An Enhanced Block Pre-Processing of PRESENT Algorithm for Fingerprint Template Encryption in the Internet of Things Environment

Norliza Katuk* and Ikenna Rene Chiadighikaobi

School of Computing, Universiti Utara Malaysia, Malaysia
*Corresponding author

Abstract: Many previous studies had proven that The PRESENT algorithm is ultra-lightweight encryption. Therefore, it is suitable for use in an IoT environment. However, the main problem with block encryption algorithms like PRESENT is that it causes attackers to break the encryption key. In the context of a fingerprint template, it contains a header and many zero blocks that lead to a pattern and make it easier for attackers to obtain an encryption key. Thus, this research proposed header and zero blocks bypass method during the block pre-processing to overcome this problem. First, the original PRESENT algorithm was enhanced by incorporating the block pre-processing phase. Then, the algorithm's performance was tested using three measures: time, memory usage, and CPU usage for encrypting and decrypting fingerprint templates. This study demonstrated that the proposed method encrypted and decrypted the fingerprint templates faster with the same CPU usage of the original algorithm but consumed higher memory. Thus, it has the potential to be used in IoT environments for security.

Keywords: Internet of Things, security, fingerprint template, encryption, lightweight

1. Introduction

Small computing devices are becoming more widespread and emerged as an essential part of the Internet of Things (IoT). It is the backbone of applications in various domains like healthcare, agriculture, transportation, smart cities [1]. In this type of network, devices send a large volume of sensitive data; therefore, data security is the researchers' main concern, primarily through encryption algorithms [2]. However, traditional symmetric encryption algorithms are not suitable for IoT devices due to hardware limitations. They cannot achieve acceptable hardware conditions and performance with their limited power supply [3]. Therefore, lightweight encryption algorithms become the best option to ensure the security of such information [1, 4-6] that is generated by such small devices [5, 7].

Furthermore, resource-constraint environments require lightweight cryptography to accommodate features of compact implementation, small memory, and low power supply [1]. The lightweight cryptography addresses the limitation of that traditional cryptography which cannot be used in this environment due to high implementation costs [8, 9]. Examples of lightweight block encryption algorithms include PRESENT, CLEFIA, MIBS, and LBlock [10]. Many lightweight encryption algorithms have been invented so far to be applied in various settings. Panahi et al. [11] listed eighty-one algorithms with their features to demonstrate the available algorithms for lightweight implementation. One of the popular ultra-lightweight encryption algorithms is PRESENT [12], with simple encryption key scheduling [1, 13] compared to others. Various types of data sent within devices in IoT networks need to be protected in terms of confidentiality using encryption algorithms. One crucial data is user credential information such as password and personal identification number (PIN) to verify their identity before using the network or system resources. In addition to passwords and PINs, many IoT systems use biometric factors such as facial images and fingerprints for user authentication purposes. The biometrics authentication system performs a matching process between the captured fingerprint or facial images (using a scanner or camera) with the authentication information stored in the database. This biometric information is called a template, which is stored in the form of the hexadecimal string [14]. This study mainly focuses on the fingerprint template captured by optical fingerprint scanners and then stored in the form of hexadecimal strings, not the fingerprint images.

Further, this study also aims to protect the templates generated by the scanners through an appropriate encryption algorithm. However, the initial part on how the scanners perform feature recognition and extraction processes is beyond this study's scope. It is because IoT applications use an optical fingerprint sensor attached to a microcontroller for user authentication. Fingerprint sensors are an industry standard, which has been widely used and adopted. However, the significant gap in the current fingerprint templates generated by the industry-standard optical fingerprint sensors is that they are still in plain text, exposing them to security attacks and consequently causing user identity theft. Hence, they should be protected using appropriate encryption algorithms. Based on this situation, there is an urgent need to look into how these templates could be protected within the constraint-resource environment.

This study aims to improve the PRESENT algorithm so that it can be adapted to encrypt fingerprint templates generated by optical scanners in an IoT environment. Recent studies have found that PRESENT is an ultra-lightweight encryption algorithm that can be used in IoT environments [1, 3, 10, 11, 15]. However, the performance of this algorithm in encrypting fingerprint templates in the form of hexadecimal strings has not been explored. Furthermore, to the best of our knowledge, the existing research has only focused on encrypting fingerprint images using public datasets like Fingerprint Verification Competition (FVC). Nevertheless, no research has explored the encryption algorithms suitable for fingerprint templates generated by these optical scanners. Thus, the results of this research can contribute to the improvement of data security protection in industry-standard optical scanners.

2. Related Works

This section will describe two basic things in this study: background on the PRESENT algorithm and fingerprint template encryption. In addition, it will also list findings from similar studies to establish a fundamental knowledge of the studies proposed in this article and their related development.

2.1 The PRESENT Algorithm

PRESENT is an ultra-lightweight block encryption algorithm with a lower implementation cost than similar algorithms [12, 16]. The algorithm performed substitution and permutation processes on the plaintext of a block of 8-byte in 31 rounds with the encryption key to generate its ciphertext. The overall encryption process using the PRESENT algorithm is illustrated in Figure 1.

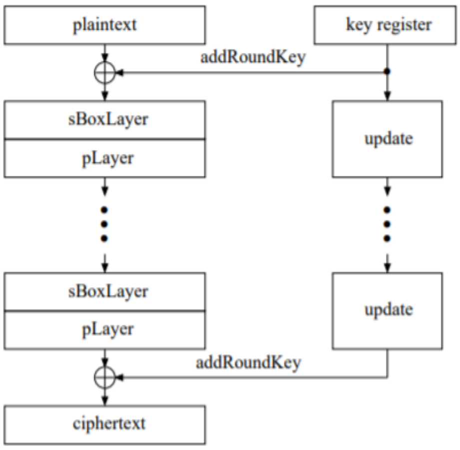

Figure 1. The PRESENT encryption process [12, 16]

The PRESENT algorithm uses 80-bit or 128-bit key size for performing the encryption. However, this study focuses 80-bit key to accommodate the IoT resource-constraint environment. The key is stored in a key register K with individual bytes are stored in decreasing order as represented in (1).

$$K = k_{79}\, k_{78} \ldots\ldots k_1 k_0 \quad (1)$$

The algorithm will extract 64-bit subkey Kj in which j is the number of a round of the key scheduling process as rendered in (2).

$$K_j = k_{63}\, k_{62,} \ldots\ldots k_1 k_0 = k_{78}\, k_{78} \ldots\ldots k_{17} k_{16} \quad (2)$$

After that, the algorithm updates the key register K as stated in (3), (4) and (5) in producing the addRoundKey function.

$$[k_{79}\, k_{78,} \ldots\ldots k_1 k_0\,] = [k_{18}\, k_{17,} \ldots\ldots k_{20} k_{19}] \quad (3)$$

$$[k_{79}\, k_{78} k_{77} k_{76}] = S[k_{79}\, k_{78} k_{77} k_{76}] \quad (4)$$

$$[k_{19}\, k_{18} k_{17} k_{16} k_{15}] = [k_{19}\, k_{18} k_{17} k_{16} k_{15}] \oplus round\_counter \quad (5)$$

The output of addRoundKey will be XORed with the plaintext before the input text undergoing the SBoxLayer and pLayer process. The SBoxLayer performed a substitution process using substitution rules in Table 1.

Table 1. PRESENT S-BoxLayer [12, 16]

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The output of SBoxLayer goes into the pLayer process using the rules in Table 2. The entire encryption process using the PRESENT algorithm is represented by Algorithm 1.

Table 2. PRESENT pLayer [12, 16]

i	*P(i)*	*i*	*P(i)*	*i*	*P(i)*	*i*	*P(i)*
0	0	16	4	32	8	48	12
1	16	17	20	33	24	49	28
2	32	18	36	34	40	50	44
3	48	19	52	35	56	51	60
4	1	20	5	36	9	52	13
5	17	21	21	37	25	53	29
6	33	22	37	38	41	54	45
7	49	23	35	39	57	55	61
8	2	24	6	40	10	56	14
9	18	25	22	41	26	57	30
10	34	26	38	42	42	58	46
11	50	27	54	43	58	59	62
12	3	28	7	44	11	60	15
13	19	29	23	45	27	61	31
14	35	30	39	46	43	62	47
15	51	31	55	47	59	63	63

Algorithm 1: PRESENT encryption process [12]

```
generateRoundKeys()
for i = 1 to 31
do
   addRoundKey(state,Ki)
   sBoxLayer(state)
   pLayer(state)
end for
addRoundKey(state,K32)
```

Since its introduction in 2007, the PRESENT algorithm has gained researchers' attention to apply it in various domains and improve its' encryption and decryption performance. As a result, there have been positive and encouraging developments in terms of the studies' results. Furthermore, researchers take various aspects of the algorithm into account to sustain the algorithm in providing security protection for data, especially in the IoT environment. Table 1 lists the studies that improved the PRESENT algorithm and their contributions to the body of knowledge.

Table 3. Summary of related works on the PRESENT encryption algorithm

Study	Description	Contributions
Wang [16]	The study introduced a reduced-round variant of PRESENT and conducted differential cryptanalysis of the algorithm on an 80-bit key.	The study managed to break the algorithm at 16-round, however, not the 31-round.
Yap et al. [7]	This study proposed a block cipher (BC) suitable for electronic product code (EPC) name EPCBC based on the PRESENT algorithm.	The algorithm encrypts a 48-bit block size and 96-bit key and is proven secure against related-key differential attacks.
Z'aba et al. [8]	This study proposed an encryption and decryption circuit of the PRESENT algorithm named I-PRESENT™.	This study provides time and cost savings of circuit usage due to implementing encryption and decryption using the same circuit.
Tang et al. [4]	This study proposed a dynamic S-box to replace the S-BoxLayer in the PRESENT algorithm by applying crossover and mutation in the genetic algorithm concept.	The enhanced PRESENT encryption algorithm has an avalanche effect and the ability to resist differential and linear attacks.
Chatterjee and Chakraborty [9]	The study enhanced the PRESENT algorithm by reducing the encryption round, modifying the Key Register updating technique, and adding a new layer between S-BoxLayer and P-Layer.	The improved algorithms have been able to improve the performance of the original PRESENT.
Jain et al. [17]	The study developed a differential distinguisher algorithm based on a deep learning approach to	The algorithm was able to attack PRESENT encrypted text up to five rounds. However, the

	launch differential attacks on 3-6 rounds of PRESENT encrypted text.	algorithm was not able to attack the complete rounds of the PRESENT encrypted text.
Maro [18]	The study used the Boolean satisfiability problem to evaluate the reliability of the PRESENT encryption algorithm.	It models the encryption process using algebraic analysis of the PRESENT algorithm.
Kwon et al. [5]	The study implemented the PRESENT encryption algorithm on AVR microcontroller that supports Electronic Code Book and Counter.	The study implemented a compact PRESENT algorithm with improved execution time.
Maro [19]	This study compared the power consumption of AES and PRESENT encryption algorithms using ELMO tool.	This study estimates the probability of instructions leakages for the AES and PRESENT implementations.
Chen [6]	The study proposed a key library generation algorithm for PRESENT encryption that was stored in the chip.	The evaluation of the algorithm was tested on image data to demonstrate that the algorithm is suitable for encrypting images within an IoT environment.
Hussam [20]	The study proposed an encryption algorithm based on PRESENT and TWINE for securing data in the cloud environment.	The study proposed a lightweight encryption algorithm that is secure and suitable for protecting data within the cloud environment.
Tao [10]	The study suggested the use of a dynamic key update method for the PRESENT algorithm applied in a vehicular network.	The dynamic key increases the difficulty to detect and decrypt the key to prevent data from being decrypted, stolen and tampered with.
Panahi et al. [11]	This study compared the performance of ten lightweight algorithms for IoT environments. One of them was PRESENT, and they were tested on two microcontrollers.	Among the findings, the study found that PRESENT had the highest encryption execution time of the other nine tested algorithms; nevertheless, it had the lowest encryption throughput.
Sahmi et al. [2]	This study proposed a method to secure message queue telemetry transport protocol using AugPAKE algorithm and PRESENT encryption.	This method provides mutual authentication between the publisher and subscriber and protects the published message's confidentiality and integrity.
Sruthi and Rajasekaran [15]	The study proposed a Signcryption scheme that employed PRESENT to encrypt data and use Elliptic-curve cryptography (ECC) to encrypt the key.	The outcomes of the study suggested that the encryption time was improved using the proposed scheme, which makes it suitable for the resource-constraints environment.

The studies summarized in Table 1 resolve different aspects of using the PRESENT algorithm in various domains. However, many researchers found that block encryption algorithms often face key-differential attacks capable of breaking standard encryption algorithms such as AES-128 and KASUMI [7]. Key-differential attacks is a cryptanalysis technique used on blocks encryption by studying input differences and their relationship to output to retrieve the secret cryptographic key of the encrypted text [17]. Just like other block encryption algorithms, PRESENT also faces the same problem. Thus, this study attempts to solve this problem so that the encryption algorithm is robust, and attackers would not be able to retrieve the encryption key.

2.2 Fingerprint Template Encryption

The fingerprint is widely used as an authentication method due to its unique features [14] and part of the human body facilitating the authentication process. Unlike the smartcards that might be lost or stolen and password forgot due to password overload, a fingerprint is considered convenient [14]. The fingerprint information of a user is called a fingerprint template. Yau [21] and Yau [22] defined a fingerprint template as "a set of stored fingerprint features extracted from the fingerprint of a user. It is stored during the enrollment process to represent the actual owner of the fingerprint." Generally, it is user fingerprint information stored in the database during the enrolment process [14]. The way of recognizing the fingerprint features varies depending on the fingerprint sensors used in capturing the fingerprint images. Nonetheless, the fingerprint features would be stored in a template of hexadecimal string formats. The entire process of fingerprint recognition [21, 22] is illustrated in Figure 2.

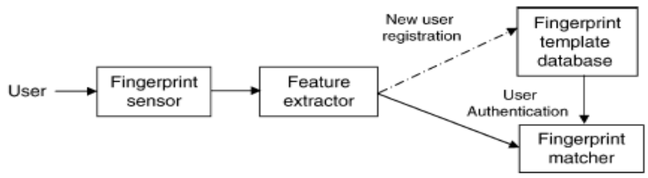

Figure 2. The process of fingerprint recognition [21, 22]

The sensor will scan the finger during the recognition process to obtain a digital image for the fingerprint pattern. Then, the extractor algorithm will identify the fingerprint characteristics. Various methods can be used to determine fingerprint characteristics; however, the popular ones are based on minutia. Next, the fingerprint properties will be converted into binary or hexadecimal string form [14]. Figure 3 shows how the process of transforming a fingerprint image into a template. Fingerprint sensors are available in three types of technology: optical, capacitive and ultrasonic [23]. Each technology uses a different method to recognize a fingerprint, define a fingerprint feature and convert that feature into a template form. However, international standards have provided a fingerprint template format to facilitate the interoperability process by fingerprint sensors' manufacturers. For example, the International Organization for Standardization (ISO) [24] and The American National Standards Institute (ANSI) [25] provide the standard of minutiae template exchange format to allow interoperability of worldwide adoption of different fingerprint recognition systems.

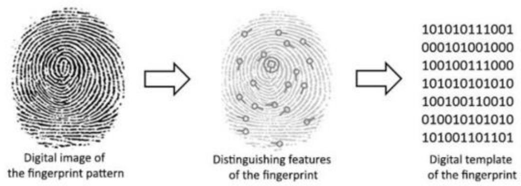

Figure 3. An example of finger minutia [26]

2.3 The Gap

Optical and capacitive sensors are two common types of sensors available in the market. Capacitive sensors are widely used in smartphones [27], while optical sensors are used more frequently in IoT environments [28]. This study focuses on optical sensors and fingerprint templates generated by these sensors to represent user fingerprint features. In particular, this study investigates the template produced by the optical sensor of the AS608 model with the characteristics described in Figure 4. These sensors are readily available in the market at a price of around USD10-20. In addition, it can also be connected to a microcontroller board and programmed to be used as a fingerprint recognition system within an IoT environment.

The template size generated by these sensors is 512 bytes in the form of a hexadecimal string. Figure 5 shows an example of an actual fingerprint template generated by these sensors. This template is in plaintext form, which is an unencrypted template. Thus this template is vulnerable to various threats and security attacks that can cause user identity theft [14]. Thus, this study focuses on encrypting this template to protect its confidentiality, thus avoiding identity theft problems and other potential security issues. Furthermore, this study intends to use PRESENT [12], a lightweight encryption algorithm suitable for IoT environments [1, 2, 11, 13].

PRESENT is a block encryption algorithm that accepts one block's input equals 8 bytes with an 80-bits key. Thus, a fingerprint template with 512 bytes will generate 64 blocks of data to be encrypted with the PRESENT algorithm separately, as shown in Table 3. No padding is needed in this data as all blocks are in the equal size of 8 bytes. This study used a Python's code for the PRESENT algorithm as programmed by Buchanan [29] and key "AC08170000000088EF21". Figure 6 shows the encrypted fingerprint template.

- Supply voltage: 3.3VDC
- Model: JM-101B 1923
- Operating current: <60mA
- Peak current: <70mA
- Fingerprint imaging time: around 1.0 seconds
- Window area: 15.3mm x 18.2mm
- Signature file: 256 bytes
- Template file: 512 bytes
- Storage capacity: 162 templates
- Interface: TTL Serial
- Baud rate: 9600, 19200, 28800, 38400, 57600 (default is 57600)
- Temperature rating: -20C to +50C
- Resolution: 500 dpi
- Backlight colour: Blue
- Full Dimensions: 45 x 20 x 22mm

Figure 4. The AS608 optical fingerprint sensor module with its specification

The main problem in block encryption is that the block's ciphertext generated a pattern of similar block string that leads to a key-differential attack, i.e. the attacker performs a cryptanalysis process to guess the encryption key by comparing the encrypted blocks [16, 30]. In the example of fingerprint template (i.e., Sample 1) in Figure 5 and Table 4, there are twenty times occurrences of "0000000000000000". Hence, this block was encrypted to "8c99215c7a117a6a". Attackers are aware that fingerprint templates would have many occurrences of "0000000000000000" due to the minutia pattern of the fingerprint.

Sample 1

FFFFFFFFFFFFFFFFFF03612BA0017F018B0000000000000000000000000000000000000000000000
000000000000000000000000000024000700AF0001010101010101010101010101010101010101188A
EF01FFFFFFFF02008271A668DE8125A69E76ABAA3E6E35EC9E5FBAC15E6DC2C03E76C6AC1E9C4E54
3E55D8025E3CAC1A9F4C3899DF61C7583FAACCA8FF84526BDFA956693F83386ADC7FC5D6BD8932
93BA8B442A5A8BC7157A87492ADA863528984C5101F52641817F2A4B011F2647981D32BA98D827
3817B9304618792AB7C25634CF0CF637CD98F73BECEF01FFFFFFFF0200823A5159F73250197313C02
C9E1848D81F1C51183B1DD454B91F568F19999A21DE6D95921FFFFFFFFFFFFFFFFFFFFFF00000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000F0000000E91F001000D201200100000141DEF
01FFFFFFFF0200820303552A0001200186000000000000000000000000000000000000000000000
00000000000000000000000000000000D00020081000CCC00F0003FFCF3FFFFFFFFBAAAAAAAAAAA
AAA9655555555555554445500400400000000000000000000000000000000000000000000000000
000000000000000000000000000000001344EF01FFFFFFFF0200824C10AB3E0C

Figure 5. An example of a plain fingerprint template acquired using an optical sensor (Sample 1)

Sample 1 (Encrypted)

21edccff63f05a6ae43258f64ceb8c775cb53b4f270bd69b8c99215c7a117a6a8c99215c7a117a6a8c99
215c7a117a6a8c99215c7a117a6a9493fbe2904b33314b8aa0cf225a0aa5fae5ea622f9051dbd4ecb15
78407a30f8fae90651050e8495e33ebe1d3c2f6703b7e43dfda0cc60da206a40b15fd58bba4a27fa3b0
86c0d89a35e71ec813e658e3d159a4b488461273b818b69ae5eb8dc3db6d05b16fd50dc150d2750ac
109f2828f5562373c83a753aa3cbae73d14a7673c1d235562629b30e45fddefb37c1933c5e9f8429224
5876f2c35fe82bbea7e11028f09ce775875faa1bfefc98bf6f39b06e92124d69f0aa14a0088d9032a730a
3bc9f88759ab7a7b0da64b533c6f77c0306fab57592e0bb163a31636cb28c8c99215c7a117a6a8c9921
5c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7
a117a6aead57b26ff83de1dc1ce0069663e4df2d1cfa9649c74c5b46f9c7c0bc5abb0888cd963092aad4
8bb9a631a473851bfad8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a
42aab723a95669c747a8d59381a164aea83387ded16538f9d62fdf3d54ecd177c908d3095f2b32458c9
9215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6a8c99215c7a117a6aa87d29
81b6726cadb8d408e499c1c8d0

Figure 6. The encrypted fingerprint template (i.e., Sample 1) using PRESENT

Further, it is also known that the fingerprint template contained header information of "FFFFFFFFFFFFFFFF", which encrypted as "21edccff63f05a6a". Therefore, the key-differential analysis could be done quickly by comparing these two blocks. Generally, block encryption is safe from brute-force attacks [31]; however, encrypted fingerprint templates reveal a pattern that facilitates this attack. Hence, there is a need to improve the PRESENT algorithm to be used ideally for fingerprint templates generated using optical sensors, which has been the aim of this study.

3. The Enhanced PRESENT Encryption Algorithm

This study proposes a block pre-processing phase by managing the individual fingerprint template block so that it is suitable to be encrypted with the PRESENT algorithm. It avoids blocks containing the same ciphertext that could lead to the key-differential attacks in which the key of the encryption process is breakable. As mentioned previously, the fingerprint template contains many blocks with zero values that produce many blocks with the same ciphertext (when they are encrypted). They can be recognized that represented zeros, as shown in Table. Apart from that, it is also known that the fingerprint templates have a header block containing the series of "F" sixteen times.

Table 5. An example of ten blocks of a fingerprint template with their corresponding ciphertext

Block	Template	Encrypted	Descriptions
1	FFFFFFFFFFFFFFFF	21edccff63f05a6a	The first block is always known as a header block
2	FF03612BA0017F01	e43258f64ceb8c77	The encrypted block is unique
3	8B00000000000000	5cb53b4f270bd69b	The encrypted block is unique
4	0000000000000000	8c99215c7a117a6a	The encrypted block is unique for the first time in its appearance
5	0000000000000000	8c99215c7a117a6a	The encrypted block is the same as block 4
6	0000000000000000	8c99215c7a117a6a	The encrypted block is the same as blocks 4 and 5
7	0000000000000000	8c99215c7a117a6a	The encrypted block is the same as blocks 4, 5, and 6

8	24000700AF000101	9493fbe2904b3331	The encrypted block is unique
9	0101010101010101	4b8aa0cf225a0aa5	The encrypted block is unique
10	0101010101010118	fae5ea622f9051db	The encrypted block is unique

Based on the ciphertext (i.e. Encrypted column) in Table 5, attackers would analyze the patterns of the block and look at the header block and the repeated ciphertext that would assume them as the zero-blocks. It is known that the PRESENT key is 80 bits; therefore, they can use the available information of the ciphertext along with the functions in the algorithm to get the encryption key like simple mathematical (6), (7), and (8):

$$8\ bytes\ ciphertext = PRESENT_{Encrypt}\ (8\ bytes\ fingerprint\ template, key) \quad (6)$$

$$21edccff63f05a6a = PRESENT_{Encrypt}\ (FFFFFFFFFFFFFFFF, key) \quad (7)$$

$$8c99215c7a117a6a = PRESENT_{Encrypt}\ (0000000000000000, key) \quad (8)$$

The attackers would attempt to find a key that matches the two pairs of plaintext and ciphertext in (7) and (8). Therefore, there is a need to prevent the encrypted blocks from having the same pattern in these two equations, which has been the main objective of this study. In doing this, a method named header and zero block bypass is proposed. The method skips “0000000000000000” and “FFFFFFFFFFFFFFFF” blocks for encryption and maintains the blocks in plaintext. The process begins with dividing the template TB into sixty-four blocks (B1,…B64) as (9). Then, an inspection rule is applied on each block as in (10). The method bypasses the header and the zero blocks while blocks with non-zero values will be encrypted. Algorithm 2 shows the flow of the method, while Figure 7 illustrates the entire process in a flow chart. This method is reversible; hence the decryption process would be in its opposite flow.

$$T_B = [B1, B2, \cdots, B63, B64] \quad (9)$$

$$Z_B = \begin{cases} 1\ if\ Bi\ \neq \text{"0000000000000000"}\ or\ Bi\ \neq \text{"FFFFFFFFFFFFFFFF"} \\ 0\ if\ Bi = \text{"0000000000000000"}\ or\ Bi\ = \text{"FFFFFFFFFFFFFFFF"} \end{cases} \quad (10)$$

Algorithm 2: Header and zero blocks bypass method in the PRESENT encryption process

Input : 512-byte fingerprint template string
Output : 512-byte encrypted fingerprint template string
divideStringIntoBlocks()
for i = 1 to 64
do
  checkForHeader-ZeroBlock()
  appendEncryptedBlock()
  skip
generateRoundKeys()
for i = 1 to 31
do
  addRoundKey(state,Ki)
  sBoxLayer(state)
  pLayer(state)
end for
addRoundKey(state,K32)
appendEncryptedBlock()

In this study, the original PRESENT algorithm was not modified. However, converting the string into an encrypted form was enhanced to avoid specific patterns of known blocks of the header and zeros. As a result, although the encrypted fingerprint template string revealed the header and the zero blocks, it does not jeopardize the security of the other individual blocks of the template. This is because they are unique and do not have any patterns that could break the encryption key. Hence, this study deduces that the fingerprint template string is protected.

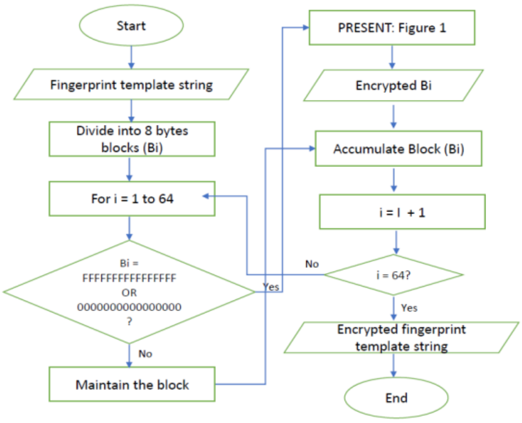

Figure 7. The flow chart of header and zero blocks bypass method

4. Evaluations

4.1 Methods

The evaluation intends to measure the performance of the proposed header and zero block bypass method. Existing studies had proven that the PRESENT algorithm is robust against cryptanalysis; hence, such analysis was omitted. This study instead focuses on how the enhanced algorithm performed in the IoT setting. Three standard evaluation measures were used: encryption time, the percentage of memory usage, and central processing unit (CPU) usage. The encryption time measures the period to transform the plain fingerprint template string into an encrypted form of a string in seconds. The memory usage was the amount of memory that the encryption process used. At the same time, CPU usage is the amount of CPU that the encryption process used. Finally, both were measured in percentage.

4.2 Tools

This study adopted the PRESENT module from www.lightweightcrypto.org [29]. A complete program was written in Phyton, and it imported the module for performing the algorithm. The programme was first written in the development machine to speed up the code development. Then, the code was transferred to the evaluation machine of a Rasberry Pi 3 that represent the IoT setting. Table 3 summarises the specification of the development and evaluation machines.

Table 6. The specification of the development and evaluation machines

Features	Development Machine	Evaluation Machine
OS Name	Microsoft Windows 10 Home Single Language	Raspbian GNU/Linux 10 (buster)
System Type	x64-based PC	32-bit (ARMV7l), Debian V10.2
Processor	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, 2803 Mhz, 4 Core(s), 8 Logical Processor(s)	Broadcom BCM2837 64bit Quad Core Processor
Installed Physical Memory (RAM)	16.0 GB	1Gbytes DDR2
Python Version	Python’s Integrated Development and Learning Environment (IDLE) v3.9.6.	Thonny with Python 3.7.3

4.3 Dataset

This study collected fingerprints from five volunteers using the AS608 optical fingerprint sensor module specified in Figure 4. The sensor generated 512-byte fingerprint templates for all five volunteers. Each template was divided into 64 blocks equally; hence, no padding was used during the encryption process as the block size is equal. The key size of the PRESENT encryption was 10 bytes, represented 80 bits. The key string was “AC08170000000088EF21”. All the fingerprint template strings are stored in text files.

4.4 Procedure

The encryption time, memory and CPU of the encryption process was measured using Python’s psutil module. Two separate Python programmes were coded to perform the encryption of the original PRESENT and the enhanced algorithms. Another two programmes performed the decryption process of the two versions of the algorithm. The psutil functions were used before and after the encryption and decryption process.

5. Results and Discussion

This evaluation focuses on comparing the performance of the enhanced PRESENT with the original encryption algorithm within an IoT environment. First, the study calculated the number of zero blocks in each of the fingerprint templates in the dataset. It ranged between twenty and thirty-two zero blocks out of the sixty-four as listed in Table 7. Therefore, it is necessary to understand the variability of zero blocks in the fingerprint templates so that the performance of the encryption algorithm can be observed accurately.

Table 7. The number of zero blocks in the fingerprint template dataset

Fingerprint Template	Number of “0000000000000000” blocks
1	20
2	32
3	31
4	29
5	26

Next, the study recorded the encryption time, memory usage, and CPU usage for encrypting the five fingerprint templates using the original PRESENT and the enhanced PRESENT. Then, the study calculated the mean and standard deviation (s.d.) of performance measures as listed in Table 8. On average, the original PRESENT encrypted the entire sixty-four blocks of the fingerprint templates in 0.59228 seconds. On the other hand, the enhanced PRESENT took 0.33501 seconds, less time in encrypting them than the original algorithm. However, the memory usage of the enhanced algorithm was 27.5% as compared to 26.9%, which is higher than the original algorithm. Nevertheless, the CPU usage of the enhanced algorithm is much lower than the original algorithm, with 27.7% and 34.6%, respectively.

Table 8. Encryption time and their memory and CPU usages

Fingerprint Template	PRESENT			Enhanced PRESENT		
	Encryption Time (s)	Memory Usage (%)	CPU Usage (%)	Encryption Time (s)	Memory Usage (%)	CPU Usage (%)
1	0.61116	26.4	31.6	0.40373	27.6	33.1
2	0.6033	26.8	29.9	0.30051	27.6	27.0
3	0.57979	26.9	29.7	0.2988	27.5	26.7
4	0.58984	27.3	42.8	0.31304	27.4	25.6
5	0.57733	27.1	39.1	0.35896	27.4	25.9
Mean	*0.59228*	*26.9*	*34.6*	*0.33501*	*27.5*	*27.7*
s.d.	*0.01469*	*0.33912*	*5.97051*	*0.04549*	*0.10000*	*3.09403*

The exact procedure was also conducted on the decryption process, whereby the study recorded the time, memory usage, and CPU usage for decrypting the five fingerprint templates using the original PRESENT and the enhanced PRESENT. Then, the study calculated the mean and standard deviation of performance measures as listed in Table 9. On average, the original PRESENT decrypted the entire sixty-four blocks of the fingerprint templates in 0.59695 seconds. On the other hand, the enhanced PRESENT took 0.34008 seconds, less time in encrypting them than the original algorithm. Furthermore, the memory usage of the enhanced algorithm was 27.7%, about a similar number to the original algorithm, which was 27.4%. The similarity was also observed in the CPU usage, where the enhanced algorithm consumed 29.7%, and the original algorithm had 29.4%.

Table 9. Decryption time and their memory and CPU usages

Fingerprint Template	PRESENT			Enhanced PRESENT		
	Decryption Time (s)	Memory Usage (%)	CPU Usage (%)	Decryption Time (s)	Memory Usage (%)	CPU Usage (%)
1	0.58863	27.4	26.9	0.4219	27.7	35.4
2	0.60461	27.3	33.6	0.29066	27.6	26.3
3	0.59023	27.4	26.4	0.31169	27.8	27.0
4	0.60603	27.4	26.4	0.32273	27.7	33.6
5	0.59527	27.3	33.6	0.35341	27.7	26.2
Mean	0.59695	27.4	29.4	0.34008	27.7	29.7
s.d.	0.00804	0.05477	3.85772	0.05104	0.07071	4.43847

Finally, the study conducted statistical tests using IBM SPSS Statistics 27 to validate the differences in the performance measures. A series of Mann-Whitney U tests were conducted on the performance measure data in Tables 8 and 9, and the results are presented in the last column of Table 10. The test results revealed that the enhanced PRESENT encryption algorithm took a significantly faster encryption time than the original algorithm. However, in turn, it used significantly higher memory usage than the original algorithm. Nevertheless, the different percentage in CPU does not lead to a significant difference. The exact test results were also demonstrated in the decryption process.

Table 10. Mann-Whitney U test on the results

Performance Dimensions	PRESENT	Enhanced PRESENT	Statistics
Encryption Time (s)	0.59228	0.33501	Z=-2.611, p = 0.08, Significant
Encryption Memory Usage (%)	26.9	27.5	Z=-2.627, p = 0.08, Significant
Encryption CPU Usage (%)	34.6	27.7	Z=-1.984, p = 0.056
Decryption Time (s)	0.59695	0.34008	Z=-2.611, p = 0.08, Significant
Decryption Memory Usage (%)	27.4	27.7	Z=-2.685, p = 0.08, Significant
Decryption CPU Usage (%)	29.4	29.7	Z=-0.106, p =1.00

6. Conclusion

This study suggested the block pre-processing phase for enhancing the PRESENT encryption algorithm to protect the secrecy of fingerprint templates within an IoT environment. It performed the process faster, with similar CPU usage and avoiding the block patterns in the encrypted templates that lead to key differential attacks. However, the drawback of the enhanced algorithm is that it increases memory usage. The study can be enhanced as potential future works by increasing the number of fingerprint template samples. Further, other measures could be used to evaluate the performance of the enhanced algorithm.

Acknowledgements

The authors thank the Ministry of Higher Education Malaysia for funding this study under the Fundamental Research Grant Scheme (Ref: FRGS/1/2018/ICT03/UUM/02/1, UUM S/O Code: 14208), and Research and Innovation Management Centre, Universiti Utara Malaysia for the administration of this study

References

[1] I. R. Chiadighikaobi, and N. Katuk, “A scoping study on lightweight cryptography reviews in IoT,” Baghdad Science Journal, Vol. 18, No. 2, pp. 989-1000, 2021.

[2] I. Sahmi, A. Abdellaoui, T. Mazri, and N. Hmina, “MQTT-PRESENT: Approach to secure internet of things applications using MQTT protocol,” International Journal of Electrical & Computer Engineering, Vol. 11, No. 5, pp. 4577-4586, 2021.

[3] L. Sliman, T. Omrani, Z. Tari, A. E. Samhat, and R. Rhouma, “Towards an ultra lightweight block ciphers for Internet of Things,” Journal of Information Security and Applications, Vol. 61, No. Sept., pp. 102897, 2021.

[4] Z. Tang, J. Cui, H. Zhong, and M. Yu, “A random PRESENT encryption algorithm based on dynamic S-BOX,” International Journal of Security and Its Applications, Vol. 10, No. 3, pp. 383-392, 2016.

[5] H. Kwon, Y. Kim, S. C. Seo, and H. Seo, “High-speed implementation of PRESENT on AVR microcontroller,” Mathematics, Vol. 9, No. 4, pp. 374, 2021.

[6] H. Chen, “An enhanced encryption algorithm with key update scheme for Internet of Things,” Journal of Physics: Conference Series, Vol. 1757, No. 1, pp. 012144, 2021.

[7] H. Yap, K. Khoo, A. Poschmann, and M. Henricksen, “EPCBC-a block cipher suitable for electronic product code encryption,” International Conference on Cryptology and Network Security, pp. 76-97, 2011.

[8] M. R. Z’aba, N. Jamil, M. E. Rusli, M. Z. Jamaludin, and A. A. M. Yasir, “I-present™: An involutive lightweight block cipher,” Journal of Information Security, Vol. 5, No. 3, pp. 114-122, 2014.

[9] R. Chatterjee, and R. Chakraborty, “A modified lightweight PRESENT cipher for IoT security,” 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, pp. 1-6, 2020.

[10] H. Tao, “Design and implementation of vehicle data transmission protocol based on PRESENT algorithm,” 2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC), Dalian, China, pp. 968-971, 2021.

[11] P. Panahi, C. Bayılmış, U. Çavuşoğlu, and S. Kaçar, “Performance evaluation of lightweight encryption algorithms for IoT-Based applications,” Arabian Journal for Science and Engineering, Vol. 46, No. 4, pp. 4015-4037, 2021.

[12] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An ultra-lightweight block cipher,” Cryptographic Hardware and Embedded Systems - CHES 2007, Vienna, Austria, pp. 450-466, 2007.

[13] M. K. Hasan, M. Shafiq, S. Islam, B. Pandey, Y. A. Baker El-Ebiary, N. S. Nafi, R. Ciro Rodriguez, and D. E. Vargas, “Lightweight cryptographic algorithms for guessing attack protection in complex Internet of Things applications,” Complexity, Vol. 2021, 2021.

[14] A. Siswanto, N. Katuk, and K. R. Ku-Mahamud, “Chaotic-based encryption algorithm using Henon and logistic maps for fingerprint template protection,” International Journal of Communication Networks and Information Security, Vol. 12, No. 1, pp. 1-9, 2020.

[15] M. Sruthi, and R. Rajasekaran, “Hybrid lightweight Signcryption scheme for IoT,” Open Computer Science, Vol. 11, No. 1, pp. 391-398, 2021.

[16] M. Wang, “Differential cryptanalysis of reduced-round PRESENT,” International Conference on Cryptology in Africa, Casablanca, Morocco, pp. 40-49, 2008.

[17] A. Jain, V. Kohli, and G. Mishra, “Deep Learning based differential distinguisher for lightweight cipher PRESENT,” IACR Cryptol. ePrint Arch., Vol. 2020, pp. 846, 2020.

[18] E. Maro, “Modeling of algebraic analysis of PRESENT cipher by SAT solvers,” IOP Conference Series: Materials Science and Engineering, Vol. 734, No. 1, pp. 012101, 2020.

[19] E. Maro, “Modelling of power consumption for Advanced Encryption Standard and PRESENT ciphers,” IOP Conference Series: Materials Science and Engineering, Vol. 1155, No. 1, pp. 012060, 2021.

[20] M. Hussam, “New lightweight hybrid encryption algorithm for cloud computing (LMGHA-128bit) by using new 5-D hyperchaos system,” Turkish Journal of Computer and Mathematics Education (TURCOMAT), Vol. 12, No. 10, pp. 2531-2540, 2021.

[21] W.-Y. Yau, “Fingerprint Templates,” Encyclopedia of Biometrics, S. Z. Li and A. Jain, eds., pp. 523-528, Boston, MA: Springer US, 2009.

[22] W.-Y. Yau, “Fingerprint Templates,” Encyclopedia of Biometrics, S. Z. Li and A. K. Jain, eds., pp. 679-684, Boston, MA: Springer US, 2015.

[23] Konsyse. “Fingerprint Scanners 101: Capacitive vs. Optical vs. Ultrasonic,”, 2021, URL: https://www.konsyse.com/articles/fingerprint-scanners-101-capacitive-vs-optical-vs-ultrasonic/, [Online; accessed on July 10, 2021].

[24] ISO/IEC 19794-2:2005, “Information technology -- Biometric data interchange formats -- Part 2: Finger minutiae data.,” The International Organization for Standardization, ed., 2005. URL: https://www.iso.org/standard/38746.html, [Online; accessed on July 10, 2021].

[25] INCITS Standard ANSI, “378: 2004-Information technology-Finger minutiae format for data interchange,” The American National Standards Institute, ed., 2004, URL: https://webstore.ansi.org/standards/incits/ansiincits3782004, [Online; accessed on July 10, 2021].

[26] D. Thakkar. “Fingerprint reader technology comparison: Optical fingerprint scanner; capacitive-based fingerprint reader and multispectral imaging sensor,” 2020, URL: https://www.bayometric.com/fingerprint-reader-technology-comparison/, [Online; accessed on July 10, 2021].

[27] M. Masoud, Y. Jaradat, A. Manasrah, and I. Jannoud, “Sensors of smart devices in the Internet of Everything (IoE) era: Big opportunities and massive doubts,” Journal of Sensors, Vol. 2019, pp. 6514520, 2019.

[28] S. Aleksic, “A survey on optical technologies for IoT, smart industry, and smart infrastructures,” Journal of Sensor and Actuator networks, Vol. 8, No. 3, pp. 47, 2019.

[29] W. J. Buchanan. “PRESENT-Asecuritysite,” 2021, URL: https://asecuritysite.com/encryption/present, [Online; accessed on July 10, 2021].

[30] S. Emami, S. Ling, I. Nikolić, J. Pieprzyk, and H. Wang, “The resistance of PRESENT-80 against related-key differential attacks,” Cryptography and Communications, Vol. 6, No. 3, pp. 171-187, 2014/09/01, 2014.

[31] M. N. Alenezi and F.S. Al-Anzi, “A Study of Z-Transform Based Encryption Algorithm”, International Journal of Communication Networks and Information Security, Vol. 13 No. 2, pp. 302-309, 2021.

Table 4. 64 blocks (B1,…B4) of a fingerprint template ready for PRESENT encryption

B1 FFFFFFFF FFFFFFFF	B2 FF03612B A0017F01	B3 8B000000 00000000	B4 00000000 00000000	B5 00000000 00000000	B6 00000000 00000000	B7 00000000 00000000	B8 24000700 AF000101
B9 01010101 01010101	B10 01010101 01010118	B11 8AEF01FF FFFFFF02	B12 008271A6 68DE8125	B13 A69E76AB AA3E6E35	B14 EC9E5FBA C15E6DC2	B15 C03E76C6 AC1E9C4E	B16 543E55D8 025E3CAC
B17 1A9F4C38 99DF61C7	B18 583FAACC A8FF8452	B19 6BDFA956 693F8338	B20 6ADC7FC5 D6BD8932	B21 93BA8B44 2A5A8BC7	B22 157A8749 2ADA8635	B23 28984C51 01F52641	B24 817F2A4B 011F2647
B25 981D32BA 98D82738	B26 17B93046 18792AB7	B27 C25634CF 0CF637CD	B28 98F73BEC EF01FFFF	B29 FFFF0200 823A5159	B30 F7325019 7313C02C	B31 9E1848D8 1F1C5118	B32 3B1DD454 B91F568F
B33 19999A21 DE6D9592	B34 1FFFFFFF FFFFFFFF	B35 FFFFFFFF 00000000	B36 00000000 00000000	B37 00000000 00000000	B38 00000000 00000000	B39 00000000 00000000	B40 00000000 00000000
B41 00000000 00000000	B42 00000000 00000000	B43 00000000 00F00000	B44 00E91F00 1000D201	B45 20010000 00141DEF	B46 01FFFFFF FF020082	B47 0303552A 00012001	B48 86000000 00000000
B49 00000000 00000000	B50 00000000 00000000	B51 00000000 00000000	B52 00000000 00000000	B53 0D000200 81000CCC	B54 00F0003F FCF3FFFF	B55 FFFFFBAA AAAAAAAA	B56 AAAA9655 55555555
B57 54445500 40040000	B58 00000000 00000000	B59 00000000 00000000	B60 00000000 00000000	B61 00000000 00000000	B62 00000000 00000000	B63 1344EF01 FFFFFFFF	B64 0200824C 10AB3E0C