [35].

In Tsai *et al.* [36] the authors propose an AES-128 encryption scheme for LoRaWAN networks that they assert to be more energy efficient than the current AES-128 implementation that is built into the protocol. This study examines the energy usage over a period of one day in a comparative manner. The study is limited to only AES-128 and the energy consumption is based on the ARM architecture only. This study has built upon the work of Tsai *et al.* [36] to create a more broadly applicable performance analysis method and framework that is protocol agnostic.

Other research [37]–[39], present mathematical models and energy maps created using statistical models to calculate energy consumption of cryptographic algorithm implementations in low-powered devices. This research takes a more practical based approach and does not use models and simulations. It is the intent that the research presented provides a more real-world performance analysis of both the devices and the ciphers tested. Similarly, in Morin *et al.* [40] theoretical modelling and analysis is conducted to determine the energy usage of communication technologies such as LoRa, Sigfox, and others in various transmission states. While the work does examine IoT power consumption, it does not focus on performance of different devices, nor does it focus on security. By contrast, in this study, devices in transmission states are not considered and results are based on experimentation rather than theoretical modelling.

III. EXPERIMENTAL ENVIRONMENT SETUP AND METHOD

This section details the required resources to reproduce the experiment results as well as the methodology adopted for the experiments. It first covers the microcontrollers that were tested and their specific performance specifications. The required resources that are needed are then detailed. A performance analysis framework is then presented, followed by the method that was adopted for the experiments, which is based upon the proposed performance analysis framework.

A. MICROCONTROLLERS

Three different microcontroller devices were selected to be used for the experiments. In choosing which devices to test, it was important that different architectures were selected for comparison as well as different bit widths. These were the ESP8266 ESP12–F WiFi Witty Cloud Development Board, STM32F103C8T6 and ATmega328P.

The Arduino range of microcontrollers is extensive. The ATmega328 was chosen over the other devices in the Arduino family such as the Arduino Uno or Mega, due to it being cheaper. The ATmega328 is also suitable to use in IoT networks and WSN due to its small footprint and lack of unnecessary chips and devices. The STM32F103C8T6 was chosen specifically as compared to other devices in the STM32 range, it is minimalist and does not have unnecessary extras included. The ESP8266 device was chosen as it is widely used in industry IoT implementations. In the case of the ESP8266 it has extra devices such as a photo resistor and red green blue (RGB) light emitting diode (LED). These extra components were removed to minimise wasted energy usage. Current sensing resistors and capacitors have been soldered onto the devices for the experiments. All three devices that were selected for testing can be seen in Fig. 1. These devices were selected for testing as they are popular devices used in IoT applications. All selected devices are cost effective, widely available and were accessible at the time of the experiments. For the key technical specifications of each device, refer to Table 4.

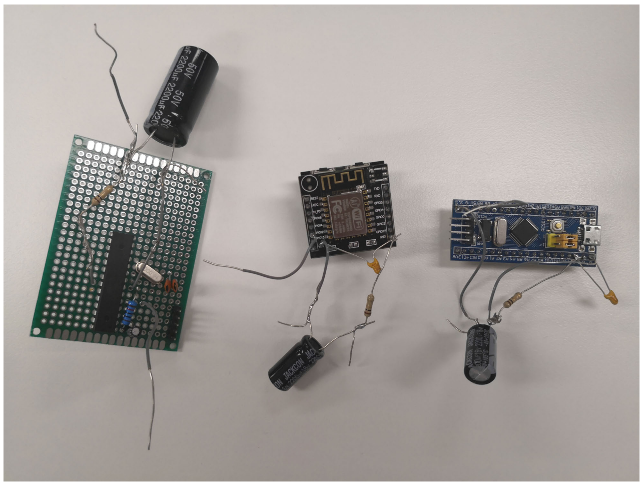


FIGURE 1. ATmega328, ESP8266 and STM32F103C8T6 Microcontroller devices from left to right. Capacitors and resistors have been soldered to the devices.

TABLE 4. Microcontroller technical specifications.

Specification	**ATmega328 [8]**	**STM32 [9], [10]**	**ESP8266 [11]–[13]**
Model #	ATmega328P	STM32F103C8T6	ESP12–F
CPU	AVR	ARM Cortex M3	Tensilica L106
Bit Width	8–bit	32–bit	32–bit
Clock Speed	16 MHz	72 MHz	80/160 MHz Modes
Clock Speed Used	16 MHz	72 MHz	80 MHz
RAM	2 KB SRAM	20 KB SRAM	160 KB SRAM
ROM	1 KB EEPROM	Not Specified	64 KB ROM
Flash	32 KB	64 KB	512 KB
Input Voltage	2.7 – 5.5 V	2.0 – 3.6 V	3.0 – 3.6 V
Input Voltage Used	5 V	3.3 V	3.9 V

B. REQUIRED RESOURCES

The resources required to conduct the experiments are detailed below:

1) As discussed previously, the three microcontroller devices that were tested were the ESP8266, the STM32F103C8T6 and the ATmega328. External universal serial bus (USB) programmers were used even when on-board USB ports were available. This decision was made to avoid uploading unnecessary boot-loader

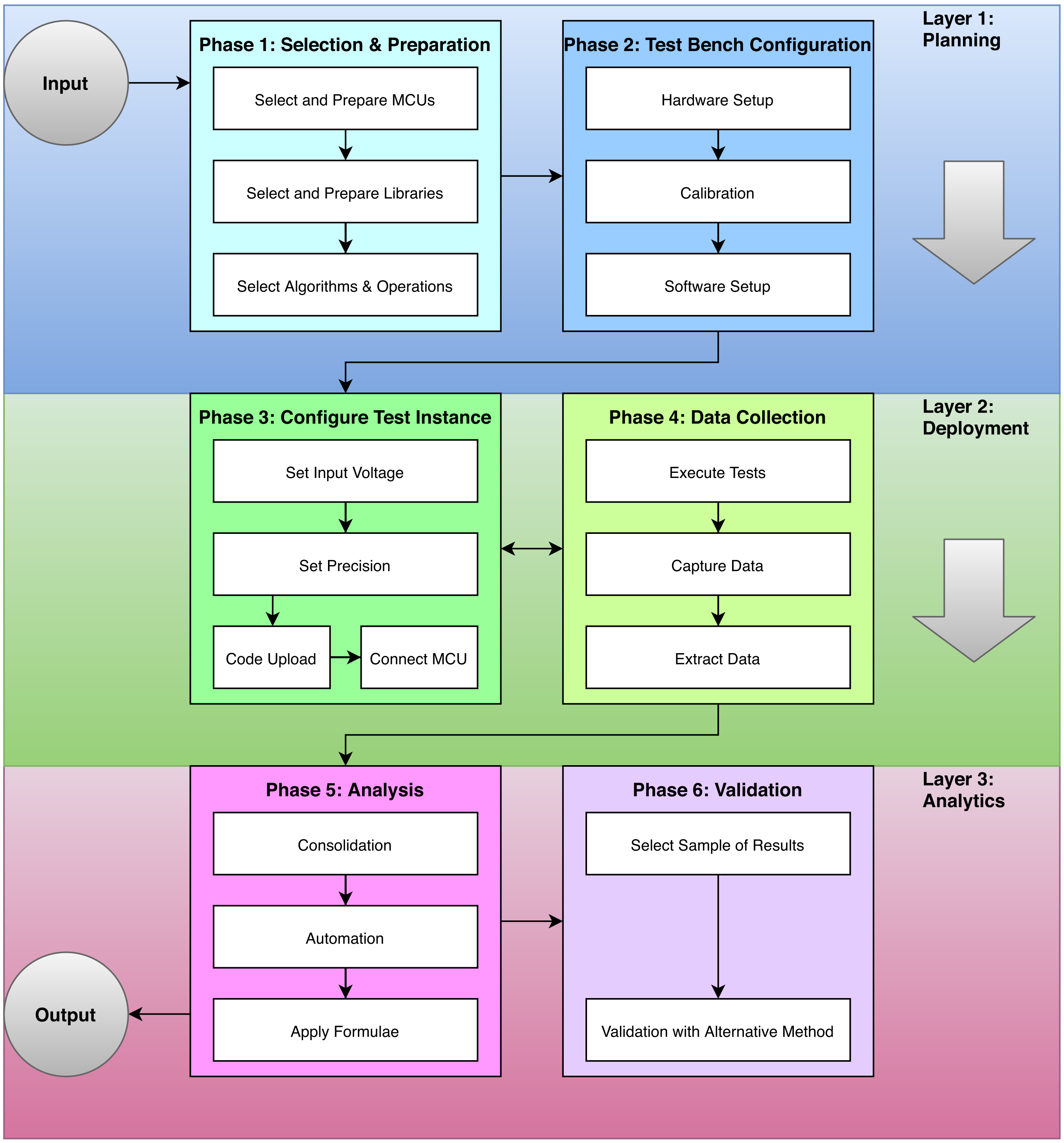


FIGURE 2. **Proposed three-layer, six-phase performance analysis framework.**

code to the devices that would consume extra space in the flash memory.

2) A Personal Computer (PC) was required for preparation and uploading of the code to the devices. Several installed resources were required. The Arduino integrated development environment (IDE) was used to produce the code for the experiments and to upload the code to the devices. The ATmega328 already had the appropriate libraries to support it included with the IDE. The STM32 libraries were enabled through the board manager in the IDE. The ESP8266 libraries had to be obtained from an external source [41].

The Arduino Cryptography Library [42] was installed to provide all the encryption algorithms to be tested.

3) An external power source to power the devices. For this study an Agilent U8031A Direct Current (DC) power supply was used. This device was selected to ensure a more stable source of power was supplied in contrast to a battery. This power supply is a linear supply, and as such there is less noise present when compared to a switching power supply.
4) A resistor to be used for current sensing. A commonly available 1 Ohm (Ω) resistor was used. The resistance was measured using a Digital Multimeter to ensure the advertised resistance value was accurate within an acceptable tolerance. The resistor was soldered onto each microcontroller device to minimise any additional sources of noise being added to the circuit.
5) Capacitors were used to stabilise the voltage output from the power supply.
6) A Digital Oscilloscope was required to obtain the experiment data for analysis. For the experiments detailed in this study, an Agilent DSO-X 2024A Digital Oscilloscope was used. This oscilloscope has four channels, three of which were utilised for the experiments. The reason an Oscilloscope was used was so that all data relating to the experiments could be captured at the one time using the multiple channels. It was also used as it is a commonly available piece of equipment that is not specialised for IoT device testing.

C. PERFORMANCE ANALYSIS FRAMEWORK

The Performance Analysis Framework proposed by this study can be seen in Fig. 2. The framework is a three-layer top-down design incorporating six distinct phases designed to provide clarity, direction and structure. The beneficiaries of this framework may include researchers, IoT network and device designers and others seeking to measure resource cost in IoT security applications.

The first layer at the top as defined in the framework is the Planning layer. This layer consists of two phases which cover the preparation and configuration of the main components that form the performance analysis experiments. The second layer moving down is the Deployment layer. In this layer there are two phases that are concerned with performing the actual experiments. The third and final layer of the framework is the Analytics layer. This layer contains two phases which are focused on consolidation, analysis, and validation of the experiment results.

1) PHASE 1: SELECTION AND PREPARATION

The Selection & Preparation phase takes two inputs - the microcontroller units (MCUs) required for performance testing and the software libraries to support this testing. Libraries that may be required in this phase include device specific and cryptographic. The MCUs may require preparation that include removing extraneous components that consume unnecessary power, such as voltage regulators, LEDs and sensors. Components that may need to be soldered onto the MCUs at this stage include capacitors and current sensing resistors.

2) PHASE 2: TEST BENCH CONFIGURATION

The Test Bench Configuration phase requires the configuration of both the hardware and software components of the measurement experiments. Measurement equipment such as an oscilloscope, digital multimeter, power supply and appropriate cables need to be setup. It is important for the accuracy and validity of the tests that any measurement equipment is calibrated according to the manufacturer's instructions. In this phase, the code that will execute the tests on the MCUs needs to be developed.

3) PHASE 3: CONFIGURE TEST INSTANCE

The Configure Test Instance phase is recurring and needs to be repeated for each combination of device, algorithm and operation that will be tested. The input voltage needs to be set appropriately for each MCU. Appropriate precision needs to be selected on the measurement equipment. Necessary modifications to the testing code for each instance as well as uploading the code to the MCU needs to occur in this phase. Finally, the device needs to be connected to the testing equipment ready for the Data Collection phase.

4) PHASE 4: DATA COLLECTION

In the Data Collection phase, the tests can be executed with the MCU connected to the measurement equipment. Once the data is captured, it must be extracted from the measurement equipment. This extraction process can vary depending on the equipment used and the functionality it provides. Once the data collection phase has concluded, Phase 3 can commence for the next combination of MCU, encryption algorithm and operation to be tested. When all combinations of algorithm, operation and MCU have been completed, phase 5 can then commence.

5) PHASE 5: ANALYSIS

In the Analysis phase, the data must be consolidated so it can be processed and analysed efficiently and accurately. Automation should be employed using scripting to remove extraneous and unnecessary data. An example of extraneous data would be the resource usage between operation cycles. Once the final data set is available, calculations can be made using appropriate formulae. By the conclusion of the analysis phase, the output produced will be the final results for the desired performance metrics.

6) PHASE 6: VALIDATION

In the Validation phase, a selection of the results should be retested with an alternative method where possible. This is done to demonstrate correctness of the testing methodology adopted. A broad selection of results across different combinations should be measured. This phase does not require all results to be retested, just a small sample to demonstrate correctness. Once correctness has been proven, this phase may no longer be necessary in all circumstances.

7) ADVANTAGES AND DISADVANTAGES OF THE PROPOSED FRAMEWORK

The proposed framework has both advantages and disadvantages that should be highlighted. The framework provides direction and guidance for researchers, designers and others interested in measuring performance and security of IoT devices. The framework itself is test bench, device and algorithm agnostic and as such has broad application and flexibility. The disadvantage of the approach taken by this framework is the scalability of layer 2. If there is a very large combination of devices, algorithms, and operations to test, the repetition between phase 3 and phase 4 could become tedious.

D. METHODOLOGY

This section discusses the methods used to conduct the power consumption, energy cost and time cost measurements of the various ciphers and operations on each of the three chosen devices. The methods used to conduct the experiments covered in this research are sound and use some techniques similar to previous work [30], [31]. A sample of the results were validated using a Keithley DMM6500 Digital Multimeter where consistent results in current were observed.

For the tests of the various AES modes and the Acorn algorithm, 16 bytes of data were hard-coded into each of the algorithms to be used to test the encryption, decryption and set key operations. In the case of the ChaCha algorithm, whilst the same basic operations were tested, the hard-coded data was 64 bytes in size. When the results for ChaCha were presented, the results were divided by 4 so that the results were comparable to the other ciphers.

For each operation on each device, code was uploaded to the device from the PC using the Arduino IDE. This code has the respective operation running in a loop. The device was then disconnected from the PC, and then connected to the oscilloscope and the power supply. To avoid introducing any further noise into the results, the current sensing resistors were soldered onto the devices and the use of any breadboards were avoided. The oscilloscope was used to capture the current and voltage of the load precisely as well as the time of each cycle. A circuit diagram was designed as shown in Fig. 3. The load and 1Ω resistor are in series. The three channels (V_1, V_2 and V_3) from the oscilloscope are across the resistor (V_1), the load and the resistor (V_2), as well as a designated pin of the load (V_3) which is used to determine the start and stop times of each operation cycle.

Between each test iteration, the pin located at (V_3) was set to high at the beginning of the operation and then returned to low at the conclusion of the operation. The time in-between these two events forms one cycle of the operation. Each cycle was clearly visible on the oscilloscope. For example, the gold line (V_3) in the lower part of Fig. 4 shows the end of one cycle and the commencement of the next cycle. The bidirectional

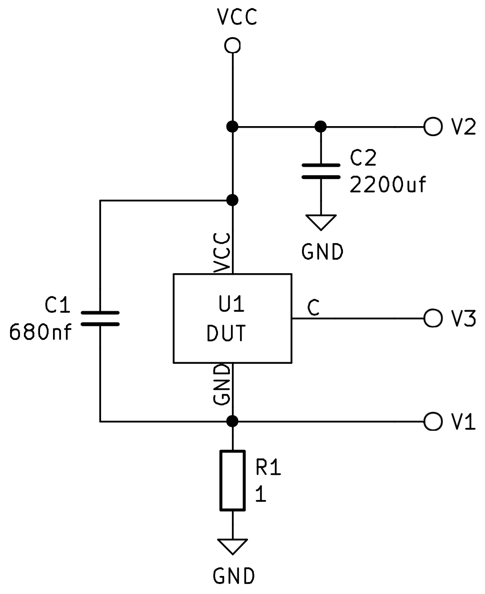


FIGURE 3. Diagram of circuit used for the experiments.

arrows in upper part of the image highlights the boundaries of three of the repeated operation cycles. In this study, the portions of V_1 in blue and V_2 in pink that occurred within the boundaries of the each of the operation cycles are considered for the purposes of the time cost, power consumption and energy cost calculations.

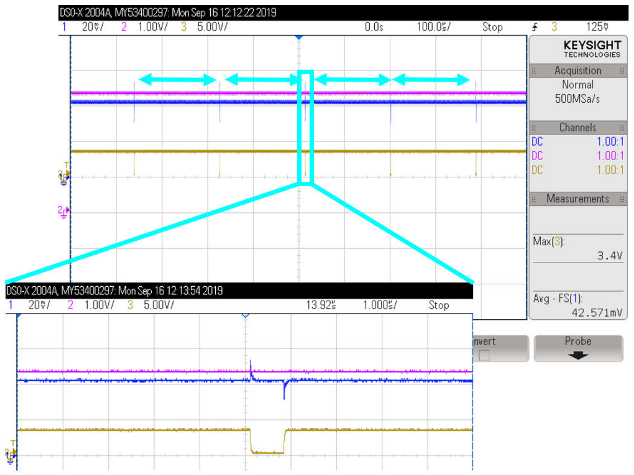

DSO-X 2004A, MY53400297: Mon Sep 16 12:12:22 2019
1 20‰/ 2 1.00V/ 3 5.00V/ 0.0s 100.0‰/ Stop 3 125‰
KEYSIGHT TECHNOLOGIES
Acquisition
Normal
500MSa/s
Channels
DC 1.00:1
DC 1.00:1
DC 1.00:1
Measurements
Max(3):
3.4V
Avg - FS(1):
42.571mV
nvert
Probe
DSO-X 2004A, MY53400297: Mon Sep 16 12:13:54 2019
1 20‰/ 2 1.00V/ 3 5.00V/ 13.92‰ 1.000‰/ Stop

FIGURE 4. Screenshot of the oscilloscope measurement.

To extract the energy cost (in Joules) of each of the individual operations, the power consumption (in watts) first needs to be calculated. This can be achieved using Formula (11) which was derived from the power law.

$$P_{Load} = \frac{V_1}{R} \times (V_2 - V_1) \tag{11}$$

where $\frac{V_1}{R}$ is the current of the circuit and $V_2 - V_1$ is the voltage of the load. Once the power consumption is obtained using Formula (11), the energy cost (Joules) of each iteration can then be calculated using Formula (12) which has been derived from Formula (11).

$$E_{Load} = \sum_{i=1}^{n} P_i \times t_i \tag{12}$$

where P_i is obtained from Formula (11), and t_1 is the time interval between the samples taken. In order to obtain the average energy cost of encryption methods on various loads, the sum of the energy cost was divided by the number of iterations (m). This is shown in Formula (13).

$$E_{Load}^{avg} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} P_i \times t_i}{m} \tag{13}$$

Once the data was analysed, it was noted there was a spike in power usage at the beginning of each cycle. This was due to pin being used to detect the cycle boundaries being set to high. This can be seen in Fig. 5. Some noise is also present and can be seen in the figure, as well as the regular oscillation of the CPU clock. Bypass capacitors were used to limit the effect of the spike and reduce overall noise. Bypass capacitors are commonly used to improve the stability of CPUs.

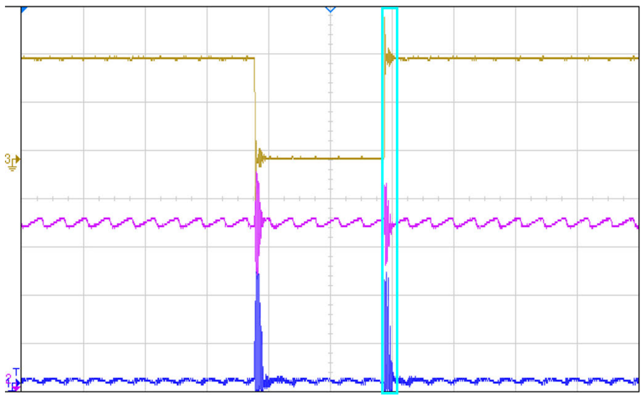
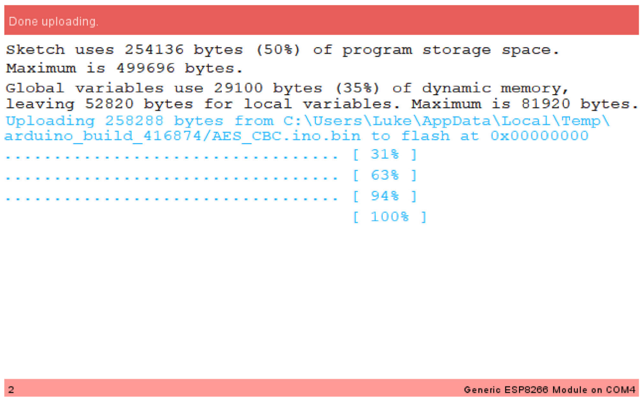


FIGURE 5. **A Spike in measurements. The CPU clock frequency is also clearly visible.**

E. MEMORY ANALYSIS METHOD

On resource-constrained IoT devices, memory utilisation is an important factor to consider. Misuse or attempted overuse of available resources can result in unexpected and undesirable behaviour. Determining the space used in the flash memory by the code is a relatively simple process. This information can be obtained from the compiler when the code is uploaded to the device. An example of this output can be seen in Fig. 6.

Determining the amount of RAM used during code execution is a more challenging task. The approach this study has taken is to obtain the minimum and maximum free memory available during the code execution. The first measurement is taken prior to the encryption algorithm running. This initial value will represent the largest amount of free memory. A measurement is taken after each instruction throughout the encryption algorithm execution. The MemoryFree library



```
Done uploading.
Sketch uses 254136 bytes (50%) of program storage space.
Maximum is 499696 bytes.
Global variables use 29100 bytes (35%) of dynamic memory,
leaving 52820 bytes for local variables. Maximum is 81920 bytes.
Uploading 258288 bytes from C:\Users\Luke\AppData\Local\Temp\
arduino_build_416874/AES_CBC.ino.bin to flash at 0x00000000
.................................. [ 31% ]
.................................. [ 63% ]
.................................. [ 94% ]
                                   [ 100% ]

2                                          Generic ESP8266 Module on COM4
```

FIGURE 6. **Example of total flash memory usage displayed from the compiler when uploading code for the ESP8266 device in AES CBC mode.**

[43] is used to obtain the measurements. The measurements were then output to the console, and the results were analysed. The peak memory usage is calculated using Formula (14).

$$Usage_{Memory} = MaxFree_{Memory} - MinFree_{Memory} \tag{14}$$

IV. RESULTS

This section examines the results of the experiments conducted on the three microcontrollers. The results are broken up into three sections. Each tested operation of all the algorithms are presented. First the power consumption is presented measured in watts per second, followed by time cost in seconds of each operation. From these two results, the total energy cost was calculated and is presented in this section.

A. POWER CONSUMPTION

Power consumption is the amount of energy consumed defined in watts, calculated as per Formula (11). The power consumption of each operation for each of the tested devices can be seen in Table 5 and Fig. 7. The ATmega328 device exhibits the lowest power consumption across all tested algorithms and operations, with the ESP8266 device exhibiting the worst performance in this metric.

Each device demonstrated relatively consistent power consumption across the board on the tested algorithms and operations. The results show that when the STM32F103C8T6 executed both the ChaCha and Acorn algorithms, a slight increase in power consumption can be seen when compared with AES.

B. TIME COST

The time cost is the total amount of time it takes for any given operation to run from start to finish. The time cost results of each operation for each of the tested devices can be seen in Table 6 and Fig. 8. Overall, the ATmega328 exhibits the worst performance in terms of time cost across all tested algorithms and operations. The ESP8266 performs the best in most operations, exhibiting the fastest performance in 17 out of the total 27 tested operations. The STM32F103C8T6 also performed well and was the fastest in the remaining 10 operations.

TABLE 5. Power consumption results (mW).

Operation	ATmega328	STM32F103C8T6	ESP8266
AES_ECB_Setkey	42.867	144.597	235.884
AES_ECB_Encryption	42.695	142.992	235.142
AES_ECB_Decryption	42.163	139.608	237.361
AES_CBC_Set_Key	43.562	144.432	235.130
AES_CBC_Encryption	42.216	143.565	235.142
AES_CBC_Decryption	42.092	141.325	235.848
AES_CFB_Set_Key	43.409	144.875	236.971
AES_CFB_Encryption	42.648	144.089	235.532
AES_CFB_Decryption	42.344	144.520	236.836
AES_OFB_Set_Key	42.509	143.757	230.224
AES_OFB_Encryption	42.808	143.276	229.691
AES_OFB_Decryption	43.749	144.033	229.931
AES_CTR_Set_Key	43.614	144.889	230.299
AES_CTR_Encryption	41.704	143.950	232.024
AES_CTR_Decryption	42.633	143.468	230.163
ChaCha_Set_Key	40.459	137.682	231.855
ChaCha_Set_IV	41.811	135.677	231.390
ChaCha8_Encryption	42.876	146.644	230.543
ChaCha8_Decryption	43.635	147.864	229.505
ChaCha12_Encryption	43.641	149.819	229.835
ChaCha12_Decryption	43.026	149.023	232.049
Chacha20_Encryption	43.527	151.029	229.423
Chacha20_Decryption	41.997	150.175	229.963
Acorn_Set_Key	41.967	131.118	229.082
Acorn_Set_IV	41.587	154.298	232.891
Acorn_Encrypt	43.290	152.473	230.331
Acorn_Decrypt	43.161	156.851	230.027

TABLE 6. Time cost results (μS).

Operation	ATmega328	STM32F103C8T6	ESP8266
AES_ECB_Setkey	160.250	34.638	35.407
AES_ECB_Encryption	533.750	102.667	102.944
AES_ECB_Decryption	1156.250	186.100	146.833
AES_CBC_Set_Key	159.000	37.542	37.400
AES_CBC_Encryption	559.375	105.500	107.250
AES_CBC_Decryption	1180.000	188.700	150.167
AES_CFB_Set_Key	158.500	37.567	37.440
AES_CFB_Encryption	573.000	106.444	107.125
AES_CFB_Decryption	575.313	106.250	107.437
AES_OFB_Set_Key	158.500	37.450	37.420
AES_OFB_Encryption	566.250	105.222	107.056
AES_OFB_Decryption	567.500	105.375	107.250
AES_CTR_Set_Key	159.000	37.433	37.420
AES_CTR_Encryption	584.063	108.625	109.188
AES_CTR_Decryption	584.688	109.250	109.500
ChaCha_Set_Key	37.545	3.705	7.990
ChaCha_Set_IV	12.657	2.638	2.345
ChaCha8_Encryption	150.234	10.077	6.900
ChaCha8_Decryption	150.234	10.175	6.950
ChaCha12_Encryption	194.167	11.375	7.966
ChaCha12_Decryption	194.271	11.480	8.011
Chacha20_Encryption	282.500	14.000	10.168
Chacha20_Decryption	282.656	14.059	10.191
Acorn_Set_Key	11.210	1.543	1.637
Acorn_Set_IV	4780.000	166.000	119.929
Acorn_Encrypt	335.600	13.450	9.950
Acorn_Decrypt	329.800	12.900	10.125

The time cost of the AES algorithm operating in ECB, CBC, CFB, OFB and CTR modes are very similar. For example, the time costs of AES encryption in any mode on the ATmega328 is around 0.5 milliseconds. The STM32F103C8T6 is shown to perform most AES operations quicker than the other devices.

The results show that the ChaCha algorithms and the Acorn algorithms exhibit similar time costs in the encryption and decryption operations, with both algorithms performing substantially faster than AES.

C. ENERGY COST

Energy cost is the amount of energy used expressed in joules and calculated as per Formula (12). The total energy cost of each operation for each of the devices can be seen in Table 7 and Fig. 9. The ESP8266 exhibits the highest energy cost overall, with the highest energy cost in 14 operations. The ATmega328 is also expensive in terms of energy cost, using the most energy in the remaining 13 operations. The STM32F103C8T6 is the most energy efficient device in all tested algorithms and operations.

TABLE 7. Energy cost results (μJ).

Operation	ATmega328	STM32F103C8T6	ESP8266
AES_ECB_Setkey	6.912	5.041	8.470
AES_ECB_Encryption	22.895	14.752	24.324
AES_ECB_Decryption	48.857	26.051	34.971
AES_CBC_Set_Key	6.970	5.458	8.911
AES_CBC_Encryption	23.720	15.218	25.337
AES_CBC_Decryption	49.774	26.739	35.534
AES_CFB_Set_Key	6.924	5.478	8.991
AES_CFB_Encryption	24.479	15.410	25.349
AES_CFB_Decryption	24.467	15.428	25.564
AES_OFB_Set_Key	6.780	5.418	8.730
AES_OFB_Encryption	24.347	15.147	24.705
AES_OFB_Decryption	24.937	15.249	24.775
AES_CTR_Set_Key	6.978	5.461	8.733
AES_CTR_Encryption	24.462	15.709	25.450
AES_CTR_Decryption	25.034	15.746	25.318
ChaCha_Set_Key	1.540	0.518	1.858
ChaCha_Set_IV	0.540	0.363	0.557
ChaCha8_Encryption	6.468	1.486	1.597
ChaCha8_Decryption	6.583	1.514	1.601
ChaCha12_Encryption	8.501	1.714	1.846
ChaCha12_Decryption	8.386	1.719	1.874
Chacha20_Encryption	12.324	2.124	2.346
Chacha20_Decryption	11.897	2.121	2.357
Acorn_Set_Key	0.480	0.206	0.384
Acorn_Set_IV	199.201	25.691	28.047
Acorn_Encrypt	14.572	2.055	2.308
Acorn_Decrypt	14.278	2.029	2.343

D. MEMORY UTILISATION

This section discusses the flash memory for all three of the tested devices, and the RAM utilisation of the algorithms on the ATmega328 and STM32F103C8T6. At the time of the RAM testing, a suitable compatible library to conduct accurate testing on the ESP8266 device was not able to be sourced.

Memory utilisation is an important metric for resource-constrained devices. Understanding the memory usage of each algorithm can help developers to choose the most suitable algorithm for a given application and to also avoid any potential programming issues.

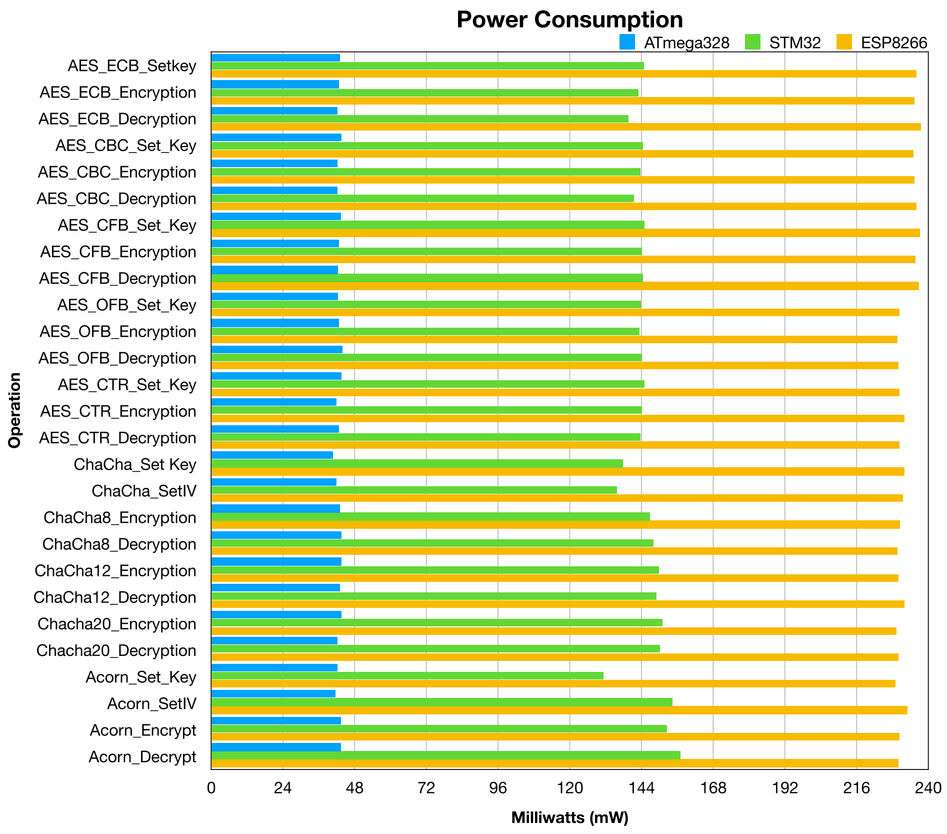


FIGURE 7. Power consumption comparison of results. Refer to Table 5 for all values.

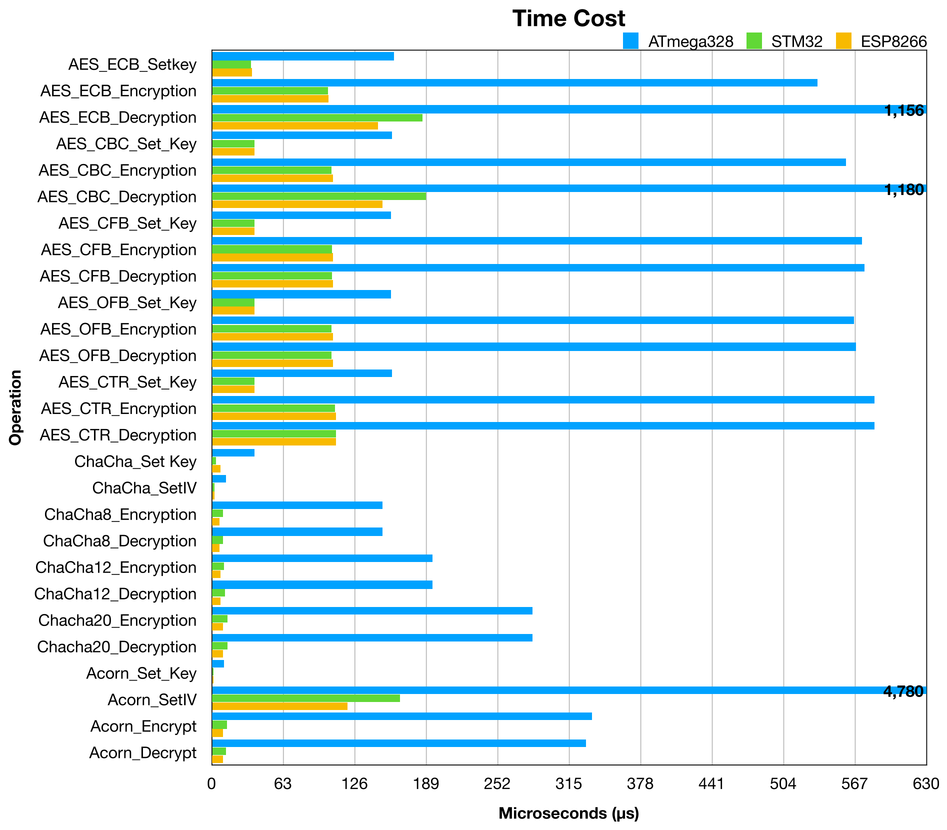


FIGURE 8. Time cost comparison of results. Refer to Table 6 for all values. Values that continue beyond the graph boundary are marked.

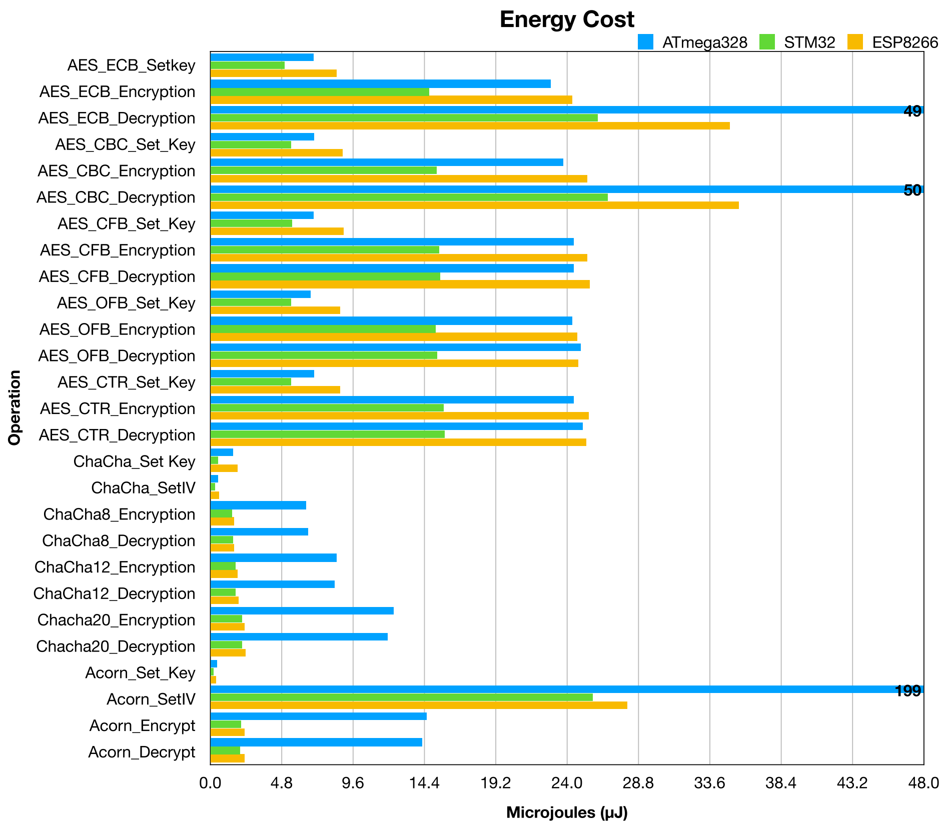


FIGURE 9. Energy cost comparison of results. Refer to Table 7 for all values. Values that continue beyond the graph boundary are marked.

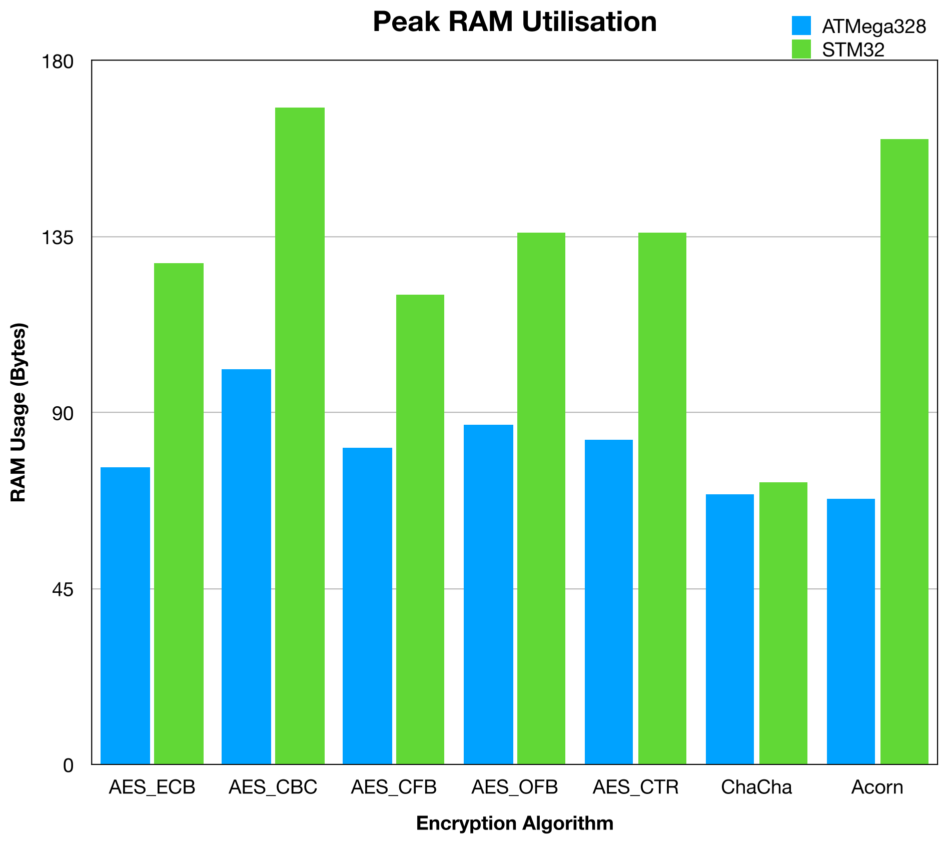


FIGURE 10. Peak RAM utilisation of each cipher.

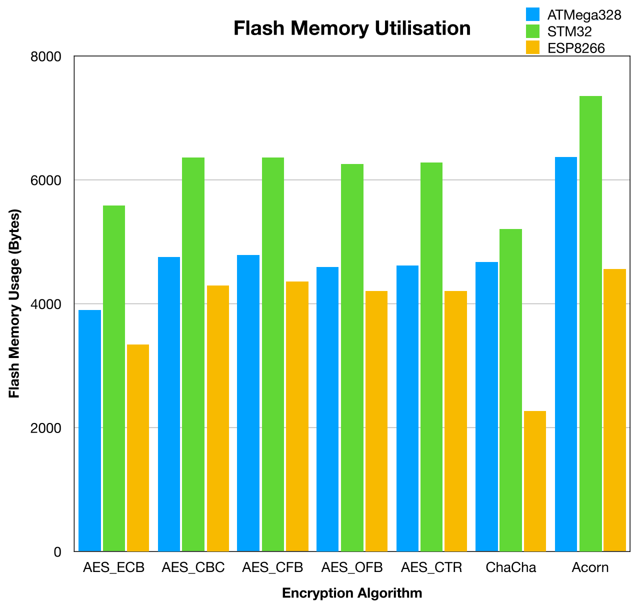


FIGURE 11. Flash memory utilisation of each cipher.

TABLE 8. Peak RAM utilisation (Bytes).

Encryption Algorithm	ATmega328	STM32F103C8T6
AES_ECB	76	128
AES_CBC	101	168
AES_CFB	81	120
AES_OFB	87	136
AES_CTR	83	136
ChaCha	69	72
Acorn	68	160

TABLE 9. Flash memory utilisation (Bytes).

Encryption Algorithm	ATmega328	STM32F103C8T6	ESP8266
AES_ECB	3892	5576	3340
AES_CBC	4760	6352	4288
AES_CFB	4776	6352	4356
AES_OFB	4594	6256	4208
AES_CTR	4622	6280	4204
ChaCha	4668	5200	2260
Acorn	6370	7360	4560

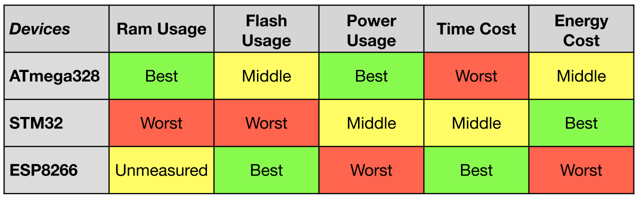
Fig. 10 shows that on different devices (8-bit and 32-bit), the same algorithm can take varying amounts of memory. For example, the peak RAM usage for AES on the ATmega328 is around 90 Bytes, but it takes from 120 to 170 Bytes on the STM32F103C8T6. The peak RAM usage for ChaCha on the ATmega328 is very similar to the usage on the STM32F103C8T6. Refer to Table 8 for the full table of results.

Fig. 11 shows the STM32F103C8T6 has the highest utilisation of flash memory for all the tested algorithms, while the ESP8266 consistently displays the lowest usage. Refer to Table 9 for the full table of results.

V. DISCUSSION

Although the ATmega328 consumes less power in watts (refer to Fig. 7), it takes more time (see Fig. 8) and thus a greater amount of energy is consumed (see Fig. 9) to execute the algorithms on this device. The most appropriate use case for the ATmega328 would be if there is a limitation on the input power available.

The STM32F103C8T6 has a slightly lower efficiency in terms of time cost when compared with the ESP8266. If the use case requires a focus on raw speed, the ESP8266 would be the most suitable device to implement although the overall difference in speed when compared with the STM32F103C8T6 is negligible across most operations. The STM32F103C8T6 offers lower power consumption and thus an overall lower energy cost than the ESP8266. The STM32F103C8T6 appears to strike a better balance in terms of power consumption and speed and as such would be the most suitable device to implement in a wider variety of use cases when compared against the other two devices. A comparison matrix was created to show the performance in terms of best, worst and average in all the key performance areas. This can be seen in Fig. 12.



Devices	Ram Usage	Flash Usage	Power Usage	Time Cost	Energy Cost
ATmega328	Best	Middle	Best	Worst	Middle
STM32	Worst	Worst	Middle	Middle	Best
ESP8266	Unmeasured	Best	Worst	Best	Worst

FIGURE 12. Key performance areas matrix.

Using the data obtained from the experiments, a comparison of the time cost and energy cost of encrypting and decrypting 1 Kilobyte (KB) of data has been prepared. This has been calculated by using the average results for each ciphers' set key, encrypt and decrypt operation for both time cost and energy cost. The results of the time cost can be seen in Table 10 and Fig. 13. The energy cost can be seen in Table 11 and Fig. 14. These comparisons were calculated using Formula (15). As demonstrated by the formula, the encryption and decryption values are added together, and then multiplied by 64. This has been done as the input data input into the algorithms was 16KB. The cost of the set key is then finally added.

$$Cost = SetKey + (Encryption + Decryption) \times 64 \quad (15)$$

The set key function of both AES and ChaCha have both a lower time and energy cost than the encryption and decryption functions. In contrast, the Acorn set IV function costs ten times more energy than its encryption and decryption functions. In scenarios that require frequent key modification, Acorn could be potentially unsuitable. If this is the case, ChaCha20 would be more suitable due to the set key and set IV operations performing the best in terms of both energy cost and time cost when compared to all other algorithms. The Acorn encryption and decryption functions are similar to ChaCha20 in terms of lower cost than the equivalent AES functions.

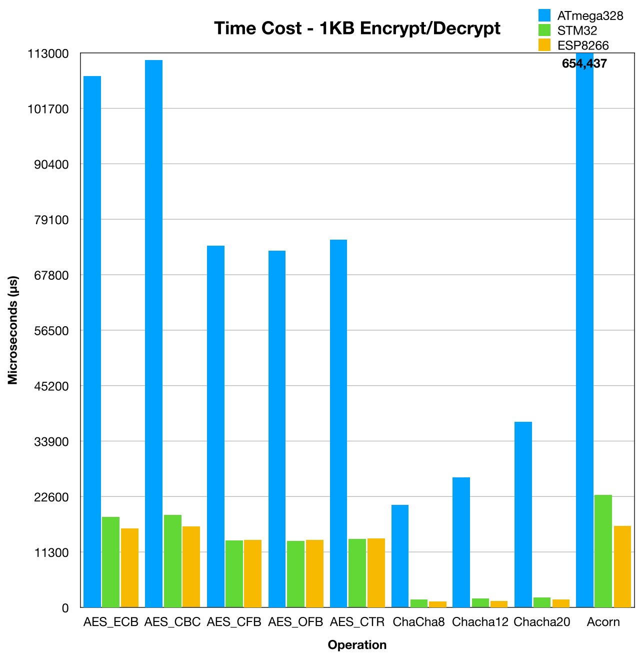


FIGURE 13. Time cost comparison of encryption and decryption for 1KB of data.

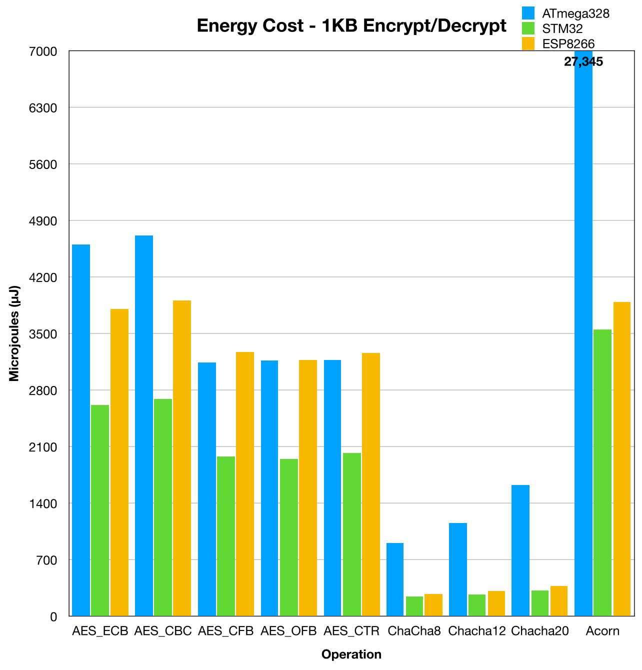


FIGURE 14. Energy cost comparison of encryption and decryption for 1KB of data.

As well as having a high energy and time cost for decryption, AES ECB mode is not recommended to be used as it is not secure, particularly in the case when dealing with a long message [44]. The AES CBC mode decryption operation shares the same higher energy and time cost as ECB mode. This may make it unsuitable for use in microcontroller devices if they are performing a large amount of decryption operations. If the decryption is being performed on a server, this may not be a concern.

In a use case involving encrypted one-way communication between a server and an IoT device, there may be situations where an algorithm that has different total energy cost between the encryption and decryption method is selected. An example of this would be AES ECB and CBC modes. The encryption operation is less expensive than the decryption operation. If it is necessary to use one of these ciphers, a suggested way to optimise energy usage on the IoT device is for down-link messages to have the server use the decryption operation to encrypt the message sent. The IoT device would then use the encryption operation to decode the message. An IoT device sending a message up-link would use the encryption operation to encrypt the message, and the server would use the decrypt operation. This has the added benefit of requiring only the one operation to be stored on the IoT device, and thus could save flash storage space.

TABLE 10. Time cost results - Encrypt/Decrypt 1KB (μS).

Cipher	ATmega328	STM32F103C8T6	ESP8266
AES_ECB	108320.250	18515.705	16021.185
AES_CBC	111479.000	18866.342	16512.067
AES_CFB	73650.500	13650.011	13769.440
AES_OFB	72718.500	13515.672	13752.976
AES_CTR	74959.000	13981.433	14033.420
ChaCha8	20887.660	1637.505	1194.528
ChaCha12	26517.660	1804.080	1330.642
Chacha20	37827.660	2137.160	1611.110
Acorn	654436.810	22935.943	16637.294

TABLE 11. Energy cost results - Encrypt/Decrypt 1KB (μJ).

Cipher	ATmega328	STM32F103C8T6	ESP8266
AES_ECB	4599.005	2616.427	3803.375
AES_CBC	4710.559	2690.687	3904.655
AES_CFB	3139.466	1979.051	3267.402
AES_OFB	3160.979	1950.818	3175.427
AES_CTR	3174.725	2018.525	3257.885
ChaCha8	905.958	238.929	277.772
ChaCha12	1151.428	266.665	311.216
Chacha20	1620.804	318.644	374.169
Acorn	27344.525	3549.934	3888.035

In real-world situations, consideration must be given to the secure transmission of data. With the limited packet sizes common to Low Power Wide Area Network (LPWAN) technologies, encrypting data could cause an unsustainable overhead when compared to the maximum transmission payload allowed. Some technologies have a very limited size available for payload. An example of this is SigFox which can only transmit a maximum payload of 16 bytes [45].

Due to library compatibility issues faced, the ESP8266 was exempted from RAM usage testing and as such the ATmega328 and STM32F103C8T6 were compared. ATmega328 has 2KB SRAM, and STM32F103C8T6 has 20KB RAM memory, which means these three algorithms take approximately 4% on the ATmega328, and less than 1% on the STM32F103C8T6, leaving an abundance of space to utilise for other uses. This may make the STM32F103C8T6 device a more appropriate choice depending on the specific use case.

Microcontrollers, like any other electrical equipment, all have different input voltage requirements as seen in Table 4. In the case of the ESP8266, throughout the testing, it was only stable when the input voltage was increased to 3.9V which is beyond its maximum voltage of 3.6V. This behaviour could be attributed to the drop in the voltage across the current sensing resistor.

VI. CONCLUSION

This research presented a framework to analyse various performance metrics of microcontroller devices that are commonly used in IoT deployments. To demonstrate this framework, a review was conducted of the performance of the AES, ChaCha and Acorn ciphers running on three different microcontroller devices, being the STM32F103C8T6, the ATmega328 and the ESP8266 Wi-Fi Witty Cloud Development Board. The devices were tested performing encryption, decryption and set key operations. Measurements were taken in the key areas of power consumption, time cost and energy cost. The results were presented in a comparative manner in each of these categories. The peak RAM usage and flash usage was also measured and presented. These results were then discussed, and observations and recommendations were made.

Overall the STM32F103C8T6 device seemed to strike the better balance between performance and speed and would be a suitable choice for many IoT deployments. The Acorn and ChaCha algorithms perform substantially faster than AES and use less energy and should also be considered for lightweight encryption uses. The extremely high cost of the Acorn set IV operation may make it unsuitable for use in many situations. Taking this into account, the ChaCha algorithm would be the best choice for most use cases.

The future work that will be conducted is to expand the testing to include a wider variety of ciphers, implementations and device combinations to form a comprehensive comparison guide that could assist designers in the selection of appropriate devices for future IoT deployments. An accurate method will be devised to measure the flash usage of the ESP8266 device. Power consumption testing using authentication tags will also be explored.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] Ericsson. *Internet of Things Forecast*. Accessed: Feb. 28, 2020. [Online]. Available: https://www.ericsson.com/en/mobility-report/internet-of-things-forecast

[3] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, and P. Siano, "IoT-based smart cities: A survey," in *Proc. IEEE 16th Int. Conf. Environ. Electr. Eng. (EEEIC)*, Jun. 2016, pp. 1–6.

[4] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl.*, Nov. 2014, pp. 230–234.

[5] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard*. Berlin, Germany: Springer, 2013.

[6] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Proc. Workshop Rec. SASC*, vol. 8, 2008, pp. 3–5.

[7] H. Wu, "ACORN: A lightweight authenticated cipher (V3)," in *Proc. Candidate CAESAR Competition*, 2016. Accessed: Aug. 16, 2019. [Online]. Available: https://competitions.cr.yp.to/round3/acornv3.pdf

[8] Atmel. *ATmega328P Datasheet*. Accessed: Sep. 9, 2019. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[9] STMicroelectronics. *STM32F103x8/STM32F103xB Datasheet—Production Data*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[10] *Arduino for STM32. STM32duino wiki: Blue Pill*. Accessed: Sep. 9, 2019. [Online]. Available: https://wiki.stm32duino.com/index.php?title=Blue_Pill

[11] Shenzhen Ai-Thinker Technology. *ESP-12F Datasheet*. Accessed: Sep. 9, 2019. [Online]. Available: https://wiki.ai-thinker.com/_media/esp8266/a014ps01.pdf

[12] ESP8266 Community Forum. *ESP8266 Wiki*. Accessed: Sep. 9, 2019. [Online]. Available: https://github.com/esp8266/esp8266-wiki/wiki

[13] Espressif Systems. *ESP8266EX Datasheet*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[14] National Institute of Standards and Technology, "Federal information processing standards publication 197," in *Proc. FIPS PUB*, 2001, pp. 46–53.

[15] M. A. Kumar and S. Karthikeyan, "Investigating the efficiency of blowfish and Rejindael (AES) algorithms," *Int. J. Comput. Netw. Inf. Secur.*, vol. 4, no. 2, p. 22, 2012.

[16] W. Stallings, "NIST block cipher modes of operation for confidentiality," *Cryptologia*, vol. 34, no. 2, pp. 163–175, Mar. 2010.

[17] *ISO/IEC 9797-1:2011 Information Technology—Security Techniques—Message Authentication Codes (MACs)—Part 1: Mechanisms Using a Block Cipher*, Standard ISO/IEC 9797-1, International Organization for Standardization, 2011.

[18] M. Goll and S. Gueron, "Vectorization on ChaCha stream cipher," in *Proc. 11th Int. Conf. Inf. Technol., New Generat.*, Apr. 2014, pp. 612–615.

[19] E. Bursztein. *Google Online Security Blog: Speeding Up and Strengthening*. [Online]. Available: https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html

[20] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," *Secur. Commun. Netw.*, vol. 9, no. 10, pp. 1226–1246, Jul. 2016.

[21] W. Diehl, F. Farahmand, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Face-off between the CAESAR lightweight finalists: ACORN vs. Ascon," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Naha, Japan, Dec. 2018, pp. 330–333.

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

[23] B. Dezfouli, I. Amirtharaj, and C.-C.-C. Li, "EMPIOT: An energy measurement platform for wireless IoT devices," *J. Netw. Comput. Appl.*, vol. 121, pp. 135–148, Nov. 2018.

[24] C. Huth, R. Guillaume, P. Duplys, K. Velmurugan, and T. Güneysu, "On the energy cost of channel based key agreement," in *Proc. 6th Int. Workshop Trustworthy Embedded Devices*, 2016, pp. 31–41.

[25] F. Zhang, R. Dojen, and T. Coffey, "Comparative performance and energy consumption analysis of different aes implementations on a wireless sensor network node," *Int. J. Sensor Netw.*, vol. 10, no. 4, pp. 192–201, 2011.

[26] G. C. C. F. Pereira, R. C. A. Alves, F. L. D. Silva, R. M. Azevedo, B. C. Albertini, and C. B. Margi, "Performance evaluation of cryptographic algorithms over IoT platforms and operating systems," *Secur. Commun. Netw.*, vol. 2017, pp. 1–16, Aug. 2017.

[27] D. M. Tung, N. Van Toan, and J.-G. Lee, "Exploring the current consumption of an Intel edison module for IoT applications," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*, May 2017, pp. 1–6.

[28] L. P. I. Ledwaba, G. P. Hancke, H. S. Venter, and S. J. Isaac, "Performance costs of software cryptography in securing new-generation Internet of energy endpoint devices," *IEEE Access*, vol. 6, pp. 9303–9323, 2018.

[29] G. Guimaraes, E. Souto, D. Sadok, and J. Kelner, "Evaluation of security mechanisms in wireless sensor networks," in *Proc. Syst. Commun. (ICW05, ICHSN05, ICMCS05, SENET05)*, 2005, pp. 428–433.

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

[31] C. Panait and D. Dragomir, "Measuring the performance and energy consumption of AES in wireless sensor networks," in *Proc. Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, Oct. 2015, pp. 1261–1266.
[32] C.-C. Chang, D. J. Nagel, and S. Muftic, "Balancing security and energy consumption in wireless sensor networks," in *Proc. Int. Conf. Mobile Ad-Hoc Sensor Netw.* Berlin, Germany: Springer, 2007, pp. 469–480.
[33] C.-C. Chang, S. Muftic, and D. J. Nagel, "Measurement of energy costs of security in wireless sensor nodes," in *Proc. 16th Int. Conf. Comput. Commun. Netw.*, Aug. 2007, pp. 95–102.
[34] C.-C. Chang, D. J. Nagel, and S. Muftic, "Assessment of energy consumption in wireless sensor networks: A case study for security algorithms," in *Proc. IEEE Int. Conf. Mobile Adhoc Sensor Syst.*, Oct. 2007, pp. 1–6.
[35] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Proc. 3rd IEEE Int. Conf. Pervas. Comput. Commun.*, Mar. 2005, pp. 324–328.
[36] K.-L. Tsai, Y.-L. Huang, F.-Y. Leu, I. You, Y.-L. Huang, and C.-H. Tsai, "AES-128 based secure low power communication for LoRaWAN IoT environments," *IEEE Access*, vol. 6, pp. 45325–45334, 2018.
[37] Q. A. Al-Haija, H. Enshasy, and A. Smadi, "Estimating energy consumption of Diffie Hellman encrypted key exchange (DH-EKE) for wireless sensor network," in *Proc. IEEE Int. Conf. Intell. Techn. Control, Optim. Signal Process. (INCOS)*, Mar. 2017, pp. 1–6.
[38] L. C. Zhong, J. M. Rabaey, and A. Wolisz, "An integrated data-link energy model for wireless sensor networks," in *Proc. IEEE Int. Conf. Commun.*, vol. 7, Jun. 2004, pp. 3777–3783.
[39] R. A. F. Mini, M. D. Val Machado, A. A. F. Loureiro, and B. Nath, "Prediction-based energy map for wireless sensor networks," *Ad Hoc Netw.*, vol. 3, no. 2, pp. 235–253, Mar. 2005.
[40] E. Morin, M. Maman, R. Guizzetti, and A. Duda, "Comparison of the device lifetime in wireless networks for the Internet of Things," *IEEE Access*, vol. 5, pp. 7097–7114, 2017.
[41] ESP8266 Community Forum. *Arduino Core for ESP8266 WiFi Chip*. Accessed: Mar. 7, 2020. [Online]. Available: https://github.com/esp8266/Arduino
[42] R. Weatherley. *Arduino Cryptography Library*. Accessed: Mar. 7, 2020. [Online]. Available: https://github.com/rweather/arduinolibs
[43] Arduino. *Arduino Playground—AvailableMemory*. Accessed: Mar. 7, 2020. [Online]. Available: https://playground.arduino.cc/Code/AvailableMemory/
[44] D. Jayasinghe, R. Ragel, J. A. Ambrose, A. Ignjatovic, and S. Parameswaran, "Advanced modes in AES: Are they safe from power analysis based side channel attacks?" in *Proc. IEEE 32nd Int. Conf. Comput. Design (ICCD)*, Oct. 2014, pp. 173–180.
[45] Sigfox. *Qualification | Sigfox Build*. Accessed: Mar. 20, 2020. [Online]. Available: https://build.sigfox.com/study

LUKE E. KANE received the B.Info.Tech. degree (Hons.) in computer science from the Queensland University of Technology (QUT), Brisbane, QLD, Australia, in 2019, where he is currently pursuing the Ph.D. degree in the Internet of Things (IoT) performance and security. He was an Associate Lecturer in network security. He is with QUT as a Sessional Academic and Teaching the bachelor's students in networking and system administration. His research interest includes implementation and design of the secure IoT architectures to support critical infrastructure.

JIAMING JAMES CHEN received the bachelor's degree (Hons.) in information technology from the Queensland University of Technology (QUT), Australia, where he is currently pursuing the Ph.D. degree in the Internet of Things (IoT) and cybersecurity. He is actively involved in a government-funded industry project which is related to the design and manufacture of a monitoring system using the IoT/ global navigation satellite system (GNSS) sensors. This automates monitoring of civil structures to reduce risks and costs in the construction and maintenance of infrastructure assets. He is also developing a network monitoring system to be used with high-volume/velocity/variety network traffic in recognised national critical infrastructure, for an electricity transmission system operator in Australia. His research interests include designing and developing the secure IoT architectures for smart cities/industries.

REBECCA THOMAS received the Master of Information Technology degree in security from the Queensland University of Technology (QUT), Brisbane, QLD, Australia, in 2019. She was with QUT as a Sessional Academic, for a period of eight months, teaching the bachelor's students in I.T. networks and network security. She is currently a Graduate Security Specialist with a leading Telecommunication company, Australia. She received the International Merit Scholarship from QUT.

VICKY LIU received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2011. She is currently a Lecturer with the Science and Engineering Faculty, Queensland University of Technology. Her Ph.D. Dissertation proposed information system architecture to facilitate the enforcement of privacy and security. She is actively involved in a number of government-funded research projects in addressing solutions for designing appropriate the IoT architectures and balancing performance and security for the IoT ecosystems. Her research interests include network and security, in particular focusing on the Internet of Things (IoT) technologies and security aspects.

MATTHEW MCKAGUE received the B.Sc. degree (Hons.) in mathematics from the University of Regina, Regina, SK, Canada, in 2004, and the M.Math. and Ph.D. degrees in combinatorics and optimisation from the University of Waterloo, Waterloo, ON, Canada, in 2005 and 2010, respectively. He was a Research Fellow with the Centre for Quantum Technologies, Singapore, and a Lecturer with the Computer Science Department, University of Otago, Dunedin, New Zealand. He is currently a Lecturer in cryptography with the Queensland University of Technology, Brisbane, QLD, Australia.