Naif Arab University for Security Sciences
Journal of Information Security & Cybercrimes Research
مجلة بحوث أمن المعلومات والجرائم السيبرانية
https://journals.nauss.edu.sa/index.php/JISCR

JISCR

Software Optimisation of Lightweight Klein Encryption in the Internet of Things

Seyed R. Ghorashi[1,2*], Tanveer Zia[3], Yinhao Jiang[1,2], and Michael Bewong[1]

[1]Charles Sturt University, NSW, Australia
[2]Cyber Security Cooperative Research Centre, Joondalup. WA, Australia
[3] Center of Cybercrimes and Digital forensics, Naif Arab University for Security Sciences, Riyadh, Saudi Arabia.

Received 12 Oct. 2021; Accepted 15 Dec. 2021; Available Online 30 Dec. 2021

Abstract

The Internet of Things (IoT) and Wireless Sensor Network (WSN) devices are prone to security vulnerabilities, especially when they are resource constrained. Lightweight cryptography is a promising encryption concept for IoT and WSN devices, that can mitigate these vulnerabilities. For example, Klein encryption is a lightweight block cipher, which has achieved popularity for the trade-off between performance and security. In this paper, we propose one novel method to enhance the efficiency of the Klein block cipher and the effects on the Central Processing Unit (CPU), memory usage, and processing time. Furthermore, we evaluated another approach on the performance of the Klein encryption iterations. These approaches were implemented in the Python language and ran on the Raspberry PI 3. We evaluated and analysed the results of two modified encryption algorithms and confirm that two enhancing techniques lead to significantly improved performance compared to the original algorithm.

I. Introduction

The fast development of wireless communication technologies calls for smarter devices such as Wireless Sensor Network (WSN) and embedded systems with lower power consumption [1]. These technologies enable sensors to communicate and transfer data over short distances. As the development of WSN devices improves, these sensors' applications and reliability grow. For example, Radio Frequency Identification (RFID) tags can operate at longer ranges, temperature and motion sensors have higher accuracy readings. [2]. The applications of these technologies are highly valued and are used in fields such as agriculture, the military, and healthcare. A big advantage of the sensors is the low-cost budget required for implementation and maintenance. However, these advantages have drawn much attention to the sensor's security and vulnerabilities. Generally speaking, the components of WSN devices can only offer constrained resources such as limited Central Processing Units (CPU), low memory, and small battery packs [3]. Thus, sensor devices have been

Keywords: Information security, Internet of Things (IoT), Klein block cipher, Wireless Sensor Networks (WSN), Lightweight cryptography, Software optimisation.

Production and hosting by NAUSS

* Corresponding Author: Seyed Ramin Ghorashi
Email: sghorashi@csu.edu.au

doi: 10.26735/PXAE9280

challenged by malicious adversaries exploiting security vulnerabilities. The hardware constraints put limitations on the approaches for security protection. This problem has prompted academics and industry players to develop security schemes for these devices [4], [5], [6].

A well-known security approach for protecting the WSN devices is called cryptography. Many cryptography schemes have been developed to protect data [7], [8]. Lightweight cryptography comprises a family of cryptography schemes that offer strong security metrics but with less computation complexity, making it suitable for devices with limited resources [9]. The new proposed Klein block cipher is targeted for the Internet of Things (IoT) and WSN. The Klein encryption is a symmetric cryptography that has three versions, Klein-64, Klein-80, and Klein-96 [10], [11]. In this study, we focus on improving the performance of Klein-64. However, we expect that our approaches can be performed on other versions of Klein and produce consistent results. Our work evaluates an implementation of Klein-64 with different methodologies and tries to optimise it with novel modification.

The Klein block cipher is optimised for WSN devices which contain less hardware resources. For security vulnerabilities, this becomes a trade-off between security and performance. In our work we analyse the Klein block cipher and propose two approaches which improve the performance of the Klein. The contribution of this study are as follows:

- The first approach is to replace the algorithm with most resource consumption with an alternative algorithm that uses less resources of the hardware.
- The second approach analyses different iterations of the Klein block cipher to investigate the efficient iteration for performance and security. Next, we demonstrate the effectiveness of our approach by conducting extensive empirical evaluation on 3 S-box and increased iteration.

The rest of the paper is structured as follows. Section II discusses related work and explains the background of the Klein block cipher and current attacks; Section III describes our approaches and the experiments relating to the optimisation of the Klein block cipher; Section IV presents the results of our experiments followed by discussion and future work in Section V and the conclusion is outlined in Section VI.

II. Background

In this section, we examine the development of the Klein encryption scheme and the cryptanalysis on the Klein block cipher.

A. Related Work

With the advanced usage of WSN devices, many lightweight cryptography schemes have been developed and implemented on constrained devices. However, it is widely important to consider the effect of the lightweight cryptography on the devices as they are limited to processing, memory, and power resource. It is then important to evaluate the footprint of the lightweight cryptography schemes to analyse the burden of each cycle and their effect on the devices.

The authors of [12] proposed a new lightweight block cipher, TWINE which aims to achieve hardware and software efficiency. Their experiments involved software performance of TWINE encryption and decryption. The results show significant performance used either on micro controllers or high-end CPUs.

In another study, the authors of [13] described an ultra-lightweight block cipher, PRESENT. Their contribution was to explain the operation of the block cipher and its efficiency on security and hardware. They conducted hardware experiments on PRESENT and compared the results with other lightweight block ciphers.

The operation of PRESENT is slightly different to other lightweight block ciphers such as Advance Encryption Standard (AES), Light Encryption Device (LED) or Klein [14]. This is because AES, LED and Klein use diffusion on their last algorithm in the cycle where PRESENT uses confusion. This also can be true for TWINE [15].

As many lightweight block ciphers were developed for constrained devices, not enough block

ciphers were analysed for their performance efficiencies on limited resource devices. As Klein's scheme is similar to the AES's scheme and for its unique advantage of hardware and software capabilities on constrained devices, there has been a limited research on each algorithm of Klein block cipher. A summary of the related work gap analysis is discussed in Table I.

TABLE I
RESEARCH GAP ANALYSIS

Ref	Contribution	Gap Analysis
[12]	Software performance	The experiment was conducted on type-2 generalised Feistel-permutation networks (FPN) whilst many algorithms are also designed on Substitution-permutation networks.
[13]	Security and hardware	The design of algorithm's operation and schemes can impact the experiment results. The hardware performance of PRESENT on a microcontroller was significant however, similar experiments are required for software performance for both the operation and the design of the scheme.
[13],[14]	Diffusion and Confusion	The diffusion operation shown to be more impactful on the resources than the diffusion, nonetheless further study is required on algorithms with diffusion operation.

B. Related Work and Our Contribution

As some lightweight cryptography schemes are developed and experimented for their hardware, software and security features, many cryptographic schemes developed for microcontrollers are not studied for their software or hardware performances.

In the related work section, lightweight and ultra-lightweight block ciphers were designed for microcontrollers, and in their respected studies, the block ciphers were analysed for their performance. These block ciphers were usually from a FPN family or have different structure of operation that allows them to perform better. Our contribution is to experiment a block cipher that is also designed for microcontrollers that has higher throughput in their operation such as AES. Klein block cipher as lightweight cryptography scheme shows similar structure of operation in its scheme. The last operation of Klein block cipher is a diffusion operation which uses higher performance to achieve its task.

As PRESENT is ultra-lightweight, it is a Substitution-permutation Network (SPN) cipher that has opted a higher number of rounds. The authors of [13] describe that with such higher number of rounds, the cipher becomes more secure against round differential characteristics attack. Our contribution is to experiment Klein block cipher because it is from SPN family, has a lower number of rounds per cycle and it is not immune to differential characteristics attack. Our study will experiment the increase of Klein block cipher rounds for the effect of its software performance on constraint device.

C. Klein Block Cipher Overview

The Klein encryption scheme was proposed by Gong et al. [6] in 2011. The Klein block cipher is a family of Substitution Permutation Network (SPN) ciphers with variable key sizes of 64, 80, 96-bits, and 12, 16, and 20 iterations, respectively. Since then, much cryptanalysis has been performed. There have been reports that the keys have been successfully exploited via iterative differential characteristics analysis and the parallel cut meet-in-the-middle attack [16]. The structure of the Klein block cipher is very similar to AES. This is because the original authors of Klein encryption acquired the algorithms from AES [17]. Some specifications of Klein, however, are different to AES, for example, the secret key size and the block size. The operation of Klein and AES are therefore, the same. However, their performance has a huge impact on their operations with their slight differences. The Klein block cipher has four algorithms per round *AddRoundKey*, *SubNibble*, *RotateNibble*, and *MixNibble*, which are briefly described as follows:

- *AddRoundKey* - The first set of data encryption is offered with a combination of plaintext and the key using XOR operation.
- *SubNibble* - Using the non-linear 4-bit S-box, the linear data array is transformed by the bitwise operation. A difference between AES as it uses byte-wise operation.
- *RotateNibble* - In the *RotateNibble* algorithm, the state is rotated four nibbles, (two bytes) to the left per each round.
- *MixNibble* - *MixNibble* algorithm multiplies the columns of the data array by a modular polynomial equation.

The last algorithm is an adaptation of the Advanced Encryption System's (AES) MixColumn algorithm. The involutive 4 bit S-box and Rijndael's MixColumn allow the Klein block cipher to operate with a low memory requirement. This is an improvement in efficiency that can affect both software and hardware implementations. A representation of one round of the Klein block cipher is shown in Fig. 1.

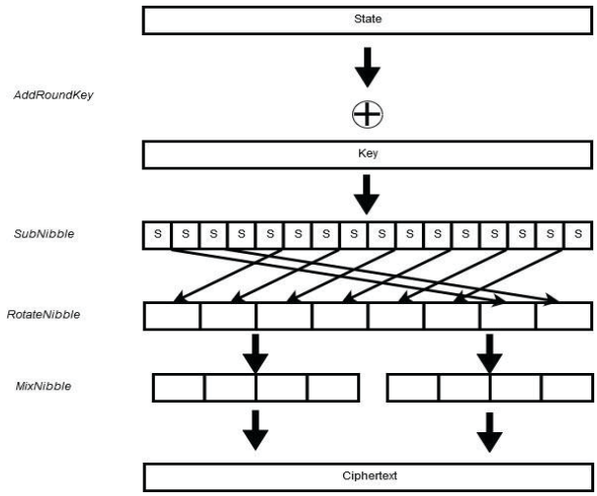

Fig. 1 One Cycle of Klein Block Cipher.

C. Threats to Klein

Security analysis is important for cryptographic schemes, especially in the case of lightweight encryptions, which aim to achieve security with less computational overload, such as the Klein block cipher [18]. This section provides an overview of two security vulnerabilities in the Klein encryption and brief analysis of the application of the threats to the Klein block cipher.

Parallel-Cut Meet-In-The-Middle (PCMITM) Attack: Security analyses are critically important for cryptography algorithms to observe and analyse any security holes in the algorithms. The PCMITM attack application to the Klein block cipher could succeed if *AddRoundKey*, *SubNibble*, and *RotateNibble* have diffusion between the higher and lower nibbles within the bytes of the state. Also, in order to perform a PCMITM attack, there must be a nibble separation. In the *MixNibble* algorithm, the higher and lower nibbles are mixed, and when it is done, it produces a diffusion rate. Therefore, for a PCMITM attack to be performed, it must compute the higher and lower nibbles [18], [19], [20], [21]. The results of a PCMITM attack on the Klein block cipher are given in the following form: The attack for Klein-64 was achieved in 10 rounds out of 12. The parameter of the attack is k = 64, the number of rounds is r = 10, and the time is t = 0 seconds. The attack that was performed in 10 rounds was achieved in T(64,10,0) = 262 seconds and used M(64,10,0) = 260 of memory [18].

Differential Characteristics Attack: A differential characteristics attack is another popular cryptanalysis, which exploits the algorithm's key, first by identifying the input and output differences of a round. One technique that differential characteristics use is to identify the differences by conditioning the input and output and the path it followed to reach them. Once the differential of six rounds is satisfied, then the exploitation of the last *MixNibble* and *RotateNibble* can be determined after the last *SubNibble*. This is usually at the seventh round, at which point the differences should be zero for higher nibbles.

The attack performs on lower nibbles and shifts them through the S-box. The outcome will be reversed through the MixColumn. Once the results are produced, the attack can be considered feasible if the lower nibbles are active. To reduce the key recovery costs, the trials can be reduced 258 times; the attacks always appear to work within the number of trials [5], [22], [23], [24]. Key recovery in six rounds was achieved in 10 seconds. In seven rounds, it was 296 seconds, and in eight rounds, it was 1,344 seconds. Although the first test was

quick, the authors state that the first test was mainly by brute force using neutral bits. They also specified that their program was sped up due to using an Athlon64 X2 Dual-Core 4400+ microprocessor and an 8-bit lookup table for *SubNibble*

III. Optimisation Approaches and Experiments

This section is divided into three parts. The first part explains the details of the experiment environments, the second and the third part explains the details of the approaches. Similar approaches have been reported before for other symmetric encryption ciphers; however, these approaches have not been experimented with on the Klein block cipher [14], [25], [26]. In this study, two approaches were selected to optimise the performance of Klein encryption. It is important to note that Klein encryption is a lightweight block cipher and must remain lightweight throughout the experiments.

A. Experiment Environment

In this study, the Klein block cipher is mainly used and tested on two devices. The first device was a Windows 10 laptop environment with an Intel i7, dual-core CPU with a clock rate of 2.4 Gigahertz (GHz) and 8 Gigabyte (GB) of memory with 1600 MHz memory speed. The second device was a Raspberry Pi Model 3 with a Broad-com BCM2711 quad-core CPU with a clock speed of 1.5 GHz and 1 GB of memory with 900 MHz of memory speed. The information of the technologies used in this study can also be found in Table II. The purpose of using these two devices was to evaluate the outcome and the consistency of the Klein block cipher in two different environments. Since no reference code of the Klein block cipher implementation has been published by its designers, an implementation of Klein-64 has been written in the Python 3 programming language. To make sure that the integrity and consistency of the application were met, the implementation of the program was followed Gong et al. [6]. In the initial design of Klein encryption, the program was divided into 4 individual scripts. Each script contained the implementation of the steps in the Klein block cipher. Since the four steps in the Klein block cipher present one iteration, we implemented the application to iterate twelve rounds.

TABLE II
Table of Development Environment

	Constrained Device	**Non-constrained device**
Resources	**Raspberry Pi 3**	**Windows 10**
CPU	Broad-com BCM2711	i7 Dual Core
CPU clock speed (GHz)	1.5	2.4
Memory	1	8
Memory speed (MHz)	900	1600

The application is written in Python language for its simplicity and use [27], although it could have been implemented in other languages such as C, but regardless of the platform, we expect to find consistent results.

Raspberry Pi 3 (RPI3) is a small and inexpensive computer device that is used in network computing, remote monitoring and with sensor networks [28]. In this paper, a RPI3 is chosen for the experiments because of the accessibility of the device and the current usage of RPI3 in the IoT. Due to limitations of the research the Windows 10 computer is used as a different environment to observe the consistency and continuity of different experiments. The authors of [12], have also conducted similar experiment with microcontrollers as well as higher-end CPUs. Nonetheless, the results of our experiments from this device would show a realistic outcome if performed on a WSN device.

B. Alternative Algorithm

Alternative algorithm is the replacement of an algorithm of the Klein block cipher that has poor performance with an algorithm that can provide an improvement to the performance. The main objective of this approach is to replace the last algorithm (*MixNibble*) of the Klein block cipher with 3-stage S-box. On the other hand, an evaluation of the overall resource benchmark of each algorithm of the Klein block cipher will be measured. Although

the overall software performance of Klein-64 has been evaluated [4], the software performance of each algorithm in the Klein block cipher is unknown [4], [29]. The construction of the Klein block cipher is to purposely measure the CPU percentage, the memory in bytes that the algorithm used, the overall time the program was executed, and the amount of data for the specific algorithm of the Klein block cipher. Finally, both algorithms will be compared for efficiency [30].

The *MixNibble:* In the last round of the Klein block cipher, the state is processed by Rijndael's MixColumn (*MixNibble*). This step works on a 4 bytes element of the Galois Field, and the equation can be written as:

$$(2^8)=x^8+x^4+x^2+1$$

The output is composed of 4 bytes and is multiplied by the matrix below [19].

$$\begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix}$$

The evaluation of the Klein block cipher has been achieved by executing and analysing the *MixNibble* algorithm. The execution of this algorithm was done once in Windows 10 and once in the RPI3 environment. The implementation is done by acquiring four different scripts with necessary information given, as mentioned above, to evaluate the data, CPU speed, memory usage, and time it took [31]. Once the results for the normal Klein block cipher algorithm are achieved, the 3 Substitute Box (3 S-box) algorithm will be tested. The 3 S-box implementation is a 4-bit permutation that applies to 16 nibbles after the *RotateNibble* algorithm. Table II represents the values of 3 S-box. Also, to ensure the message's integrity, before the 3 S-box, the result from the *RotateNibble* is XORed with the key. The first algorithm of the S-box applies, and the result is again XORed with the key. It proceeds with the second S-box and, the result of the second S-box is XORed with the key. It finishes the round after the third S-box. This can be written as shown in Algorithm 1.

C. Increase Iteration

Unlike the SIMON and the SPECK block ciphers that are part of the Feistel-Permutation family, the Klein block cipher is from the SPN family [3], [32]. SPN ciphers provide more equilibrium between linear and non-linear permutations. The software is critically important to consider in bit permutation because implementing non-linear bit permutation can cause complexity in operation. Furthermore, SPN block ciphers such as PRESENT are cheaper in hardware and software. Therefore, they are more likely to have more rounds to satisfy permutation.

Regarding the current attacks on the Klein block cipher, such as the differential characteristics attack and the PCMITM attack, both attacks have been successful in key recovery within 8 and 10 rounds, respectively. Regarding the Klein block cipher with key exploitation at lower rounds, this study proposes to take an approach against key recovery by increasing the iteration of the Klein block cipher. The implementation would include four different configurations. They are 12, 16, 20, and 32 rounds. Each configuration is executed with the same key and has the same block size. In the execution of each configuration there will be some gradual delay. Nonetheless, this optimisation is more targeted on the effectiveness of attacks on the Klein block cipher. However, we would note each configuration's time to complete the iterations.

Given SPN ciphers are likely to be balanced in software, the permutation of bits is more efficient. Also, the fact that there has been successful cryptanalysis of block ciphers such as Klein-64 means it is beneficial to construct a program like the Klein block cipher with more iterations. In implementing increased rounds, the program was divided into four scripts consisting of different iterations. The four programs consist of 12, 16, 20, and 32 rounds. Each round had the same key and plaintext applied, but it ran in different configurations. The evaluation of the programs would be based on the CPU rate, the ratio of the memory, the memory used, and the time each took.

TABLE III
SUBSTITUTE BOXES USED IN ALGORITHM TRIPLE NIBBLE

Substitute Box 1																
	x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$Sub_1(X)$	B	F	8	C	9	E	7	6	2	4	D	0	3	A	5	1
Substitute Box 2																	
	X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$Sub_2(X)$	8	3	F	1	6	B	4	E	0	C	D	5	9	A	7	2
Substitute Box 3																	
	X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	$Sub_3(X)$	9	B	4	7	2	C	E	3	F	0	D	1	5	A	6	8

Algorithm 1 Triple Nibble Substitutes

Input: 64-bit binary state STATE RotateNibble

Output: 64-bit ciphertext CT

1: **for** i = 1 to 3 do:
2: $STATE \leftarrow STATE \oplus subkey_i$
3: $STATE \leftarrow sub_i(STATE)$;
4:
5: **end for**
6: **return** CT

IV. Results

In this section, we present the results of the two approaches described in section III.

A. Alternative Algorithm

The four scripts that consisted of each algorithm of the Klein block cipher were evaluated for CPU speed, usage of memory, the time executed, and the data each process. Each round was tested on two different devices to show the software performance. The first experiment was tested using Windows 10. When the Python program was executed in the Windows environment, it showed how much CPU was used for the specific process. The first algorithm had a higher percentage rate than the second and third algorithms; however, in the fourth algorithm, a bigger percentage rate was shown. The usage of the memory for all programs remained similar, but in terms of the execution time, the first and the fourth programs had longer times compared to the second and third programs. The results are shown in Table IV.

The evaluation of the Klein block cipher using RPI3 showed results with slightly different percentages for each algorithm. In this implementation, the CPU percentage of the first and second algorithms was reasonably low compared to the third algorithm, which slightly increased. Finally, the last algorithm had the highest percentage of all. In terms of the memory ratio, the results were similar to the results in the Windows 10 environment. The memory percentage slightly increased towards the last algorithm; however, the time for each algorithm to execute was different. Again, the results were similar to the results for the Windows 10 environment. The second and third algorithms took the same time to execute. The results of the algorithms run on the Raspberry Pi environment are shown in Table V.

The results for 3 S-box in Windows 10 showed a higher CPU rating when conducted initially. The memory usage was at 8352 bytes, the execution took almost 0.05 seconds, and the data processed was 267 bytes. The results are shown in Table VI.

The results for 3 S-box in RPI 3 showed a high CPU percentage. The memory usage was at 17076 bytes, the execution took almost 0.09 seconds, and the data processed was 267 bytes. The results are shown in Table VII.

TABLE IV
Evaluation of Klein Block Cipher in Win 10

Algorithm	CPU (%)	RAM (KB)	Time (s)	Data (Byte)
AddroundKey	35.0	8500	0.078095	289
SubNibble	25.0	8500	0.0156.22	89
RotateNibble	25.0	8468	0.015626	178
MixNibble	58.3	8624	0.0781302	264

TABLE V
Evaluation of Klein Block Cipher on RPI 3

Algorithm	CPU (%)	Memory (%)	RAM (KB)	Time (s)	Data (Byte)
AddroundKey	7.0	1.8	17268	0.141437	289
SubNibble	7.0	1.8	17292	0.065789	89
RotateNibble	9.7	1.8	17244	0.065367	178
MixNibble	16.0	1.8	17324	0.260921	264

TABLE VI
Evaluation of 3 S-BOX in Win 10

Algorithm	CPU (%)	RAM (KB)	Time (s)
AddroundKey	35.0	7908	0.079
SubNibble	25.0	8008	0.0157
RotateNibble	25.0	8048	0.0161
3 S-box	**25.0**	**8352**	**0.0468**

TABLE VII
Evaluation of 3 S-BOX in RPI 3

Algorithm	CPU (%)	Memory (%)	RAM (KB)	Time (s)
AddroundKey	7.0	1.8	17265	0.142
SubNibble	7.1	1.8	17292	0.072
RotateNibble	8.9	1.8	17248	0.065
3 S-box	**27.0**	**1.8**	**17076**	**0.088**

Given the individual results of Klein encryption algorithms, it is important to evaluate the optimised block cipher (3 S-box) within a full iteration (12 rounds) of Klein block cipher and then compare the results with the original Klein block cipher within a full iteration. The evaluation of the original and 3 S-box

is performed both on Windows 10 and Raspberry Pi. Each test consisted of a full iteration and ran four times. Their average values are recorded in Table VIII.

The results show a relatively high CPU percentage compared to the time taken. Similarly, the results of the 3 S-box in both platforms showed a higher percentage with a lower execution time. A study [33] has found that Klein encryption has a lower degree of diffusion and a higher degree of confusion. In our first experiment, we examined the performance of each algorithm in the Klein block cipher. The results showed that *MixNibble* and *RotateNibble* had a higher CPU percentage than other algorithms when performed on the Raspberry Pi device. In a related paper [33], the authors also compared the program memory and data memory usages of Klein encryption with the Tiny Encryption Algorithm (TEA), the High Security and Light Weight (HIGHT), and the KATAN encryption systems. The program memory usage of Klein is higher than the rest of the block ciphers, and the reason is the weight of the program code. Despite *MixNibble* having a diffusion property, it has managed to consume higher CPU power and take more time for encryption. Another performance analysis concluded that RC6 provides greater security than RC5 with the extra (four) registers [34]. Although this increases the throughput and diffusion, the encryption will be achieved in fewer rounds.

B. Increased Iteration

The results of the four configurations constructed to evaluate the performance of the resources and the time each took are shown in Table IX. Each configuration has been executed four times for data accuracy. The results showed a high clock speed CPU for the (12) rounds, but this gradually came down by the fourth attempt. The memory usage average showed 17.31KB. The next configuration was 16 rounds which showed a high clock speed at the first execution, but it halved by the fourth attempt. The memory usage was similar to the average of the 20 rounds configuration. In the 20 and 32 rounds configuration, the clock speed was similar, however, the memory usage increased from the 20 rounds configuration. Regardless of the memory usages in all configurations, the CPU speed and the execution time were highly noticeable for their effect on the hardware resources.

TABLE VIII
FULL ITERATION OF ORIGINAL AND 3 S-BOX KLEIN BLOCK CIPHER IN RPI 3 AND WIN 10

Platform	Algorithm	CPU (%)	Memory (%)	RAM (KB)	Time (s)
RPI 3	Original	21.9	1.8	17076	0.381
RPI 3	3 S-box	29.1	1.775	17076	0.052
Win 10	Original	15.92	NA	8352	0.149
Win 10	3 S-box	15.53	NA	8352	0.057

TABLE IX
FULL ITERATION OF ORIGINAL AND 3 S-BOX KLEIN BLOCK CIPHER IN RPI 3 AND WIN 10

Iteration	CPU (%)	Memory (%)	RSS (KB)	Time (s)
12	19.1	1.8	17315	0.043
16	**10.9**	**1.8**	**17394**	**0.54**
20	7.9	1.8	17919	0.7
32	7.55	1.875	18081	1.26

V. DISCUSSION

In this section, we discuss the results of the approaches to the modified Klein encryption. The Klein block cipher with 64-bit key size was tested and evaluated with two different approaches. As we recall, Klein encryption is a lightweight cipher suited for WSN devices. The two approaches conducted on Klein encryption were performed previously on other encryption ciphers such as AES, Blowfish, and a lightweight block cipher. We proposed to enhance the performance of Klein encryption using these two approaches. Although the performance enhancement experiments similar to our approaches have been tested on other encryption schemes, no studies have tested these approaches on the Klein encryption. Our study aimed to enhance the performance of the Klein block cipher. However, since Klein encryption is a lightweight cipher, it must remain lightweight regardless of the proposed idea. Therefore, we developed the two approaches on the Klein block cipher to evaluate the software performance first. By evaluating the effect of each approach, we expect to have a better understanding of the performance of the modified Klein block cipher.

A. Optimisation Alternative Algorithm

The four scripts constructed to test the Klein block cipher were the *AddRoundKey*, *SubNibble*, *RotateNibble*, and *MixNibble* algorithms. Each algorithm had similar results when performed on Windows 10 and RPI3 devices. The recorded results between the two devices show some information related to their hardware. It is known that the Windows 10 device has 8 GB of memory. However, not all the memory is available for use. For example, the computer that ran the first set of results had 4.3 GB used for processes and 3.6 GB of memory available. This is a total of 7.9 GB available for the whole computer hardware to use. And it had 0.08 GB reserved for other hardware purposes.

On the other hand, the RPI3 device was supplied with 1 GB of memory. However, the official website of RPI3 and its configuration files states that it only uses 0.88 GB of memory for the processes. The configuration files show 0.128 GB of memory usage by the Graphical Processing Unit (GPU). The total amount shows 1.008 GB of physical memory. It is important to consider that other processes are also running. But in this regard, the Windows device has more available memory to spare than the Raspberry Pi device. Therefore, the results show a higher rate of the memory. The memory management of these two devices is entirely unpredictable, as many processes could be running at the time of the implementation and evaluation. However, to make sure the results of the programs were consistent, the same Python libraries were used to measure the resources and the time of the programs. As discussed above, each device has different hardware, and they output different readings in terms of the results. Nonetheless, they have produced similar outcomes. With the first program tested, the initial run showed a higher rating within CPU percentage in the Windows device compared to the RPI3 device. For the Windows device, 35% of CPU time (0.84 GHz out of 2.4 GHz) was spent executing the program, compared to 7% of CPU time for the RPI3 device. The RPI3 device has a maximum of 1.5 GHz, which means only 0.0105 GHz was used for this run. The high CPU percentage of the first algorithm in the Windows device is due to the operation that is being handled. As mentioned previously, the XOR operation can have a higher transistor than other logic gates. The memory percentage is the ratio of the Resident Set Size (RSS) set to the physical memory of the device. The ratio of the memory was not an option available on the Windows device. The actual memory size recorded in the Windows device was relatively smaller than the memory used in the RPI3 device. Again, this could be related to the distribution of resources across the processors of each device. However, the memory results show a stable reading for both devices. In this regard, the amount of the used data was the same. However, there was a slight delay of 0.063 seconds (44.7% slower) when the program was executed in Raspberry Pi. In the Windows device, the *SubNibble* and *RotateNibble* algorithms showed similar results, considering that the *RotateNibble* has a higher amount of data to process. Nevertheless, the *SubNibble* algorithm had a longer script to process. The process of *SubNibble* is to identify each nibble from the list and perform nibble permutation. On the other hand, *RotateNibble* is swapping the bytes in the list. The difference between the two is the data that is being manipulated and the number of codes within the script. The important information about the results is the memory used, and the time each took to complete. The memory difference is 32 Kilobytes (KB), and the time is 0.000004 seconds. In the Raspberry Pi device, the *SubNibble* algorithm took 2.7% less CPU time. The CPU usage of algorithms run in Windows, and Raspberry Pi are shown in Figures 2 and 3, respectively. As shown, the CPU usage in both devices was high for *MixNibble* while the lowest CPU usage was *SubNibble* and *RotateNibble* in Windows and *AddRoundKey* and *SubNibble* in the Raspberry Pi environment.

The memory difference between *RotateNibble* and *SubNibble* is 48 KB and 0.000422 seconds. This shows *RotateNibble* takes 9% more time to execute than *SubNibble*. The RAM usage of algorithms run in Windows, and Raspberry Pi is shown in Figures 4 and 5, respectively. The RAM usage in both devices was high for *MixNibble*, while the lowest RAM usage was for *RotateNibble*.

The last algorithm, *MixNibble*, shows a significant amount of resources both in the Raspberry Pi

and Windows devices. This is due to the amount of data processed. This section used the XOR logical gate and multiplication, which increased the need for more resources due to the long operation. The 3 S-box in the Windows 10 and Raspberry Pi environments showed a high execution of CPU. This was because the program had 3 XOR operations plus three sets of *SubNibble* algorithms [35]. The operation consumes a higher CPU at the initial point; however, the memory usage for both has stayed reasonably low. The time that each took for execution is 0.05 and 0.09 seconds, respectively. When 3 S-box is compared with the *MixNibble* algorithm run in the Windows 10 environment, *MixNibble* used twice as much CPU. The amount of memory *MixNibble* used was 8,624 KB, and the amount of memory 3 S-box used was 8,352 KB, a difference of 272 KB. In terms of the time difference, there was a delay in the *MixNibble* algorithm of 0.0313 seconds. In the Raspberry Pi environment, the *MixNibble* algorithm was executed 1.68% faster than 3 Sbox. The difference in memory usage between 3 S-box and *MixNibble* is 248 KB. Also, 3 S-box executed 0.0172 seconds faster than *MixNibble*.

The 3 S-box showed a higher CPU percentage when executed. However, in relation to other resource consumption, the 3 S-box was analysed to have optimised performance for memory, time, and implementation of cost. Despite the advantages of software performance, security may not be guaranteed, and the results of this study are based on factual argumentation. Enhanced S-box, compared to the original AES encryption, replaces the Rijndael function [25]. The authors performed a NIST statistical test and cryptanalysis attack on the AES block cipher with enhanced S-box. The results of the experiments showing the original security of AES was not modified rather added confusion to the property of this algorithm which is diffusion.

B. Optimisation of Increased Iteration

Evaluation of the four programs with different iteration lengths was done using the RPI3 device, an environment closest to a wireless sensor device. The replication of this environment gave several different yet interesting results on the effect of each iteration. The four tested programs on the device had four configurations: 12, 16, 20, and 32 iterations. The size of the plaintext and keys were consistent across all configurations. Since the software performance of Klein-64 has been evaluated with 3,689 iterations per byte, this study focused on the iterations with the above configurations with only 8 bytes (64-bits).

For better results, each iteration was performed four times. The average time of the execution of each configuration was recorded. The results for the first configuration show higher resource usage. The CPU usage of the first run had a high rate of 27%. Then after the fourth run, it had a rating of 7%. There was a delay of 10 seconds between each run.

Nonetheless, the amount of CPU usage for the first until the third run was high in the first configuration. The memory usage was stable, with a mean of 17,315 KB. The average time that it took to complete was 0.043 seconds.

The second configuration had 16 iterations. The results for this configuration show a high CPU usage rate for the first run, but from the second to the fourth run, the percentage halved. Consequently, the difference between the first and last run memory usage was 1.41%. Compared to the average memory usage of the first configuration, only 0.45% extra memory was used.

In the third configuration, the 20 iterations, the CPU usage seemed to be low when initiated but gradually increased by the end of the fourth run. However, the memory increased in usage by 2.25%. The average time that it took to perform the full iteration was 0.7 seconds, 25% and 176% slower when compared to second and first configurations, respectively. The final configuration, which had 32 iterations, resulted in a lower CPU usage time (mean of 7.55%); however, the memory usage ratio towards the program increased during the last run. A rating of 1.9% showed 18,200 KB of memory usage. The average time it took to complete the whole iteration was 1.26 seconds. The memory usage difference between 12 iterations and 32 iterations increased 5.4%. In terms of average delays, it was 1.217 seconds slower to perform 32 iterations than 12 iterations (that is 186%).

Nonetheless, the CPU usage of the first configuration was higher than the last configuration (that is 86% slower) when the 12 iterations were executed. Usually, increasing the iteration of an encryption system does not make the encryption scheme optimised; however, it is possible to infer that the encryption operations could be more efficient. A recent study [36] on the power consumption of Raspberry Pi 2 GPU rendering between software and hardware was done, and the results showed hardware rendering is more efficient than software rendering. Although hardware uses more power, software rendering takes more time. We can assume the same behaviour occurred with the experiments done. However, it is worth mentioning that this experiment was mainly affected the CPU and memory from the software perspective. Introducing higher iterations can cause a delay in implementation and more resource consumption, and using lower rounds such as 12 iterations could cause very high resource consumption. In the real-world scenario, 16 iteration configurations seem to be ideal when considered for a sensor device. Since the 16 iteration configuration encrypted 64 bits of information (8 characters), if the message had to be longer, e.g., 25 (32 characters), the time to run would be 2.16 seconds. In terms of resource usage, it would consume 43.6% of the CPU and use 69,576 KB of memory. When compared to other configurations, the size of the memory usage would be higher. In the first configuration, although it had a lower memory usage, more CPU was used.

In the differential characteristics attack, for 8 rounds of key recovery, it was recorded that the program would take 1,344 seconds to achieve the round key. This attack was done on Klein-64 with 12 rounds. In the 16 rounds of Klein encryption, this would mean 1,792 seconds to key recovery or 28% more delay [5], [18]. Kumar and Rana [14] increased the number of rounds from 10 to 16 for encryption and decryption. The authors stated that a higher number of rounds resulted in higher security for intruders to breach. For this case they have modified the AES block cipher with 320 bits key size for 16 rounds. The operation of the modified AES block cipher is dependent on the initial key, which is generated by the Polybius square with a 6 by 6 matrix, that includes both numbers and letters. The values are arranged with no repetition from left to right. In this paper, DES, TDES, AES and modified AES have been implemented based on Throughput on various sizes of files. The results of the modified AES block cipher showed the highest time to encrypt was for the 16 rounds. With the implementation of increased rounds, the Polybius square authors claim it contains higher security and is less prone to intruders. Ahmed et al. [34] have suggested that with increased rounds, the change of one-pixel is possible in image encryption; however, the rate of one-pixel change is small. Therefore, they concluded that by increasing the number of rounds, a higher security is achieved.

C. Future Work

Although the need for WSN devices is highly significant, the secrecy of the data that they transmit is important too. With this requirement, performance of the encryption must be suitable for constrained devices to run effectively. In future work, we are determined to implement different logical gates operations such as AND gate and OR gate for their lower resource consumption. This study performed software optimisation of Klein block cipher; however, security optimisation of Klein block cipher is yet required to be evaluated in further studies.

VI. Conclusion

This study evaluated the software performance of the Klein block cipher. The analysed approaches were a modified algorithm (3 S-box) and increased iteration of the original Klein block cipher. The 3 S-box had a greater effect on software performance with a small limitation when compared with the *MixNibble* algorithm. Improvements were observed with the average CPU, memory and time when 16 iterations were used compared to default iteration of Klein-64. Results from repeating experiments confirmed the initial findings. The results of suggestive approaches (3 S-box and increased iterations) showed more efficiency in performance than the original algorithm.

Acknowledgment

This work has been supported by the Cyber Security Research Centre Limited, whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

References

[1] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Bus. Horiz.*, vol. 58, no. 4, pp. 431–440, 2015, doi: 10.1016/j.bushor.2015.03.008.

[2] E. Leloglu, "A review of security concerns in internet of things," *J. Comput. Commun.*, vol. 5, no. 1, pp. 121–136, 2016, doi: 10.4236/jcc.2017.51010.

[3] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck lightweight block ciphers," in *2015 52nd Annu. Des. Autom. Conf.*, CA, USA, pp. 1–6, doi: 10.1145/2744769.2747946.

[4] M. Cazorla, K. Marquet, and M. Minier, "Survey and benchmark of lightweight block ciphers for wireless sensor networks," in *2013 Int. Conf. Secur. Cryptography (SECRYPT)*, Iceland, pp. 1–6.

[5] J.-P. Aumasson, M. Naya-Plasencia, and M.-J. O. Saarinen, "Practical attack on 8 rounds of the lightweight block cipher klein," in *Int. Conf. Cryptology India.*, in Progress in Cryptology-INDOCRYPY 2011, vol. 1 2011, pp. 134–145, doi: 10.1007/978-3-642-25578-6_11.

[6] Z. Gong, S. Nikova, and Y. W. Law, "Klein: a new family of lightweight block ciphers," in *REID. Scurity and Privacy (RFIDSec 2011)*, A. Juels and C. Paar, Ed., 2011, pp. 1–18.

[7] D. Luciano and G. Prichett, "Cryptology: From caesar ciphers to publickey cryptosystems," *Coll. Math. J.*, vol. 18, no. 1, pp. 2–17, 1987, doi: 10.2307/2686311.

[8] P. Dixit, A. K. Gupta, M. C. Trivedi, and V. K. Yadav, "Traditional and hybrid encryption techniques: a survey," in *Networking communication and data knowledge engineering*, G. M. Perez, K. K. Mishra, S. Tiwari, and M. C. Trivedi, Eds., Singapore: Springer, 2018, ch. 22, pp. 239–248.

[9] S. B. Sadkhan and A. O. Salman, "A survey on lightweight-cryptography status and future challenges," in *2018 Int. Conf. Adv. Sustain. Eng. Appl. (ICASEA)*, Iraq, pp. 105–108, doi: 10.1109/ICASEA.2018.8370965.

[10] B. Coskun and N. Memon, "Confusion/diffusion capabilities of some robust hash functions," in *2006 40th Annu. Conf. Inf. Sci. Syst.*, Princeton, NJ, USA, pp. 1188–1193, doi: 10.1109/CISS.2006.286645.

[11] M. Hellman, "A cryptanalytic time-memory trade-off," *IEEE trans. Inf. Theory*, vol. 26, no. 4, pp. 401–406, July 1980, doi: 10.1109/TIT.1980.1056220.

[12] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE: A Lightweight, Versatile Block Cipher," in *Proc. ECRYPT Workshop Lightweight Cryptography*, Belgium, Nov. 2011, pp. 146-169.

[13] A. Bogdanov, et al., "PRESENT: An Ultra-Lightweight Block Cipher," in *Proc. Int. Workshop Cryptogr. Hardw. Embed.*, P. Paillier and I. Verbauwhede, Eds., in Lecture Notes in Computer Sciense, vol. 4727, 2017, pp. 450-466.

[14] P. Kumar and S. B. Rana, "Development of modified aes algorithm for data security," *Optik*, vol. 127, no. 4, pp. 2341–2345, 2016, doi: 10.1016/j.ijleo.2015.11.188.

[15] P. Kushwaha, M. P. Singh, and P. Kumar, "A Survey on Lightweight Block Ciphers," *Int. J. Comput. Appl.*, vol. 96, no. 17, pp. 1-7, June 2014, doi: 10.5120/16883-6923.

[16] F. Abed, E. List, S. Lucks, and J. Wenzel, "Cryptanalysis of the speck family of block ciphers." *IACR Cryptol. ePrint Arch.*, Report 2013/568, 2013. [Online]. Available: https://eprint.iacr.org/2013/568

[17] A. Heuser, S. Picek, S. Guilley, and N. Mentens, "Side-channel analysis of lightweight ciphers: Does lightweight equal easy?" in *12th International Workshop RFIDSec 2016*, in Radio Frequency Identification and IoT Security, G. P. Hanche and K. Markantonakis, Eds., in Lecture Note in Computer Science, vol. 10155, 2016, pp. 91–104, doi: 10.1007/978-3-319-62024-4_7.

[18] I. Nikolic, L. Wang, and S. Wu, "The parallel-cut meet-in-the-middle´ attack," *Cryptogr. Commun.*, vol. 7, no. 3, pp. 331–345, 2015, doi: 10.1007/s12095-014-0118-1.

[19] V. Lallemand and M. Naya-Plasencia, "Cryptanalysis of klein," in *International Workshop on Fast Software Encryption*, in Fast Software Encryption, C. Cid and C. Rechberger, Eds., in Lecture Notes in Computer Science, vol. 8540, 2014, pp. 451–470, doi: 10.1007/978-3-662-46706-0_23.

[20] R. S. Romaniuk, "Iot–review of critical issues," *Int. J. Electron. Telecommun.*, vol. 64, no. 1, pp. 95–102, 2018, doi: 10.24425/118153.

[21] A. R. Sfar, E. Natalizio, Y. Challal, and Z. Chtourou, "A roadmap for security challenges in the internet of things," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 118–137, 2018, doi: 10.1016/j.dcan.2017.04.003.

[22] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full aes-192 and aes-256," in *15th Int. Conf. Theory Appl. Cryptology Inf. Secur.*, in Advances in Cryptology – ASIACRYPT 2009, M. Matsui, Ed., in Lec-

ture Notes in Computer Science, vol. 5912, 2009, pp. 1–18, doi: 10.1007/978-3-642-10366-7_1.

[23] M. C¸oban, F. Karakoc¸, and O. Boztas¸, "Biclique cryptanalysis of twine," in *11th Int. Conf. Cryptology Netw. Secur.*, in Cryptology and Network Security, J. Pieprzyk, A.-R. Sadeghi, and M. Manulis, Eds., in Lecture Notes in Computer Science, vol. 7712, 2012, pp. 43–55, doi: 10.1007/978-3-642-35404-5_5s.

[24] A. Gupta and S. Kaushik, "A review: Rsa and aes algorithm," *IITM J. Manag. IT*, vol. 8, no. 1, pp. 82–85, 2017.

[25] J. Juremi, R. Mahmod, S. Sulaiman, and J. Ramli, "Enhancing advanced encryption standard s-box generation based on round key," *Int. J. Cyber-Secur. Digit. Forensics (IJCSDF)*, vol. 1, no. 3, pp. 183–188, 2012.

[26] E. Tena-Sanchez, J. Castro, and A. J. Acosta, "A methodology for´ optimized design of secure differential logic gates for DPA resistant circuits," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 2, pp. 203–215, 2014, doi: 10.1109/JETCAS.2014.2315878.

[27] J. Wainer and E. C. Xavier, "A controlled experiment on python vs c for an introductory programming course: Students' outcomes," *ACM Trans. Comput. Educ.*, vol. 18, no. 3, pp. 1–16, 2018, doi: 10.1145/3152894.

[28] S. Jain, A. Vaibhav, and L. Goyal, "Raspberry pi based interactive home automation system through e-mail," in *2014 Int. Conf. Reliability Optim. Inf. Technol. (ICROIT)*, India, pp. 277–280, doi: 10.1109/ICROIT.2014.6798330.

[29] D. Irwin, P. Liu, S. Chaudhry, M. Collier, and X. Wang, "A performance comparison of the present lightweight cryptography algorithm on different hardware platforms," in *29th Irish Signals Syst. Conf. (ISSC)*, UK, 2018, pp. 1–5, doi: 10.1109/ISSC.2018.8585341.

[30] J. Hosseinzadeh and A. G. Bafghi, "Software implementation and evaluation of lightweight symmetric block ciphers of the energy perspectives and memory," *arXiv:1706.03909*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03909.

[31] J. Grossschadl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, France, 2007, pp. 1–6, doi: 10.1109/DATE.2007.364443.

[32] H. AlKhzaimi and M. M. Lauridsen, "Cryptanalysis of the simon family of block ciphers." *IACR Cryptol. ePrint Arch.*, Report 2013/543, p. 543, 2013. [Online]. Available: https://eprint.iacr.org/2013/543.pdf

[33] M. Alizadeh, M. Salleh, M. Zamani, J. Shayan, and S. Karamizadeh, "Security and performance evaluation of lightweight cryptographic algorithms in rfid," in Proc. *16th WSEAS Int. Conf. Comput.*, Kos Island, Greece, 2012, pp. 45-50.

[34] H. E.-d. H. Ahmed, H. M. Kalash, and O. F. Allah, "Encryption efficiency analysis and security evaluation of rc6 block cipher for digital images," in *2007 Int. Conf. Electr. Eng.*, Pakistan, 2007, pp. 1–7, doi: 10.1109/ICEE.2007.4287293.

[35] D. Hong, et al., "Hight: A new block cipher suitable for lower source device," in *Proc. Int. Workshop Cryptogr. Hardw. Embed. Syst.*, L. Goubin, and M. Matsui, Eds., in Lecture Notes on Computer Science, vol. 4249, 2006, pp. 46–59, doi: 10.1007/11894063_4.

[36] Q. He, B. Segee, and V. Weaver, "Raspberry pi 2 b+ gpu power, performance, and energy implications," in *2016 Int. Conf. Comput. Sci. Comput. Intell. (CSCI).*, Las Vegas, NV, USA, 2016, pp. 163–167, doi: 10.1109/CSCI.2016.0038.