Lightweight Cryptographic Algorithms on Resource-Constrained Devices

Thomas X Meng**[1]**, William Buchanan**[2]**,
Edinburgh Napier University, Edinburgh, UK
Email: **1:** thomas−meng@outlook.com, **2:** w.buchanan@napier.ac.uk

Abstract

As the embedded device and internet of things (IoTs) concept prevalent in today's world, there is an increasing demand for the security and performance requirements on deploying these devices to private and public sectors. The crucial part of it is to protect privacy, confidentiality and integrity, meanwhile, maintain an adequate level of performance during transmission, storage and access of critical information. While the conventional cryptography methods, such as the Advanced Encryption Standard (AES), SHA−2 hashing method and RSA and Diffie Hellman for message authentication and identification, work well on systems which have reasonable processing power and memory capabilities. These do not scale well into a world with embedded systems and sensor networks, in conjunction with their nature of smaller size and lower cost. Thus, in the context of resource constrained device, lightweight cryptography methods are proposed to overcome many of the problems of conventional cryptography possessed. This includes constraints related to physical size, processing requirement, memory limitation, energy drain and production cost. This paper provides a survey of the architectures that are defined as replacements for conventional ciphers within an IoTs space and discuss some trends in the design of future lightweight algorithms. The performance metrics are carefully chosen to reflect and assess the suitability for embedded devices.

The aim of this research is to identify the various performance metrics applied on different types of symmetric block ciphers especially lightweight ciphers and compare the results on versatile platforms with stream ciphers and public key methods. The comparative analysis on efficient LW cipher will be tested against other similar block ciphers on both MacBook Pro with Intel core and resource constrained device Raspberry Pi with ARM processor.

Keywords: Lightweight cryptography · Internet of Thing (IoT) · Resource constrained devices · Encryption-decryption · Symmetric cipher · Asymmetric cipher · Speed · Efficiency · Security · Cost · Raspberry Pi

1. Literature Review

It is expected that more than 18 billion of IoT devices will be deployed to the market and connected through cloud platform by the end of 2020, more than half of them would be of industrial usage. Therefore, to assure the privacy and data protection of information during transmission is of paramount [1]. The conventional symmetric block ciphers such as AES designed in 1999 [2], DES created around 1970s are proven to be robust and efficient at that time on computer systems such as servers and desktops, tablets and smart phones, so does the public key methods like RSA-2048 and Elliptic curve-25519. However, they are considered to consume excessive physical space, processing power and battery on embedded systems such as IoT, Radio Frequency Identification (RFID) devices and micro sensors. This dilemma of choice between security and functionality also applies to other type of cryptography and hashing methods. In 2007, the authors in paper "A survey of lightweight-cryptography implementations" summarizes the development of symmetric and asymmetric lightweight ciphers on embedded hardware and software [3]. They pay particular attention on cipher attributes such as: key bits, block bits, cycles per block, throughput, and logic process (area in μm) and gate equivalents (GEs). In the same year, article [4] performed evaluation tests on several LW symmetric and asymmetric ciphers. In 2008, the LW ciphers utilized for hardware and software implementations on wireless sensor networks [5]. The key size strength of LW ciphers are usually between 80 bits to 128 bits. In case of one way authentication, 64 bits to 80 bits are considering as adequate [6].

In 2009, the authors in article [7], present performance evaluation of specific symmetric ciphers such as AES, DES, 3DES, and so on. They found out that there is not significance difference among different encoding schemes (e.g. base-64, hexadecimal) applied on the encryption results. Also the higher key size plays important role in battery drain and time consumption. The authors in paper [8] conduct cryptanalysis attacks on LW block ciphers around 2012.

Various development and implementation of LW ciphers for IoTs is carried out on [9]. Around 2016, the National Institute of Standards and Technology (NIST) issued an internal report on the topic of lightweight cryptography. In this report, the NIST discussed the plan to standardized the current lightweight cryptography methods. They not only recommended the ciphers for specific goals, but also proposed the performance metrics for both hardware and software implementations [10].

Between 2015 and 2017, Efthymios Iosifidis and Konstantinos Limniotis evaluated the performance of lightweight block ciphers in TLS protocol with IoT application [11]. They concluded that the SPECK cipher exhibits significant high performance when compared to the AES in constrained devices. Furthermore, the SPECK cipher with larger key sizes than AES might has higher throughput as well.

In [12] 2017, the authors outline the most up-to-dated symmetric primitives from different organisations include academics, government agencies, and industries.

In [13] 2017, William J. Buchanan, Shancang Li and Rameez Asif present comprehensive review on the contemporary lightweight cryptography methods used over resource-limited devices. The article outlines their designs, strengths and weaknesses in term of performance and security.

In 2018, Tasnime Omrani, Rhouma Rhouma, and Layth Sliman in their article [14] analyses several popular lightweight ciphers on the IoT devices include Raspberry Pi and Arduino UNO. Their performance metrics are RAM and ROM consumption as well as execution time and security. They concluded based on their results that SPECK is the most memory efficient cipher among the rest, and about half the footprint of AES. The PRESENT is the most secure one against the cryptanalysis. The same conclusion has been drawn by a conference paper in 2013, that the SPECK is the most RAM, ROM and speed efficiency amongst other LW block ciphers (Hight, Present, PrintCipher, Katan- Katantan, Klein, Twine, SIMON, SPECK, Pride, Hummigbrid, Lblock, Mips and Piccolo) [15]. The author also identify that Rabbit is the best among other stream ciphers such as RC4, Grain, Salsa20, TRIVIUM and so on.

In 2018, another paper called "Optimal Lightweight Cryptography Algorithm for Environmental Monitoring Service Based on IoT" implements and investigates the security protocols for IoTs devices used for controls the surrounding temperature and humidity. The authors compared the CPU usage and processing time of various lightweight ciphers on Raspberry Pi and Arduino UNO [16].

The IoT are the next generation of technological innovations, and will occupy our everyday life, that includes the digital ID or the small home appliances we deemed as life essentials as well as the wireless security and fire alarms, this list is not exhaustive. The data security and privacy protection are imperative in this context, thus the lightweight zero knowledge proof methods are introduced [17] [18]. The Estonia government has already replace their RSA key to ECC for eID due to ROCA vulnerability [19]. In order to cope with the size and power drain requirements of the embedded systems, the NIST and international organisations like ISO/IEC (International Organization for Standardization and the International Electrotechnical Commission) outlines different Lightweight cryptography specifically designed for such systems[20]. In general, the three pillars for the lightweight cryptography are [13]:

- Processing power (CPU and RAM usage, execution time)
- Battery power (CPU and RAM usage, power drain, execution time)
- Physical space (RAM usage, number of logic gates, area)

Along with these, we also have ROM, the amount of memory specific method occupied which related to the code size and efficiency. The implementation of any ciphers should be publicly available. Because the Kerckhoffs's principle in conjunction with Schneier's law emphasised that the importance of transparency in the cryptography world. The existing methods are tested and scrutinised open source by the best cryptographers in the world. Thus, they can be trusted to a degree level.

In the article too much crypto, the author concludes that the current number of rounds used by modern cyptos are far more than enough for long term security. They proved the hypothesis by proposing reduced-round versions of AES, BLAKE2, ChaCha, and SHA-3 with equivalent security but faster than their full round versions [21]. Furthermore, the author also reviewed the literature and summarised that the practical pre-image attacks against hash function is no more than 2.75 rounds. For instance, with stream cipher ChaCha, the plausible attack is against its 7 round version with a key recovery method [22]. The increase in key sizes does not results in noticeable change in energy consumption. The symmetric encryption method based on AES can be robust and fairly secure if proven provide security stronger or equal to 7 rounds of AES [23]. Thereby, the Lightweight cryptos can reduce the so called security margin (rounds) to trade for efficiency. After all, the execution time and battery drain are vital for embedded systems like IoTs.

As aforementioned, many of the attacks proposed in journals are theoretical and hypothesized on reduced rounds, block and key sizes, which are not applicable in practice. For instance, in articles [24] and [25], the differential attack is only probable with significantly reduced rounds of CLEFIA. Thus, for low cost and short term security applications like embedded devices, performance and speed over high security is sought. One way to assess the strength of cipher is to define the area that the cryptography function will use on the device in unit μm^2. However, there does not exist a universal set of rules to define the strength of cryptography. They depends on various factors such as: the key lengths and randomness (entropy), block sizes, rounds, operations, mathematical algorithms, known attacks, and at last but not the least, how people use them. If an incapable person installed the cipher on a vulnerable system and accessed the internet using a unsafe tunnel and network, no matter how intricate the cipher is in its design, it will be compromised. Since most of the IoTs will work in multi-task mode, the software performance will be prioritised. The most software-oriented on the table 2 of paper "A survey of lightweight-cryptography implementations" are: AES, IDEA, TEA (XTEA, XXTEA), and so on [13] [12] [3].

Later 2019, the author readdressed the importance of secure and effective lightweight cipher with small hardware footprints, in the current global trend of 5GN (5th generation networking) smart city and applications in different layers. They compares different block ciphers includes but not limited to PRESENT, SIMON, SPECK with the classic AES methods [1].

There are many types of classification for LW cryptography. One of the prominent ones is: Substitution-Permutation Network (SPN), with S-box and P box structure. Feistel Networks (FN) based ciphers. Generalized Feistel Networks (GFN) based ciphers. Hybrid ciphers. Add-Rotate-XOR (ARX) operations based ciphers. And the nonlinear feedback shift registers (NLFSR) based ciphers. In this article, we will mainly focus on SPN, GFN, and ARX ciphers.

2. Computational Devices

The Raspberry Pi is a well-known pocket size computer system built for educational purposes. Its OS called Raspbian is a Debian-based computer operating system supports Linux functionalities. It is widely used as an comparison to the IoT devices and applications in literature [26]. In the table

1, we have an overview of technical specifications between the MacBook Pro and Raspberry Pi:

Table 1: Technical specifications

Device	**MacBook Pro**	**Raspberry Pi 4**
SoC	n/a	Broadcom BCM2711B0
Processor	3.1 GHz Intel Core i7-5557U	64-bit quad-core ARM Cortex-A72 1.5 GHz
RAM	16 GB 1867 MHz DDR3	LPDDR4 4GB
GPU	Intel Iris Graphics 6100 1536 MB	VideoCore VI 500MHz
OS	MacOS High Sierra 10.13.6	Raspbian GNU/Linux 10
GPIO Pins	n/a	40 pin GPIO header
Power	5 volts @ 3 amps	5V at 3 amps via USB Type-C
Storage	60GB SSD left	16GB SD card

The Raspberry Pi is brand new, and set up through SSH from laptop, connected to the internet through WiFi. It runs ARM (armv7l) processor with 4 CPU(s) at 600-1500 MHz. The MacBook pro runs Intel (x86_64) processor, and a Ubuntu OS virtual machine with 2 CPU(s) at 3099.194 MHz. Figure 1 depicts the experimental tools for this analysis.

3. Symmetric Cipher

The symmetric key methods are the workhorse of the industry, it essentially covers the confidentiality of the information whether during transit or static. Also, they are mainly used for data encryption and decryption process. The AES type ciphers included in ISO/IEC 18033-3 are suggested to be formidable for the embedded devices, but it can serve as an indicator how well the other symmetric ciphers performs when compared with the former. The smaller block size such as 64 bits or 80 bits with smaller keys less than 90-128 bits and less complex rounds with smaller S-boxes are considered to be under the umbrella of lightweight symmetric cipher. The symmetric ciphers are know to possess two problems: key distribution, key management.

3.1. Block Cipher

One of the main features of block cipher is that the key length is proportional to the security level. Thus, lightweight cipher with flexible key length

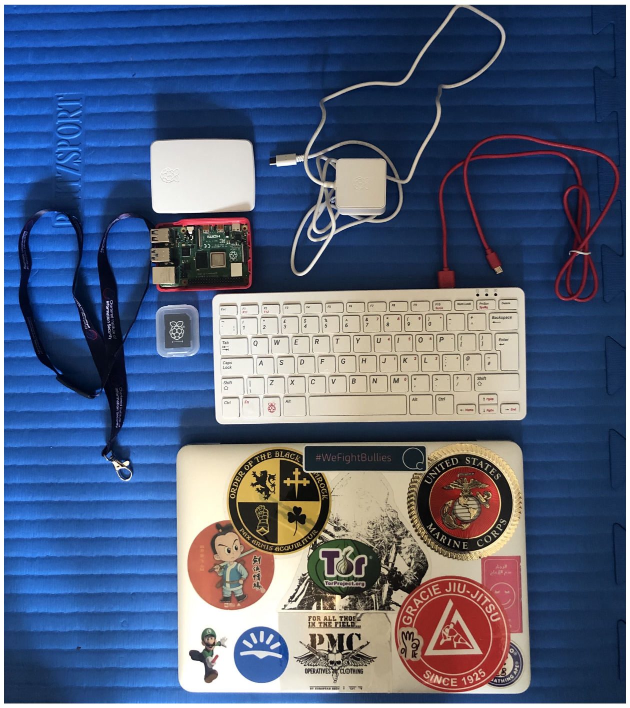


Figure 1: Experiment setup

is important in the context of resource-constrained devices. The block ciphers are known for prone to birthday attack, the length of the key and initialization vector (IV) is thereof has to be sufficiently large compare to lifespan of device. One of the criterion for the design of cryptographic algorithm is avalanche effect, such that tiny changes in the input will results in massive changes of bits in the output.

3.1.1. Substitution-Permutation Network Based Cipher

The larger the S-box the higher the security level against differential and linear analysis, yet more footprints. The AES Rijndael prone to several known attacks such as copy-and-paste attack, brute force attack and non-random numbers and differential crypto analysis. The AES-128 has key length of 128, 192 and 256 bits respectively, and corresponding rounds of 10, 12 and 14 with identical block size of 128 bits. Since ECB (Electronic Cipher Block) is not safe in practical like revealing the size of data, we will only consider the application of CBC (Cipher Block Chaining), OFB (Output Feedback), CFB (Cipher Feedback) and CTR (Counter mode, stream cipher).

The current compact hardware implementations of AES-128 only requires 2090 to 2400 GEs.

AES with Key sizes above 72 bits would be suffice and requires GPU to crack it [21]. Although it is theoretically plausible to crack a 7 rounds AES block cipher, it is impractical to do so [27]. Because it would requires 2^{99} encryption operations, with 2^{97} chosen plaintexts and 2^{100} bytes of storage. The author also includes the key-recovery attacks to their list in [28].

The main drawbacks for AES usage in embedded devices are that fixed block size of 128 with merely three possible key sizes. The number of rounds is depends on the length of the key 10, 12 and 14 rounds respectively. It is not flexible enough to adjust its configurations to suit particular needs of the devices or microchips. Consequently, AES leads to relatively large amount of memory, processing power and battery drains. In the same vein, throughput is low for AES, which means it takes security to a high standard but the actual performance is relatively low and takes up lots of spaces and logic gates to built it.

PRESENT is a hardware oriented LW block cipher, it was firstly proposed in 2007 at Conference on Cryptographic Hardware and Embedded Systems (CHES). It has minimal hardware footprint of 1570 GE claimed by the author, which is designed for passive RFID tags. PRESENT has a 80 bit or 128 bit encryption key. It uses a 64 bit block size and has a 16-value substitution-box layer, and which is a nibble-wise nonlinear substitution. The p-box layer it goes through is a bit permutation box. The parts of the encryption key is EX-OR within 32 rounds of operations. The decryption process is simply the reverse of encryption, so does the s-box (4X4) and p-layer. In contrast with the Intel processors generally 32 or 64 bits, the PRESENT is kind of ideal for IoT devices with 9 or 16 bit processors. The PRESENT cipher are known to have weaknesses such as side channel attacks and memory leakage problems similar to AES. In the paper [29], the authors conduct power analysis against software implemented PRESENT, their results demonstrate that the Add Round Key function is vulnerable to information leakage.

SKINNY is a LW block cipher with flexible key sizes range from 62 bits to 384 bits, and block sizes either 64 bits or 128 bits. It constitutes of substitution cell (SC), add constants (AC), shift rows (SR) and mix columns (MC).

Prince is a SPN based block cipher with a low latency for pervasive computing applications proposed in 2012. It has 64 bit block size and 128 bit key. Prince is specifically designed to overcome the latency [30].

Pride is another LW block cipher with SPN structure. It has fixed block size of 64 bit and key size of 128 bit with 20 operation rounds. No known attacks to date.

3.1.2. ARX based

SPECK and SIMON are ultra lightweight ARX based, modular addition only block ciphers designed for both hardware and software, developed by the NSA (national security agency) in 2013. This type of cipher is robust against side channel attacks compared to AES-like cipher with substitution box. The key sizes are 64, 72, 96, 128, 144, 192 and 256 bits, and $2n$ bit block sizes of 32, 48, 64, 96 and 128 bits. In case of a 32 bit blocks with 64-bit key and 32 rounds or 64-bit blocks with 128-bit key and 34 rounds. It is optimised for performance in software implementations. The operations include bit-wise exclusive OR, modular addition 2^n, left and right circular shifts S^i by i bits and S^{-j} by j bits. SIMON uses AND gates instead of addition. The fixed round key k is calculated via a key schedule mechanism at each round [31] [32]. The round function R_k with key k has inputs x and y such that:

$$R_k(x, y) = (\Phi_k(x, y), S^{\beta}(y) \oplus \Phi_k(x, y)) \tag{1}$$

where $\Phi_k(x, y) = (S^{-\alpha}(x) + y) \oplus k$, for rotation parameters $\alpha = 8$ and $\beta = 3$. The only exception is that when have 32 bit block size, $\alpha = 7$ and $\beta = 2$. $\oplus$ denotes modular addition. The code below is an example of round function presented for encryption:

```
// https://github.com/inmcm/Simon_Speck_Ciphers/blob/master/Python/
    simonspeckciphers/speck/speck.py
def encrypt_round(self, x, y, k):
    """Complete One Round of Feistel Operation"""
    rs_x = ((x << (self.word_size - self.alpha_shift)) + (x >> self.alpha_shift)) &
        self.mod_mask

    add_sxy = (rs_x + y) & self.mod_mask

    new_x = k ^ add_sxy

    ls_y = ((y >> (self.word_size - self.beta_shift)) + (y << self.beta_shift)) & self
        .mod_mask

    new_y = new_x ^ ls_y

    return new_x, new_y
```

The inverse of round function for decryption uses modular subtraction instead of modular addition. It is shown that SPECK with 128 bit block size and key length only requires 1396 GE compared to 2400 GE required by AES [31]. This implies that the smaller the circuit size, the lower the power drain and memory consumption. In term of security, is has been discussed in literature, several recent cryptanalysis on SPECK is presented, results substantiate that the full-round SPECK is resistant to all known cryptanalytic techniques so far with appropriate key sizes [32].

XTEA stands for extended tiny encryption algorithm originally designed by David Wheeler and Roger Needham in 1997. It is a Feistel network based block cipher with 64 bits of block, key length of 128 bit, and suggested 64 rounds. The major advantage with XTEA is its compact code size, hence the memory (ROM) requirements. It is quite interesting that the memory usage on my MacBook pro is larger than on Raspberry Pi. In contrast with the time required to run the same bit of code. The test message is "hello123" for XTEA, loop over 1000 times of encryption and decryption process. The method TEA is known suffered from several weakness, includes equivalent keys and related key attack, hence the XTEA is designed. An successor cipher XXTEA is introduced in 1998, It further enhances the strength of block cipher type TEA. Though the XXTEA is vulnerable to a chosen-plaintext attack, it is very much a theoretic estimate. Because it requiring 2^{59} queries for a block size of 212 bytes or more, and with 2 additional rounds will fix the problem.

3.1.3. Generalized Feistel Networks Based Cipher

CLEFIA is one of the Generalized Feistel Networks (GFN) family cipher proposed on 2007 by Sony, included in ISO/IEC 29192–2:2012. Its name in French means key. It is designed for DRM protocols with both hardware and software implementations [33]. It has 128, 192 and 256 bit keys corresponding to 18, 22 and 26 rounds, and a 128 bit block. CLEFIA has been under public scrutinised since 2007, no worth mentioning security issues to date. Its sister cipher is called Piccolo. It can only be attacked with reduced rounds to date. In article [34], CLEFIA has been implemented in hardware with various gate sizes and compared to AES, Camellia and SEED ciphers. It has been demonstrated that CLEFIA is superior in terms of throughput and gate (gate efficiency).

3.2. Stream Cipher

The stream cipher family is designed for smaller data size because no block size is required. On the other hand, the block cipher requires padding which is consuming additional memories, and decrease the encryption and decryption speeds. With variable key sizes you can adjust the level of security you needed for particular device and hardware. The main drawback of stream cipher is the lengthy initialisation process before the actual encryption process and lack of implemented protocols.

Rabbit is a lightweight stream cipher and was first presented by Martin Boesgaard, Mette Vesterager, Thomas Christensen and Erik Zenner at an encryption workshop in 2003 [35]. It creates a key stream from an 128 bit key and a 64 bit IV. The use of IV can prevent collision and pre-image related attacks. The encrypted value has equal bit-length to the message, thus no overhead storage.

Rabbit is nearly twice as fast as RC4 (Rivest Cipher 4) and HC-128 is about 5 fold, and is about 3 times faster than AES. Therefore, if you have speed concerns about SSL, using more compacted version of SSL with lightweight stream ciphers should alleviate the performance burden [36]. Also, it can be used in Elliptic Curve Integrated Encryption Scheme (ECIES) as key stream derivation function for the shared session secret produced by the elliptic curve method.

Trivium is another low footprint hardware utilised lightweight stream cipher proposed by Christophe De Canniére and Bart Preneel. It has key stream and IV both of 80 bits and output of 2^{64} bits.

KATAN and KTANTAN is a hardware oriented LW block cipher family, aimed to overcome the physical footprint in the hardware. KATAN hardware a fixed key into its circuit. It is vulnerable to several attacks like Meet-in-the-Middle concepts, differential attacks with reduced rounds [37][38]. KATAN is presented in this section because it is designed based on stream cipher trivium's sister called bivium.

4. Asymmetric Cipher

Asymmetric ciphers can be used as part of the public key infrastructure (PKI) to perform key exchange, identification, authentication and various other applications during data transmission and storage. It also assures the non-repudiation of transaction and documentation by signature generation and so on (TLS/SSL, web applications). The mathematical concept behinds asymmetric ciphers are the so called trapdoor function. The security level for majority of symmetric encryption methods are the key length itself, thus it is relatively straightforward to quantify the security and performance trade-off. However, things get complicated with the factorisation based method and the discrete logarithm method in finite field. For instance, the 256 bits RSA key does not as secure as it sounds when compare with 160 bits ECC (Elliptic Curve Discrete Logarithm in finite field) [39]. Hence, the RSA method (integer factorisation in prime field) is not an ideal choice for IoTs. There is other types of asymmetric cipher includes but not limited to: authenticated lightweight key exchange (ALIKE), ID-based signature scheme (IBS), Rabin (WIPR), ElGamal, TinyECC, HyperECC and the WMECC.

The PKI plays a vital part in the cryptography worlds. It responsible for certificate signing process from root and intermediate certificate authorities (CA) where the CA will use their private key to sign the issued certificate once confirms clients' identity. The issued digital certificate contains both the public key and private key. The party who distributed certificate for the purpose of authentication and identification should keep the private key in secret. The authentication can be achieved by message authentication code (MAC) like Chaskey cipher. Public key method can also be used in key exchange methods like generating the session keys or the keywords for the key derivation function (symmetric ciphers). The PKI can be employed as digital authentication and identification method between digital ID, passports and other digital ports/stations in different security levels. It can provide you with the private key to sign documents with your unique identifier and conducts other activities like paying tax.

One of the method proposed by ISO/IEC is Elliptic Light (ELLI) which is based on the famous elliptic curve cryptography with Diffie-Hellman related handshake between RFID tag and RFID reader, sensor networks [40]. In the thesis of Sandeep S. Kumar, he conducted a thoroughly analysis on ECC in constrained devices, he highlighted that the ECC has smaller key sizes, operand length and relatively low arithmetic needs [41]. On the other hand, the ECC type cipher has the advantage of lower CPU demand, less physical space taken by an SSL certificate, smaller bandwidth and fewer power consumption when compared to RSA and DSA methods. In the paper "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs" [42], the authors revisited ECC of 160, 192 and 224 bits fields versus the RSA with 1024 and 2048 bits on two 8-bit micro controllers. They pointed out that the asymmetric cipher is applicable on constrained devices without hardware accelerations. The advantage of ECC over RSA is increasing with decreasing computational power and increased key lengths.

The ELLI methods works exactly the same way as ECC. To briefly introduce it, we choose a given elliptic curve $y^2 = x^3 + ax + b \pmod{P}$ with given parameters and base point G on the curve, like the well known curve 25519, with prime number $P = 2^{255} - 19$. The user choose a private key λ usually 256 bits, thus, the pubic key is another point $A = \lambda G$ which is 512 bits for ECC, λ is the gradient of G. The generator G is chosen such that the subgroup order N has to be large enough that the x, y coordinate values does not cyclic within the range of number of additions λ.

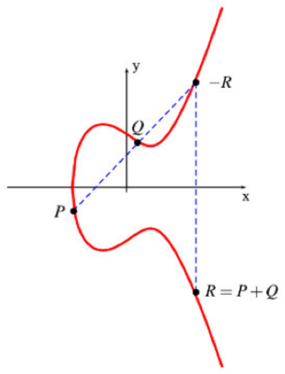


Figure 2: ECC plot

The RFID reader generates λ, then compute $A = \lambda G$, the reader pass A to the RFID tag. The tag creates $B = \epsilon G$, and generate a hash signature of the value B signed by the private key of the manufacturer which the reader can validate, where ϵ is the random private key value of 256 bits. The tag contains group $(\epsilon, B, \text{PublicKeySign}(B))$. When validating the tag, reader send A, the tag computes $C = \epsilon \times A$ which is a scalar multiplication. Then the tag sends the reader group $(B, C, \text{PublicKeySign}(B))$. The reader can thus calculate $D = \lambda \times B$ which proves $D = C$. Hence, the private key is verified. The intruder could not determine the value of C and D even if they have the challenge value of $A = \lambda G$ and the curve parameters.

The scalar multiplication is performed λ times until we reach the point A. We normally use a point addition or a point doubling. We can use the Montgomery Ladder method, which produces a fixed time calculation instead of exponential (normally related to the number of bits in λ). In ECC where 160-bits keys are equivalent to 1024 bits RSA keys, and 80-bit symmetric key encryption. Thus the key strength and the entropy is not the only measure of security level, but it can serve as an indicator how much effort in time and memory it takes for the brute force attack to crack the key with certain sizes. The 256 bit ECC key is equivalent to RSA 3072 bit keys (128 bits key strength). It is suffice to have key length 256 bits within reasonable amount of time and resources, it would take roughly the energy that can boil all the water on the planet earth to crack it [39].

It should not be computationally possible, within an reasonable time period, to determine the scalar private key between the generator and the public key value. Within Bitcoins, we use the private key to sign a transaction, and then which is proven by the public key (Elliptic Curve Digital Signature Algorithm). The value of λ is large, and prime number P is large, even we know A and G, its very difficult to determine the value of λ.

5. Testing and Evaluation

In this section, the experimental results of LW ciphers will be presented. The sketch of security level of lightweight ciphers are listed with their block sizes, key lengths, rounds, in the table below [36] [32] [43] [44] [45] [46].

	Cipher	**Block bits**	**Key bits**	**Round**	**Area GEs**
Block 1	AES-128	128	128(192, 256)	10 (12, 14)	2400(3400)
2	XTEA	64	128	64	3490
3	SIMON	32-128	64-256	32-72	763(1234)
4	SPECK	32-128	64-256	22-44	884(1280)
5	PRESENT	64	80(128)	31	1075(1884)
6	SKINNY	64(128)	64-384	32-56	1223(4268)
7	KATAN	32,48,64	80	254	462(1054)
8	Prince	64	128	12	3286(3491)
9	Pride	64	128	20	n/a
10	CLEFIA	128	128-256	18-26	2996(4950)
Stream 1	RC4(flexible)	80(IV)	128		11300(≈50000)
2	Rabbit(flexible)	64(IV)	128		3800(4100)
3	Trivium	80(IV)	80		2599(4900)
Asymmetric 1	RSA(3072)		1024 to 4096 modulus		50000
2	ELLI(25519)		256 private key	256 public key(x coordinate)	6660(18121)

Table 2: Prominent features of contemporary lightweight ciphers.

One of the criteria for LW cipher is between $1000 - 3000$ GEs, of which lesser than 1000 logic gates are desirable and categorised as ultra-lightweight ciphers. The number of GEs depends on the implementation to suits specific security requirements. The serialized implementation with goal of optimised area will present less GEs in contrast with the round-based implementations.

Most of the GEs in the table above are area optimised figures in opposite to the speed optimised implementation. It should be noted that the majority of LW ciphers are flexible on their key length and block size, user can adjust the security level based on their need and application. For instance, SPECK, SIMON and SKINNY have flexible block and key size arrangement with various rounds, which can tailor to suits specific lightweight devices depends on battery supply and security needs.

5.1. Implementation

There is a famous quote, don't cook your own crypto , because the home made crypto is neither safe nor efficient. The existing cryptos are already widely scrutinised and proven to withstand many prolonged and harshest attacks by many researchers and industries. Thus, we will use the most recognised codes in this paper for testing purpose instead of creating our own codes.

The Appendix gives an outline of the Python code used. The majority of codes are obtained from "*www.asecuritysite.com*", modifications have been made to add the test functions and performance analysis. The test vectors for ciphers are obtained from either Internet Engineering Task Force (IETF) or the NIST.

5.2. Performance Metric

The design aim of Lightweight primitives are smaller internal states, tiny block and key sizes, which does not compromise too much security. When we load up the cipher, it has footprints in both the RAM and the flash area, or the footprints when its running in the memory, if we need greater buffer for memory spaces and cache area. The size of chip required for certain type of manufacture is defined in μm^2. The size of the implementations whether on hardware or software has impact on the memory occupancy and cost. The type of programming language can results in different execution times thus efficiency. The faster the set of instructions can be performed, the quicker the processor can return to an idle state or sleep mode to minimize the power consumption. The overhead difference associated with software measurement of execution time is negligible compares to oscilloscope based methods [47]. Thereby, we can adopt the python built-in software to measure the execution time.

The performance and robustness of particular primitives require adequate measurement and criterion. They can be key size, vulnerability to known attacks, execution time with different types of data, different programming languages used, flexibility of application on various platforms, randomness of cipher (entropy), cost, so on and so forth. The number of GEs are important in the hardware implementations, it dictates the number of logic gates required to install the particular cipher, smaller GE number means cheaper circuit and less power consumption. For instance, a 4-8 bits micro controller would require around 1000 GEs or below to fulfill its design goals. Power analysis on RFID tags and power constrains devices are vital. The latency is another indicator for hardware implementation.

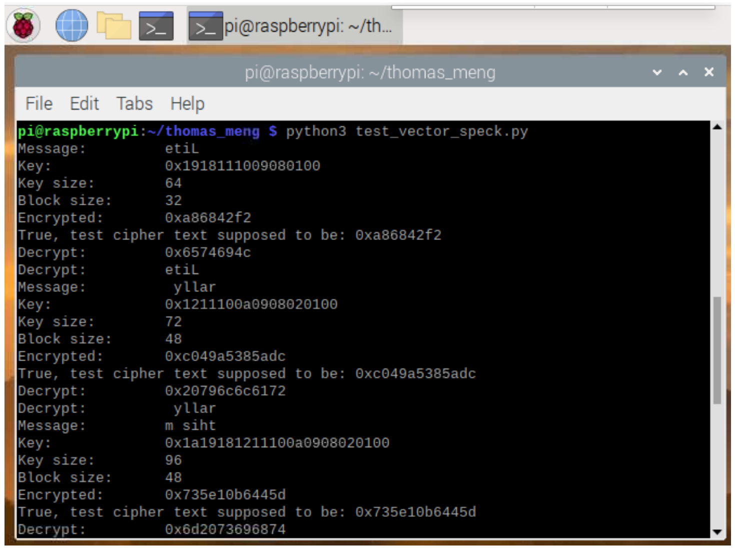
The popular performance criterion for software implementation is the implementation size and RAM consumption and the throughput. Also the encryption and decryption speed as execution time. Because the constraint on the available microprocessor in device.

The evaluation in this article is nothing comparable to the FELICS' (Fair Evaluation of lightweight Cryptographic Systems) results of popular lightweight cipher algorithms on three microcontroller: 8-bit AVR, 6-bit MSP and 32-bit ARM. Here, we only assess a wee bit of ciphers on both the Raspberry Pi and MacBook Pro with Ubuntu OS and Raspbian.

5.3. Performance Evaluation

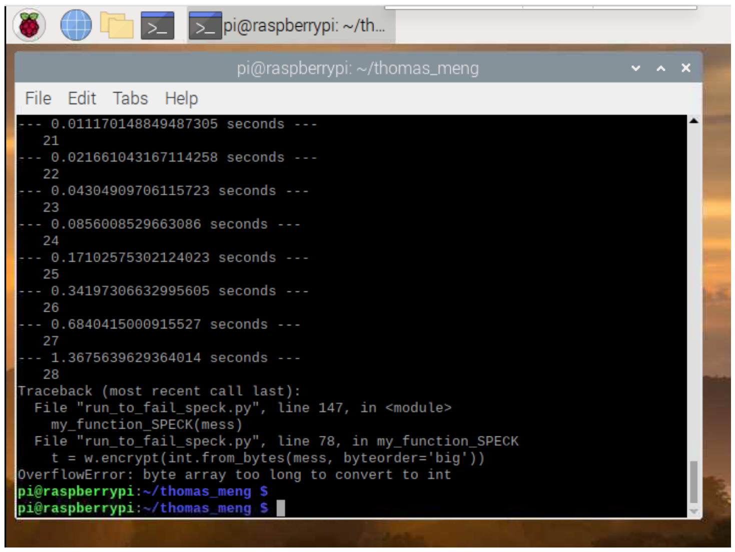
In this performance test, we will focus on the software implementation, thus the memory (RAM), CPU, execution time and throughput associated with the encryption/decryption process will be the main focuses. The RAM indicates the space to store temporary data, the CPU percentage presents the amount of processing power used to run the cipher, related to the power drain. The execution time is a direct measurement of code efficiency and number of clock cycle has positive relationship with the CPU time.

In the validation phase with chosen test vectors, we check whether the decrypted ciphertext matches the plaintext before encryption. This phase only uses AES and SPECK as representatives. The rest of the ciphers are not posted in this paper due to length of contents. The test vectors for SPECK and AES are chosen from paper [31] and [48]. It should be noted that the input message is in ASCII format, the key-phrase is in hexadecimal. Figure 3 and Figure 4 depict the test vector results for SPECK cipher in Python implementation, as well as increase the input message size until failure on Raspberry Pi.



```
pi@raspberrypi:~/thomas_meng $ python3 test_vector_speck.py
Message:          etiL
Key:              0x1918111009080100
Key size:         64
Block size:       32
Encrypted:        0xa86842f2
True, test cipher text supposed to be: 0xa86842f2
Decrypt:          0x6574694c
Decrypt:          etiL
Message:           yllar
Key:              0x1211100a0908020100
Key size:         72
Block size:       48
Encrypted:        0xc049a5385adc
True, test cipher text supposed to be: 0xc049a5385adc
Decrypt:          0x20796c6c6172
Decrypt:           yllar
Message:          m siht
Key:              0x1a19181211100a0908020100
Key size:         96
Block size:       48
Encrypted:        0x735e10b6445d
True, test cipher text supposed to be: 0x735e10b6445d
Decrypt:          0x6d2073696874
```

Figure 3: SPECK test vectors on Pi [31]

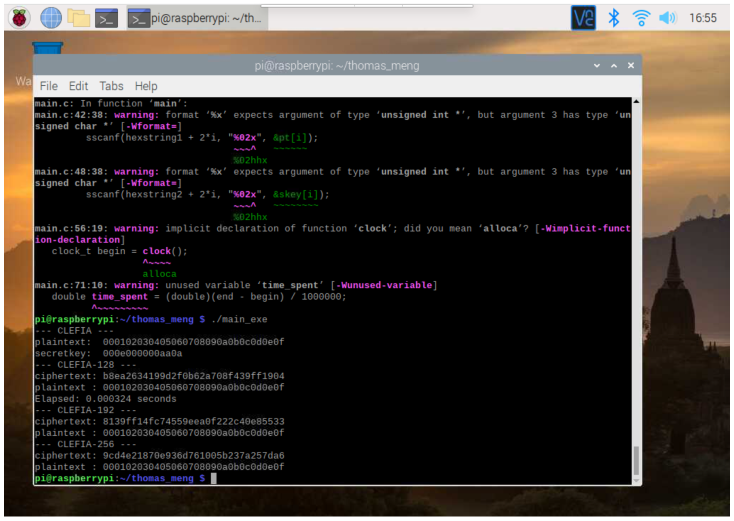


```
--- 0.011170148849487305 seconds ---
   21
--- 0.021661043167114258 seconds ---
   22
--- 0.04304909706115723 seconds ---
   23
--- 0.0856008529663086 seconds ---
   24
--- 0.17102575302124023 seconds ---
   25
--- 0.34197306632995605 seconds ---
   26
--- 0.6840415000915527 seconds ---
   27
--- 1.3675639629364014 seconds ---
   28
Traceback (most recent call last):
  File "run_to_fail_speck.py", line 147, in <module>
    my_function_SPECK(mess)
  File "run_to_fail_speck.py", line 78, in my_function_SPECK
    t = w.encrypt(int.from_bytes(mess, byteorder='big'))
OverflowError: byte array too long to convert to int
pi@raspberrypi:~/thomas_meng $
pi@raspberrypi:~/thomas_meng $
```

Figure 4: SPECK cipher run until fail

Figure 5 shows the code obtained from "*www.asecuritysite.com*" runs on Raspberry Pi with gcc compiler system.

In the performance test phase, all the tests are performed with 1000 $\times$ of (encryption + decryption).



```
main.c: In function ‘main’:
main.c:42:38: warning: format ‘%x’ expects argument of type ‘unsigned int *’, but argument 3 has type ‘un
signed char *’ [-Wformat=]
         sscanf(hexstring1 + 2*i, "%02x", &pt[i]);
                                   ~~~^   ~~~~~~
                                   %02hhx
main.c:48:38: warning: format ‘%x’ expects argument of type ‘unsigned int *’, but argument 3 has type ‘un
signed char *’ [-Wformat=]
         sscanf(hexstring2 + 2*i, "%02x", &skey[i]);
                                   ~~~^   ~~~~~~~~
                                   %02hhx
main.c:56:19: warning: implicit declaration of function ‘clock’; did you mean ‘alloca’? [-Wimplicit-funct
ion-declaration]
   clock_t begin = clock();
                   ^~~~~
                   alloca
main.c:71:10: warning: unused variable ‘time_spent’ [-Wunused-variable]
   double time_spent = (double)(end - begin) / 1000000;
          ^~~~~~~~~~
pi@raspberrypi:~/thomas_meng $ ./main_exe
--- CLEFIA ---
plaintext:  000102030405060708090a0b0c0d0e0f
secretkey:  000e000000aa0a
--- CLEFIA-128 ---
ciphertext: b8ea2634199d2f0b62a708f439ff1904
plaintext : 000102030405060708090a0b0c0d0e0f
Elapsed: 0.000324 seconds
--- CLEFIA-192 ---
ciphertext: 8139ff14fc74559eea0f222c40e85533
plaintext : 000102030405060708090a0b0c0d0e0f
--- CLEFIA-256 ---
ciphertext: 9cd4e21870e936d761005b237a257da6
plaintext : 000102030405060708090a0b0c0d0e0f
pi@raspberrypi:~/thomas_meng $
```

Figure 5: CLEFIA (C) cipher performance on Pi

1. The code execution time in second and memory in GB for various ciphers are recorded on the Raspberry Pi and MacBook Pro run for one thousand loops of the "encryption and decryption process". Also the memory (RAM) consumption and CPU usage in percentage are compared between two devices.

2. Increase the size of the random generated message ($/dev/urandom$ Linux source of pseudo randomness) in bytes (or file) until failure or took too long to compute on specific methods. For instance, to gradually increase the size of the message from 2 bytes to 2048×2^{30} bytes. The throughput is calculated as bytes/second.

In our test examples, the different modes of AES does not exhibits significant differences on their CPU usage and RAM consumption. The modes are: EBC, CBC, CFB, OFB, and CTR respectively. However, their execution time and CPU time is distinguishable, the EBC is the fastest but not secure, CTR and CBC are the second fastest modes compare to CFB and OFB in both Macbook Pro and Raspberry Pi. CBC type mode do requires additional key chaining process compares to ECB. The CFB is the slowest mode among others. This is anticipated since the ECB demands less steps on masking the fixed pattern of cipher blocks. The AES in CTR mode is relatively secure and have application in steam encryption [15].

In the above figure, we can observe both the average mean of execution

doi:10.20944/preprints202009.0302.v1

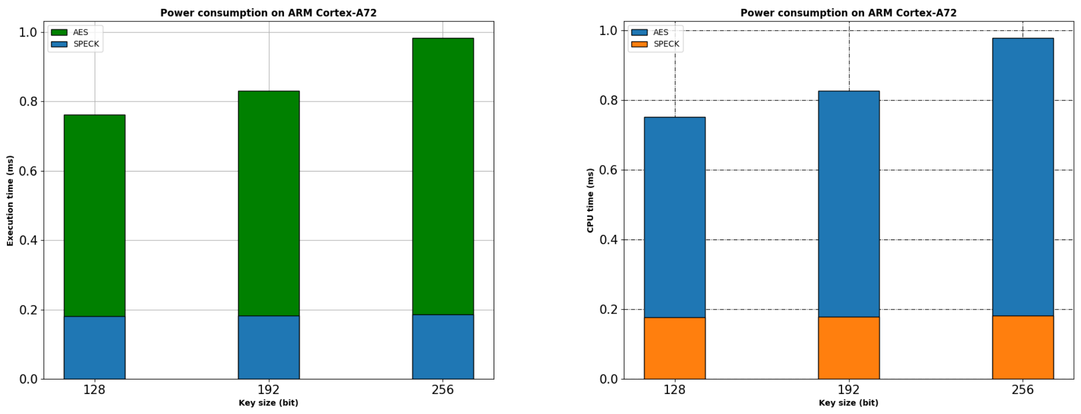


Figure 6: Execution Time, CPU Time (ms), SPECK-128 and AES-128, Raspberry Pi

time and CPU time over 10000 iterations for SPECK and AES cipher. The size of the block is 128 bit for AES and SPECK ciphers with key sizes range from 128 to 256 bit, for a fair comparison. It is shown that the SPECK cipher is more robust with increasing key sizes than the AES cipher on portable device. For SPECK, the key size from 128 to 192 bit cause increase in time about 1.16%, and a 1.98% increase in time from 192 to 256 bit. On the flip side, the key size for AES from 128 to 192 bit cause increase in time about 8.23%, and a 15.39% increase from 192 to 256 bit. On Macbook pro, the results exhibit similar trends, there is no significant overhead with increasing key sizes regarding to computational costs incurred by SPECK cipher.

In the table 3, the data in the parentheses are the numerical results obtained from Raspberry Pi.

The range of message size is from 2 bytes to 134MB. The execution time, memory are rounded to two significant figures. The Pi will only be able to run AES cipher with message size up to ≈ 1048KB. Throughput measured with clock rate at 3099 MHz on Macbook Pro (actual frequency at 3099.343 MHz), and 600 kHz on Raspberry Pi. From table 3, we can observe that the execution time for both ciphers are faster on MacBook pro than on Pi, which is as expected. On the other hand, the ARM processor is more efficient on memory usage, which can be seen from the table 4. The Raspberry Pi consumes way less memory than MacBook pro to run both AES and SPECK ciphers. Surprisingly, SPECK cipher requires more RAM usage than AES-128 in general, though it is faster than the latter. The RAM usage

Message size (bytes)	**Throughput** (AES)	**Throughput** (SPECK)	**Time** (AES)	**Time** (SPECK)
2B	6.4K (1.4k)	34k (10k)	0.21 (1.2)	0.15 (0.17)
4B	17k (3.4k)	78k (23k)	0.19 (1.1)	0.087 (0.16)
8B	37k (6.9k)	162k (47k)	0.21 (1.1)	0.0801 (0.16)
16B	54k (9.7k)	341k (94k)	0.26 (1.6)	0.076 (0.16)
32B	63k (14k)	692k (189k)	0.33 (2.02)	0.087 (0.16)
64B	129k (19k)	1443k (378k)	0.48 (2.9)	0.048 (0.17)
128B	140k (22k)	2902k (753k)	0.76 (5.4)	0.0507 (0.16)
256B	186k (21k)	5836k (1477k)	1.4 (8.8)	0.049 (0.16)
512B	204k (29k)	11484k (2853k)	2.8 (18)	0.048 (0.17)
1 KB	175k (27k)	22139k (5743k)	11 (35)	0.0548 (0.17)
2 KB	63k (30k)	34918k (7457k)	12 (66)	0.0524 (0.21)
4 KB	138k (29k)	77737k (17249k)	28 (130)	0.0565 (0.22)
8 KB	153k (30k)	58040k (30596k)	51 (280)	0.092 (0.26)
16 KB	180k (30k)	17380k (54583k)	90 (510)	0.17 (0.3)
32 KB	166k (30k)	151032k (80751k)	204 (1010)	0.13 (0.4005)
64 KB	205k (31k)	339355k (109557k)	302 (2070)	0.18 (0.88)
128 KB	198k (31k)	415223k (93432k)	610 (4060)	0.32 (0.97)
256 KB	200k (30k)	378620k (76889k)	1300 (8200)	0.85 (2.7)
512 KB	203k (31k)	553074k (78049k)	2700 (16000)	1.1 (6.7)
1 MB	n/a	565811k (77996k)	n/a	2.2 (13)
2 MB	n/a	602431k (78999k)	n/a	3.7 (26)
4 MB	n/a	596892k (78962k)	n/a	7.08 (53)
8 MB	n/a	390612k (78926k)	n/a	19 (107)
16 MB	n/a	392917k (79117k)	n/a	43 (210)
33 MB	n/a	275665k (78849k)	n/a	94 (420)
67 MB	n/a	354996k (79157k)	n/a	250 (850)
134 MB	n/a	338520k (79217k)	n/a	409 (1700)

Table 3: AES versus SPECK, throughput in byte per second, execution time in millisecond, 1024 bytes = 1 Kilobytes(KB)

for AES is about 2-3 times smaller when running exactly identical piece of code on Raspberry Pi. On the other hand, the throughput of SPECK is more than three order of magnitude larger than AES on MacBook Pro, and approximately 2 order of magnitude larger on Pi. So does the execution time, but in the opposite direction. The reduction rate of throughput (time) when changing code running environment from MacBook Pro to Pi is much smaller for SPECK compares to AES. Overall, SPECK is optimised for resource-constrained devices like IoTs.

In Figure 7, there is dramatically increase of throughput with respect to message size with SPECK on both devices (Intel and ARM processors). The increase of throughput with AES cipher can be observed from the table, but rather plateaued in the graph within the magic logarithm. This is due to the robustness of SPECK cipher with increasing message sizes. Since the substitution box are software friendly, in contrast with the bit-permutation

Input size (bytes)	**Memory** (SPECK)	**Memory** (AES)
2 B	0.037 (0.0108)	0.025 (0.0098)
4 B	0.037 (0.0108)	0.026 (0.0098)
8 B	0.037 (0.0108)	0.026 (0.0098)
16 B	0.037 (0.0108)	0.026 (0.0098)
32 B	0.037 (0.0108)	0.026 (0.0098)
64 B	0.037 (0.0108)	0.026 (0.0098)
128 B	0.037 (0.0108)	0.026 (0.0098)
256 B	0.037(0.0108)	0.026 (0.0098)
512 B	0.037(0.0108)	0.026 (0.0098)
1 KB	0.037(0.0108)	0.026 (0.0098)
2 KB	0.037(0.0108)	0.026 (0.0098)
4 KB	0.037(0.0108)	0.026 (0.0098)
8 KB	0.037(0.0108)	0.026 (0.0098)
16 KB	0.037(0.0108)	0.026 (0.0098)
32 KB	0.037(0.010)	0.026 (0.0098)
64 KB	0.037(0.010)	0.026 (0.0098)
128 KB	0.037(0.011)	0.026 (0.012)
256 KB	0.037(0.010)	0.026 (0.013)
512 KB	0.037(0.011)	0.030 (0.017)
1 MB	0.037 (0.011)	n/a
2 MB	0.038 (0.012)	n/a
4 MB	0.038 (0.014)	n/a
8 MB	0.046 (0.018)	n/a
16 MB	0.036 (0.026)	n/a
33 MB	0.0408 (0.042)	n/a
67 MB	0.10002 (0.073)	n/a
134 MB	0.16 (0.13)	n/a

Table 4: AES versus SPECK, RAM in GB

are well-suited in hardware implementation. Also, the modular addition operation is smaller and faster than S-boxes in software, thus the choice of cipher can be decided upon to these factors [3].

In table 5, we assessed the performance of several LW ciphers on both MacBook Pro and Raspberry Pi. There is a negative relationship between throughput and the power consumption of specific cipher. Thus, if the execution time of cipher is short, that indicates less memory consumption of the cipher per bit.

Although CLEFIA-128 has higher throughput than SIMON and SPECK, it is not necessary the most efficient cipher because it is the only cipher that implemented in C on the above table. C is known to be an efficient language. The pure efficiency of specific algorithm has to be examined with the same programming language. The rest of the codes are found either on GitHub or on the *asecuritysite.com* with Python implementations. In the context of execution time and throughput, CLEFIA-128 works more efficient with C implementation than the Python. Though the Python implementation implemented is optimised with multiplication in the binary finite field. The C code size is more compact too, but by no means optimised for speed. The

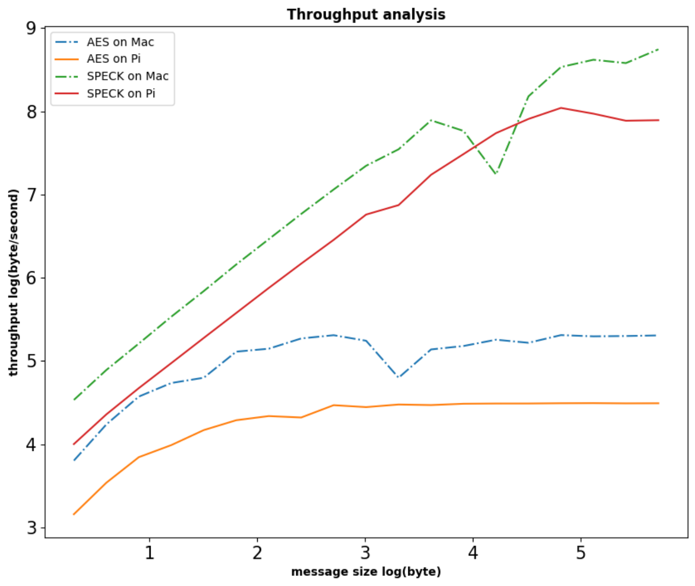


Figure 7: Throughput analysis, SPECK and AES-128

fact that CLEFIA is a 128 bit block cipher and XTEA, Present, Prince, Pride are all from 32 bit to 64 bit blocks should be taken into account too. Considering the extra bit of security CLEFIA offered, it is really performed well against other ciphers. In the point of view of efficient cipher, we compare the throughput (Bytes/Second), time, and memory consumption (GB) directly linked to power consumption. Since the implementation of ciphers are from elsewhere, we do not analyse the size of the implementation and physical footprint in this section. Overall, SPECK and SIMON outperform all other ciphers in Python on the above table in terms of execution time in conjunction with memory, especially on Pi. It is about more than 500 times faster than the Prince cipher on MacBook, and more than 3 order of magnitude (1068 times) faster than Prince on Pi. The stream ciphers such as RC4 and Rabbit are the least fast ciphers (include the key derivation process). The XTEA cipher is also considered as fast cipher and only slower than SPECK and SIMON. However, the C implementation of CLEFIA-128 is the fastest and about 2 times faster than SPECK on Pi. It is also worth mention that when we compile the code with gcc and flag -01 or -03, the average running

	Cipher	MacBook Pro			Raspberry Pi		
	Metric	Time ms	Memory	Throughput	Time ms	Memory	Throughput
1	AES-128	0.21	0.026	37k	1.1	0.0098	7k
2	XTEA	0.066	0.013	121k	0.22	0.01005	36k
3	SIMON	0.0956	0.011	83k	0.26	0.0085	29k
4	SPECK	0.0635	0.013	126k	0.15	0.0104	47k
5	PRESENT	1.4	0.013	5.3k	3.2	0.0104	2.4k
6	SKINNY	2.5	0.011	3.1k	3.4	0.0084	2.2k
7	KATAN	4.09	0.012	1.9k	11	0.00802	0.67k
8	Prince	42	0.012	0.18k	170	0.0095	0.046k
9	Pride	14	0.013	0.55k	30.6	0.011	0.26k
10	CLEFIA-128(C)	0.021	n/a	745k	0.081	n/a	97k
10	CLEFIA-128(Python)	0.57	0.011	13k	1.6	0.0086	4.7k
11	RC4	0.12	0.011	64k	0.28	0.0084	26k
12	Rabbit	0.304	0.013	13k (26k)	0.85	0.0079	8.7k

Table 5: Comparison of lightweight ciphers, 8 bytes message, 16 bytes key, time in millisecond

time over 1000 iterations decreased more than one magnitude. This fact contributes to the configuration of compiler gcc version 8.3.0.

In the context of memory consumption, SIMON, XTEA, SPECK, CLEFIA, and stream cipher RC4, Rabbit are all memory efficient ciphers on MacBook Pro. But on Raspberry Pi, the AES-128 cipher consume way less RAM than it is on MacBook. To summarise, the ARX and SPN based ciphers performed well and achieved overall satisfactory level in term of throughput on Pi, except Prince and Pride.

The CPU usage in percent is not presented in above tables. But it is tested that for SPECK cipher, the CPU usage in percent ($\approx 50\%$) is about 2 times higher on MacBook Pro compares to Raspberry Pi ($\approx 25\%$). Given this fact the power consumption is directly related to the CPU usage, we can conclude that SPECK is more efficient on ARM processor than Intel core.

In [3], the author stated that 160 bits ECC prime fields would be suffice for lightweight embedded device, thus we will use curve 25519 which is certainly secure in the same context. The performance test for asymmetric cipher is conducted differently from the symmetric cipher. It is evident that asymmetric cipher is much slower than symmetric ones in terms of execution time and correlated power consumption [15]. All the asymmetric ciphers are run 10 iterations in contrast with 1000 iterations run by the symmetric ciphers. This is because asymmetric cipher is generally more computational expensive than symmetric counterpart, ten iterations alone is suffice to observe the time differences.

Algorithm	MacBook Pro			Raspberry Pi		
Metric	Clock	CPU	Memory	Clock	CPU	Memory
ELLI	0.0100	0.0099	0.0119	0.0471	0.0470	0.0091
RSA	2.7950	2.7867	0.0147	3.1218	3.1213	0.0122

Table 6: Comparison of lightweight asymmetric ciphers, time in second, memory in GB

In Table 6, clock time and CPU time are counted in seconds, memory is measured in GB. The numbers are rounded to two significant figures for convenient. The table depicts that the light elliptic performs more than 270 times faster than the RSA on Macbook pro with the same level of security (256 bits ECC vs 3072 bit RSA). And 66 times faster on Pi. The difference in memory usage is not that significant, only 23% less memory used between ELLI and RSA on MacBook pro, and 34% less on Pi. Memory-wise, ELLI is more efficient to use on small devices. On the other hand, there is less memory requirement to run ELLI on Pi compared to MacBook Pro. Overall, The result demonstrates that it is almost physically infeasible to implement RSA greater than 1024 bit into RFID tag. The sole reason we decide not to perform the ROM analysis is because of the discrepancy in the codes we employed, are by no means either hardware or software optimum. The software implementation of the same cipher primitive will aim to reduce the code size while increase the throughput, optimally find a balance between efficiency and energy cost. It is often contradicts with hardware implementation which aims to reduce the chip area (GEs).

Regarding to the security level and vulnerability of LW ciphers, there are 3 general criterion adopted in [14] as:

(i) The diffusion level (e.g. P-box) is described as an inversion of an input bit whether in the key or plaintext, will generate a 50 percent or more change in the output bit.

(ii) The confusion level (e.g. S-box) describes the relationship between the plaintext, the key and the ciphertext has to be complex enough.

(iii) The linear and differential cryptanalysis describes these two cryptanalysis are the most two common techniques used to exploit the vulnerabilities of symmetric type ciphers. To say a particular cipher system is not susceptible to these attacks, the complexity of execute these attacks must be greater than the brute force effort of 2^{N_k}, where N_k is the number of bits in key length.

There are mathematical equations which can be used to estimate confusion level and diffusion level namely Hamming distance in [49].

6. Conclusion

The primary focus of this article is to provide a comparative evaluation of the performance of various lightweight symmetric and asymmetric ciphers on the Intel and ARM processor families. The aspiration is to throw out a brick to attract a jade, as in the ancient Chinese proverbs. In the same vein, it is aimed to attracts more competent researchers to open the door to research area of resource-limited devices such as intelligent sensor, IoTs, RFID, micro controllers. The inherited limited capabilities of the resource constrained devices reduced the suitability of the pre-existing conventional symmetric ciphers in their applications.

In the main context, the architecture and attribute of different lightweight ciphers are reviewed through the literature. Also we confirm that the inconsistency between the execution time and memory consumption of ciphers running on different devices, especially IoT device like Raspberry Pi. Results shows superiority of the lightweight crypto compare to the traditional algorithm in context of efficiency and speed, as well as in term of power consumption. All the performance evaluations include the Advanced Encryption Standard (Rijndael) as a benchmark since its popularity across industry and academic. In conclusion, in constrained environment, the SPECK and SIMON type ciphers are the most efficient cipher family in Python implementation with large key size of 128 bits. It thus leads to a future of high performance, ubiquitous and pervasive computing. Furthermore, The C implementation of CLEFIA-128 is more efficient and robust than its python implementation. It is worthwhile to implement various LW ciphers on the protocols like SSL/TLS, to address their strengths and weaknesses during data transmission between client and server devices.

7. Future Direction

Since the IoT devices on battery will have limited energy resources, the energy consumption of the encryption and decryption process in term of

$Joule/second$ needs to be measured for the lightweight schemes. To test the battery consumption of ciphers on Raspberry Pi, mobile devices and Laptop, we can measure the pertinent consumption speed when increasing the message and file sizes.

The throughput can be calculated using average of ten to twenty random generated messages with identical size and low standard deviation. We can do analysis on throughput of any algorithm which can be calculated from the ratio of total size of database divided by total execution time during encryption or decryption process for symmetric ciphers with device clocked at a low frequency in kHz. Also the code sizes can be taken into account [50]. Considering implement the figure of merit (FOM) metric for measuring the energy efficiency of ciphers, $\text{FOM} = \frac{\text{throughput}[Kbps]}{\text{area squared}[GE^2]}$ and or other metrics like balanced performance metric and cycle per bytes [51].

This performance analysis can be conducted with different implementation as well, for instance, python implementation versus C++. A combined survey of lightweight ciphers on applications of security protocol SSL/TLS is sought, where the choice of LW asymmetric cipher with LW symmetric cipher are tested and identified on resource constrained devices. We can achieve such prototype experiment by build a multiple Raspberry Pi servers, and clients.

It should be pointed out that the performance evaluation on other resource constrained devices are required, to form a more holistic view of the LW ciphers. Because there is different criterion for selecting appropriate cipher to cater the specific systems. In foreseeable future, we can conduct linear crypanalysis to test the robustness of the existing lightweight ciphers. Another possible research area is the lightweight authenticated encryption and the LW zero-knowledge proofs. At last but not the least, it remains to be seen whether the mathematical Big O notation can be used as metric for the computational complexity measurement of ciphers in time and space.

8. Acknowledgement

First and foremost, my heartfelt gratitude towards Professor William Buchanan OBE. As someone who never studied computer science nor cyber security that wishes to have a career transition, he encouraged me to strive my best, answering my dumb questions 24/7 on Slack. This small piece of work could not be done without his support.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 September 2020 doi:10.20944/preprints202009.0302.v1

9. Appendix

SPECK cipher throughput value:

```
# from https:// asecuritysite .com/encryption/speck
import speck
import numpy as np
import binascii
import sys
import time, timeit
import os, random, string, psutil

def memory_usage_percent():
    # measure the memory usage
    proc = psutil.Process(os.getpid())
    memory = proc.memory_percent()
    return memory

def memory_usage_mb():
    # measure the memory usage in MB
    import psutil
    proc = psutil.Process(os.getpid())
    memory = proc.memory_info()[0] / float(2 ** 20)
    return memory

def randomString(stringLength=10):
    """Generate a random string of fixed length """
     letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(stringLength))

mess = os.urandom(8)
## 128 bits key length in hex, and block size 64 bits.
k='0x1b1a1918131211100b0a090803020100'

def getBinary(word):
    return int(binascii.hexlify(word), 16)

if (len(sys.argv)>1):
   mess=str(sys.argv[1])
   m=getBinary(mess)

if (len(sys.argv)>2):
   k=str(sys.argv[2])

key=int(k,16)

```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 September 2020 doi:10.20944/preprints202009.0302.v1

```
print ("Message:\t",mess)
print ("Key:\t\t",k)

ksize=(len(k)-2)*4

bsize=32
if (ksize==72): bsize=48
if (ksize==96): bsize=48
if (ksize==128): bsize=64

print ("Key size:\t", ksize)
print ("Block size:\t", bsize)

## 128 bits key length
w = speck.SpeckCipher(key, key_size=ksize, block_size=bsize)
def my_function_SPECK(mess):
    t = w.encrypt(int.from_bytes(mess.encode(), byteorder='big'))
    res = w.decrypt(t)
    return 0

counter = 0
print("(encryption+decryption) for increasing message size")
print("from 2 bytes to 2048*2**13 bytes")
if (1==1):
     list_mess = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
     2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
     2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
     2048*2**14, 2048*2**15, 2048*2**16, 2048*2**17]
     # list = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
     # 2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
     # 2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
     # 2048*2**14, 2048*2**15, 2048*2**16, 2048*2**17, 2048*2**18,
     # 2048*2**19, 2048*2**20, 2048*2**21, 2048*2**22,2048*2**23,
     # 2048*2**24, 2048*2**25, 2048*2**26, 2048*2**27, 2048*2**28,
     # 2048*2**29, 2048*2**30]
     print("Cipher throughput test speck:")
     for bit in list_mess:
          print(" message size in byte", bit)
          mess = os.urandom(bit)
          start_time = time.time()
          my_function_SPECK(mess)
          print("--- %s throughput bytes/second ---" %(np.divide(bit,
          (time.time() - start_time))))
```

SPECK cipher test vectors:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 September 2020 doi:10.20944/preprints202009.0302.v1

```
# from https:// asecuritysite .com/encryption/speck
import speck
import random
import binascii
import sys
import time
import timeit
import os, random, string, psutil

def memory_usage_percent():
    # measure the memory usage
    proc = psutil.Process(os.getpid())
    memory = proc.memory_percent()
    return memory

def memory_usage_mb():
    # measure the memory usage in MB
    import psutil
    proc = psutil.Process(os.getpid())
    memory = proc.memory_info()[0] / float(2 ** 20)
    return memory

def getBinary(word):
    return int(binascii. hexlify (word), 16)

## The first 5 cases are valid, the last two cases are outside of
## the range of block size for our test code.
mess_stack = ['6574694c', '20796c6c6172', '6d2073696874', '74614620736e6165',
'3b7265747475432d', '65776f68202c656761737520', '6c617669757165207469206564616d20'
    ]

key_stack = ['0x1918111009080100', '0x1211100a0908020100', '0
    x1a19181211100a0908020100',
'0x131211100b0a090803020100', '0x1b1a1918131211100b0a090803020100',
'0x0d0c0b0a0908050403020100', '0x0f0e0d0c0b0a09080706050403020100']

cipher_stack = ['a86842f2', 'c049a5385adc', '735e10b6445d', '9f7952ec4175946c',
'8c6fa548454e028b', '9e4d09ab717862bdde8f79aa', 'a65d9851797832657860fedf5c570d18']
for i in range(7):
    mess = bytes.fromhex(mess_stack[i]).decode('utf-8') # mess = '6574694c'.decode("
        hex")
#mess='etiL'

## 128 bits key length in hex, and block size 64 bits.
    k = key_stack[i]
    key=int(k,16)

    print ("Message:\t",mess)
    print ("Key:\t\t",k)

    ksize=(len(k)-2)*4

    bsize=32
    if (ksize==72): bsize=48
    if (ksize==96): bsize=48
    if (ksize==128): bsize=64

    print ("Key size:\t",ksize)
    print ("Block size:\t",bsize)

    w = speck.SpeckCipher(key, key_size=ksize, block_size=bsize)
    t = w.encrypt(int.from_bytes(mess.encode(), byteorder='big'))

    print ("Encrypted:\t",hex(t))
    if('0x'+cipher_stack[i]==hex(t)):
        print("True, test cipher text supposed to be:", hex(t))

        res = w.decrypt(t)

        hexstr= hex(res)
        print ("Decrypt:\t",hexstr)

        res_str=bytes.fromhex(hexstr[2:]).decode('utf-8')
        print ("Decrypt:\t",res_str)
```

SPECK cipher capacity test:

```

# from https://asecuritysite.com/encryption/speck
import speck
import binascii
import time, timeit
import os, sys, random, string, psutil

def memory_usage_percent():
    # measure the memory usage
    proc = psutil.Process(os.getpid())
    memory = proc.memory_percent()
    return memory

def memory_usage_mb():
    # measure the memory usage in MB
    import psutil
    proc = psutil.Process(os.getpid())
    memory = proc.memory_info()[0] / float(2 ** 20)
    return memory

def randomString(stringLength=10):
    """Generate a random string of fixed length """
     letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(stringLength))
mess = randomString(8)
#mess='hello123'

## 128 bits key length in hex, and block size 64 bits.
k='0x1b1a1918131211100b0a090803020100'

def getBinary(word):
    return int(binascii.hexlify(word), 16)

if (len(sys.argv)>1):
   mess=str(sys.argv[1])
   m=getBinary(mess)

if (len(sys.argv)>2):
   k=str(sys.argv[2])

key=int(k,16)

print ("Message:\t",mess)
print ("Key:\t\t",k)

ksize=(len(k)-2)*4

bsize=32
if (ksize==72): bsize=48
if (ksize==96): bsize=48
if (ksize==128): bsize=64

print ("Key size:\t",ksize)
print ("Block size:\t",bsize)

## 128 bits key length
def my_function_SPECK(mess):
   w = speck.SpeckCipher(key, key_size=ksize, block_size=bsize)
   t = w.encrypt(int.from_bytes(mess.encode(), byteorder='big'))
```

```
    res  = w.decrypt(t)
    #hexstr= hex(res)
    return 0

clock_time_sum = 0
cpu_time_sum = 0
for i in range(1000):
     start_time_c = time.time()
     start_time = time.process_time()
     my_function_SPECK(mess)
     cpu_time_sum += (time.process_time() - start_time)
     clock_time_sum += (time.time() - start_time_c)
cpu_time_sum = cpu_time_sum/1000
clock_time_sum = clock_time_sum/1000
print("Average clock time over 1000 rounds = ", clock_time_sum)
print("Average CPU time over 1000 iterations:", cpu_time_sum)

counter = 0
print("(encryption+decryption) for increasing message size")
print("from 2 bytes to 2048*2**16 bytes")

''' time and memory consumption against input message sizes in bytes'''
if (1==1):
     list  = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
     2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
     2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
     2048*2**14, 2048*2**15, 2048*2**16]
     print("Cipher capacity test speck (time):")
     for byte in list :
          counter += 1
          print("  ", counter)
          mess = randomString(byte)
          start_time = time.time()
          my_function_SPECK(mess)
          print("--- %s seconds ---" % (time.time() - start_time))

if (1==1):
     list  = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
     2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
     2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
     2048*2**14, 2048*2**15, 2048*2**16]
     print("Cipher capacity test speck (memory):")
     for byte in list :
          mess = randomString(byte)
          my_function_SPECK(mess)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 September 2020 doi:10.20944/preprints202009.0302.v1

```
        pid = os.getpid()
        py = psutil.Process(pid)
        memoryUse = py.memory_info()[0]/2.**30
        print('memory use GB:', memoryUse)
```

RC4 stream cipher execution time test:

```
## https://github.com/DavidBuchanan314/rc4
from RC4 import RC4
import memory_mana
import os, psutil, time, timeit

def memory_usage_mb():
    # measure the memory usage in MB
    import psutil
    proc = psutil.Process(os.getpid())
    memory = proc.memory_info()[0] / float(2 ** 20)
    return memory

cipher = RC4(b"secret", streaming=False)
msg = b"I love Scotland"

ciphertext = cipher.crypt(msg)
print(ciphertext) #=> b'\xa4\x16\xbes\xf4\xc1\xf6\xf5Q\xa4\xcf\xb0\x92\x88\x91'

plaintext = cipher.crypt(ciphertext)
print(plaintext) #=> b"I love Scotland"
assert(plaintext == msg)

mess = os.urandom(8) # 64 bit message
passPhrase = os.urandom(16) # 128 bits key

def my_function_rc4(mess, passPhrase):
    cipher = RC4(passPhrase, streaming=False)
    ciphertext = cipher.crypt(mess)
    plaintext = cipher.crypt(ciphertext)
    return 0


print("RAM usage MB:",def memory_usage_mb())
print("CPU usage percenage:",psutil.cpu_percent())
start_time_1 = time.time()
start_time = time.process_time()
my_function_rc4(mess, passPhrase)
print("Excution time--- %s seconds ---" % (time.time() - start_time_1))
print("CPU time --- %s seconds ---" % (time.process_time() - start_time))
print("CPU percentage:", psutil.cpu_percent())
print("RAM usage MB:", memory_usage_mb())


memo_sum = 0
for i in range(1000):
    my_function_rc4(mess, passPhrase)
    pid = os.getpid()
    py = psutil.Process(pid)
    memo_sum += py.memory_info()[0]/2.**30 # memory use in GB
memo_sum = memo_sum/1000
print("Average memory over 1000 iterations = ", memo_sum,"GB")


clock_time_sum = 0
cpu_time_sum = 0
for i in range(1000):
    start_time_1 = time.time()
    start_time = time.process_time()
    my_function_rc4(mess, passPhrase)
    cpu_time_sum += (time.process_time() - start_time)
    clock_time_sum += (time.time() - start_time_1)
cpu_time_sum = cpu_time_sum/1000
clock_time_sum = clock_time_sum/1000
print("Average clock time over 1000 iterations  = ", clock_time_sum)
print("Average CPU time over 1000 iterations = ", cpu_time_sum)


counter = 0
print("Cipher capacity test RC4 (execution time):")
print("(encryption+decryption) with increasing message size")
print("from 2 bytes to 2048*2**16 bytes")
if (1==1):
    list = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
    2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
    2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
    2048*2**14, 2048*2**15, 2048*2**16]
    for bit in list:
        counter += 1
        print("  ", counter)
        mess = os.urandom(bit)
        start_time = time.time()
        my_function_rc4(mess, passPhrase)
        print("--- %s time seconds ---" % (time.time() - start_time))


print("Cipher capacity test mode RC4 (memory):")
if (1==1):
    list = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048,2048*2,
    2048*2**2,2048*2**3,2048*2**4,2048*2**5,2048*2**6,2048*2**7,
    2048*2**8,2048*2**9,2048*2**10,2048*2**11,2048*2**12,2048*2**13,
    2048*2**14, 2048*2**15, 2048*2**16]
    for bit in list:
        mess = os.urandom(bit)
        my_function_rc4(mess, passPhrase)
        pid = os.getpid()
        py = psutil.Process(pid)
        memoryUse = py.memory_info()[0]/2.**30
        print('memory use GB:', memoryUse)
```

10. Power and Energy Consumption

The role of energy and power consumption in the context of IoTs can not be stressed enough. It dictates the size of devices and thereby the costs of manufacturing them. Also its security level against various side-channel attacks. The measurement unit is either Joules or Watt, which can be calculated from current in ampere (A), voltage (V) and time (S). We aim to dip our toe in the water by confirm the negative relationship between throughput and power consumption. Generally speaking, hardware AES performs better on a battery powered device than software implemented AES [52]. In [53], the authors proved that the key schedule process consumes more energy in transmission protocol than the actual encryption/decryption processes for both the symmetric and asymmetric ciphers. In addition, the energy consumption of asymmetric cipher is more depend on the key size than their symmetric counterpart. The power consumption changed by executing only the designated cipher on the Raspberry Pi with CPU frequency at 600 MHz. If of interest, the energy consumption can be calculated from the execution time and power consumption value. Given the current I_t and voltage V_t over time, one can calculate the approximate of energy consumption as:

$$E = \int_0^t V_t I_t dt \approx \sum_0^n V_t I_t \Delta t \tag{2}$$

Where Δt is the time interval over n discrete samples [54]. Since the above equation of energy consumption depends on the time interval, hence the cipher with more execution time would consume more energy in general. It is thus vital to identify the robust cipher that does not susceptible to the change of input data size and other parameters like modulus N and prime P. In this example, we have the constant power supply according to Raspberry Pi power adaptor as 5 voltages at 3 amperes via USB port. Testing device is KEWEISI USB voltage current meter. Although WiFi is in use for display monitor, we take into account of WiFi consumption into the IDLE state and all other programs and USB port are put on halt. The average current drawn by the ARM processor while running the ciphers over 10000 times. The current consumption for (encryption+decryption) is measured for AES, SPECK and CLEFIA. The mean average of 32 samples of power consumption are recorded. From the Table, we can argue that with ten

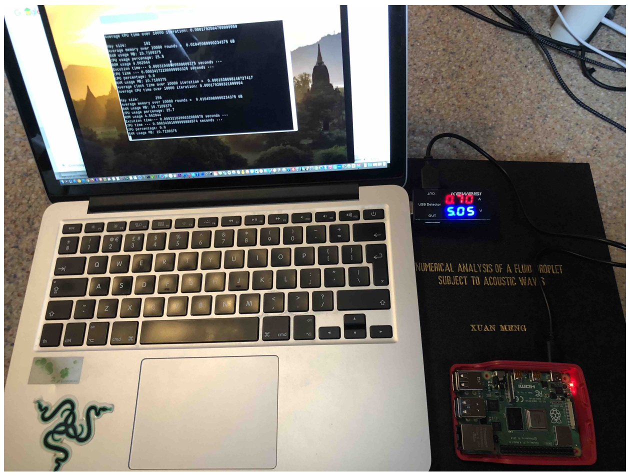

KEWEISI
0.70
5.05
NUMERICAL ANALYSIS OF A FLUID DROPLET SUBJECT TO ACOUSTIC WAVES
XUAN MENG

Figure 8: Experiment setup

Cipher	**Enc/Dec**	**Enc**	**Dec**
AES:	3.41 W, 680 mA	3.16 W, 630 mA	3.06 W, 610 mA
SPECK:	3.01 W, 600 mA	2.81 W, 560 mA	2.86 W, 570 mA
CLEFIA:	3.51 W, 700 mA	3.36 W, 670 mA	3.41 W, 680 mA

Table 7: Power consumption, the Pi 4B uses 520 mA (2.61 Watts) at 5.02V, in idle state.

thousands running of both SPECK, CLEFIA and AES cipher with block size 128 bit, key size 128 bit and 8 bytes input data. The SPECK cipher consumes less energy at its peak current usage when compare to AES cipher. Surprisingly, the CLEFIA cipher consumes the most amount of energy when compare to others. The energy for encryption and decryption are almost identical contributed to the fact that, the latter is the reverse of or very similar in structure to the former. This coincides with the observation made in [55], namely uniform average power dissipation. Overall, the SPECK is the most power efficient block cipher combined with the results obtained from execution time comparison.

In the second experiment, we can use an external battery with Raspberry Pi, drain the battery without execute any program in Pi and count the time required to drain the battery completely. Then, we recharge the battery and repeatedly run the designated cipher code on Pi until the battery is empty. Hence, we can divide the total time required to drain the battery by the total battery energy to derive the usage percentage.

[1] N. A. Gunathilake, W. J. Buchanan, and R. Asif, "Next generation lightweight cryptography for smart iot devices:: implementation, challenges and applications," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019, pp. 707–710.

[2] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Performance comparison of the aes submissions," in *Second AES Candidate Conference*, 1999.

[3] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design &amp; Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[4] R. Roman, C. Alcaraz, and J. Lopez, "A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 231–244, 2007.

[5] C. Paar, A. Poschmann, and M. Robshaw, "New designs in lightweight symmetric encryption," in *RFID Security*. Springer, 2008, pp. 349–371.

[6] Z. Gong, S. Nikova, and Y. W. Law, "Klein: a new family of lightweight block ciphers," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2011, pp. 1–18.

[7] H. Kader and M. Hadhoud, "Performance evaluation of symmetric encryption algorithms," *Performance Evaluation*, pp. 58–64, 2009.

[8] A. Anjali and K. Saibal, "A survey of cryptanalysis attacks on lightweight block ciphers," *IRACST International Journal of Computer Science and Information &amp; Security (IJCSITS)*, vol. 2, no. 2, p. 2012, 2012.

[9] H. Seo and H. Kim, "Implementation of lightweight cryptographic algorithm for internet of things," *Rev KIISC*, vol. 25, pp. 12–19, 2015.

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

[11] E. Iosifidis and K. Limniotis, "A study of lightweight block ciphers in tls: The case of speck," in *Proceedings of the 20th Pan-Hellenic Conference on Informatics*, 2016, pp. 1–5.

[12] A. Biryukov and L. P. Perrin, "State of the art in lightweight symmetric cryptography," 2017.

[13] W. J. Buchanan, S. Li, and R. Asif, "Lightweight cryptography methods," *Journal of Cyber Security Technology*, vol. 1, no. 3-4, pp. 187–201, 2017.

[14] T. Omrani, R. Rhouma, and L. Sliman, "Lightweight cryptography for resource-constrained devices: a comparative study and rectangle cryptanalysis," in *International Conference on Digital Economy*. Springer, 2018, pp. 107–118.

[15] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight cryptography for embedded systems–a comparative analysis," in *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2013, pp. 333–349.

[16] J. Jeong, L. Bajracharya, and M. Hwang, "Optimal lightweight cryptography algorithm for environmental monitoring service based on iot," in *International Conference on Information Science and Applications*. Springer, 2018, pp. 361–367.

[17] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[18] E. Bertino, F. Paci, R. Ferrini, and N. Shang, "Privacy-preserving digital identity management for cloud computing." *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 21–27, 2009.

[19] E. I. S. Authority, "Roca vulnerability and eid: Lessons learned," pp. https://www.ria.ee/sites/default/files/content–editors/kuberturve/roca–vulnerability–and–eid–lessons–learned.pdf, 2018.

[20] D. J. Rani and S. E. Roslin, "Light weight cryptographic algorithms for medical internet of things (iot)-a review," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*. IEEE, 2016, pp. 1–6.

[21] J.-P. Aumasson, "Too much crypto," *Real-World Crypto 2020*, Dec 2019.

[22] A. R. Choudhuri and S. Maitra, "Significantly improved multi-bit differentials for reduced round salsa and chacha," *IACR Transactions on Symmetric Cryptology*, pp. 261–287, 2016.

[23] S. A. Hirani, "Energy consumption of encryption schemes in wireless devices," Ph.D. dissertation, University of Pittsburgh, 2003.

[24] C. Tezcan, "The improbable differential attack: Cryptanalysis of reduced round clefia," in *International Conference on Cryptology in India*. Springer, 2010, pp. 197–209.

[25] Y. Li, W. Wu, and L. Zhang, "Improved integral attacks on reduced-round clefia block cipher," in *International Workshop on Information Security Applications*. Springer, 2011, pp. 28–39.

[26] P. N. Patil, "A comparative analysis of raspberry pi hardware with ar-duino phidgets beaglebone black and udoo," *International Research Journal of Engineering and Technology (IRJET)*, pp. 1595–1600, 2016.

[27] P. Derbez, P.-A. Fouque, and J. Jean, "Improved key recovery attacks on reduced-round aes in the single-key setting," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 371–387.

doi:10.20944/preprints202009.0302.v1

[28] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2011, pp. 344–371.

[29] O. Lo, W. J. Buchanan, and D. Carson, "Correlation power analysis on the present block cipher on an embedded device," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1–6.

[30] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger *et al.*, "Prince–a low-latency block cipher for pervasive computing applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2012, pp. 208–225.

[31] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers." *IACR Cryptology ePrint Archive*, vol. 2013, no. 1, pp. 404–449, 2013.

[32] ——, "Simon and speck: Block ciphers for the internet of things." *IACR Cryptology ePrint Archive*, vol. 2015, p. 585, 2015.

[33] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher clefia," in *International workshop on fast software encryption*. Springer, 2007, pp. 181–195.

[34] M. Katagi, S. Moriai *et al.*, "Lightweight cryptography for the internet of things," *Sony Corporation*, vol. 2008, pp. 7–10, 2008.

[35] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius, "Rabbit: A new high-performance stream cipher," in *International Workshop on Fast Software Encryption*. Springer, 2003, pp. 307–329.

[36] M. Boesgaard, M. Vesterager, T. Christensen, and E. Zenner, "The stream cipher rabbit," *ECRYPT Stream Cipher Project Report*, vol. 6, p. 28, 2005.

[37] C. De Canniere, O. Dunkelman, and M. Knežević, "Katan and ktantan—a family of small and efficient hardware-oriented block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 272–288.

[38] B. Zhu and G. Gong, "Multidimensional meet-in-the-middle attack and its applications to katan32/48/64," *Cryptography and Communications*, vol. 6, no. 4, pp. 313–333, 2014.

[39] E. T. K. L. Thorsten, "Kleinjung."universal security; from bits and mips to pools, lakes–and beyond"," 2013.

[40] I. Damgård, N. Fazio, and A. Nicolosi, "Non-interactive zero-knowledge from homomorphic encryption," in *Theory of Cryptography Conference*. Springer, 2006, pp. 41–59.

[41] S. S. Kumar, "Elliptic curve cryptography for constrained devices," Ph.D. dissertation, Verlag nicht ermittelbar, 2006.

[42] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpus," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 119–132.

[43] M. Braun, E. Hess, and B. Meyer, "Using elliptic curves on rfid tags," *International Journal of Computer Science and Network Security*, vol. 2, pp. 1–9, 2008.

[44] O. Barahtian, M. Cuciuc, L. Petcana, C. Leordeanu, and V. Cristea, "Evaluation of lightweight block ciphers for embedded systems," in *International Conference for Information Technology and Communications*. Springer, 2015, pp. 49–58.

[45] S. S. Gupta, A. Chattopadhyay, K. Sinha, S. Maitra, and B. P. Sinha, "High-performance hardware implementation for rc4 stream cipher," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 730–743, 2012.

[46] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of aes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 69–88.

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

[48] N.-F. Standard, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.

[49] H. Hu, L. Zhang, and J. Wu, "Hamming distance based approximate similarity text search algorithm," in *2015 Seventh International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2015, pp. 1–6.

[50] A. B. Ahmed and A. B. Abdallah, "Architecture and design of high-throughput, low-latency, and fault-tolerant routing algorithm for 3d-network-on-chip (3d-noc)," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1507–1532, 2013.

[51] A. Shoufan, T. Wink, G. Molter, S. Huss, and F. Strentzke, "A novel processor architecture for mceliece cryptosystem and fpga platforms," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2009, pp. 98–105.

[52] C.-W. Hung and W.-T. Hsu, "Power consumption and calculation requirement analysis of aes for wsn iot," *Sensors*, vol. 18, no. 6, p. 1675, 2018.

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

[54] T. Nie, L. Zhou, and Z.-M. Lu, "Power evaluation methods for data encryption algorithms," *IET software*, vol. 8, no. 1, pp. 12–18, 2014.

[55] A. Sinha and A. P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 220–225.