Computer Security Symposium 2021
26 - 29 October 2021

Cubicle: A Family of Space-hard Ciphers for IoT

Rentaro Shiba[1,a)] Ravi Anand[1] Kazuhiko Minematsu[2] Takanori Isobe[†1,†2]

Abstract: As IoT has increasingly evolved in recent years, it has become more important to ensure security on IoT devices. Many of such devices are under the threat of side-channel attacks. To protect from the threat, the cryptographic implementation which offers security against side-channel attacks is important. However, such cryptographic implementations require a large number of clock cycles to execute and can only be resistant to certain types of side-channel attacks. In this paper, we aim to provide secure cryptographic implementation against side-channel attacks using space-hard ciphers which are secure in the white-box setting. Specifically, we first show that using a space-hard cipher is secure against side-channel attacks. After that, we propose a new family of space-hard ciphers dubbed Cubicle. This cipher has opted for the implementation on processors of ARM Cortex-M which are widely used for IoT devices. We show that Cubicle is secure against various attacks including side-channel attacks and the performance is about 4 to 5 times higher than one of the existing space-hard cipher, SPACE in devices powered by the Cortex-M processor.

Keywords: IoT, ARM, space-hard, block cipher, side-channel resistance

1. Introduction

During the past decade, we have witnessed a dawn of a new era of the Internet of Things (IoT). Generally speaking, the IoT describes a network of physical devices equipped with various sensors, software, and other technology which allows these devices to gather, process, and modify information to provide new services over the internet. These devices are able to recognize events and changes in the surroundings and act and react mainly without any human interaction. However, such huge deployments of IoT devices, wearable, sensory, etc., will definitely pose various security, privacy, and trust threats which put these devices at high risk and thus harming the end-users.

With an increase in security concerns and awareness, proper countermeasures are implemented in some devices to avoid easy access to the attacker via attack surfaces like SPI, I2C, UART, and JTAG. Unfortunately, even these countermeasures against hardware attacks cannot assure a secure system, especially the famous hardware attack called the Side-Channel Attacks (SCA).

Side-channel attacks are easy to implement while being extremely powerful against cryptographic implementations. These attacks exploit the information leakages in the system, and these leakages are known as side-channels. Instead of targeting the standard cryptographic algorithms, these attacks target their implementation on the physical devices to recover the secret parameters by measuring and analyzing the leaked information like power analysis, timing analysis, electromagnetic analysis, etc.

In this work, we aim to provide a secure cryptographic implementation in the grey-box setting, which is a security model that allows attackers to execute side-channel attacks from the perspective of space-hardness. Space-hardness is the security requirement to evaluate the difficulty of code lifting attacks, which is introduced by Bogdanov and Isobe [1] in ACM CCS 2015. Since space-hard ciphers which are secure against code lifting attacks in terms of the space-hardness are secure against key extraction in the white-box setting, the attackers can not extract the secret key in also the grey-box setting.

Implementation of existing space-hard ciphers such as SPACE and SPNbox [2] basically requires SIMD instructions for parallel table-lookup, polynomial multiplication, and shuffle operation. However, not all processors of IoT devices have these instructions to perform such complex operations. One of a family of ARM processors, Cortex-M does not support such complex instructions though it is widely used for IoT devices. Thus, if we were to implement these space-hard ciphers without such instructions, it would take a large number of clock cycles to execute. In addition, space-hard ciphers with their large size of the input to the table are difficult to implement on IoT devices from the perspective of the flash size.

To solve these problems, we first show the security of space-hard ciphers in environments where occurs leakage of the table information. After that, we propose a new family of space-hard cipher opted for such environments. In detail,

[1] University of Hyogo, Japan
[2] NEC Corporation, Japan
[†1] Presently with National Institute of Information and Communication Technology, Japan
[†2] Presently with PRESTO, Japan Science and Technology Agency, Japan
[a)] rentaro.shiba@gmail.com

our contributions are summarized as follows.

Proof of the bound for the table-leakage To prove the security of space-hard ciphers in the grey-box, we show the bound of the table-leakage caused by code lifting attacks of grey-box attackers. Based on that bound, we can estimate the number of times the table of space-hard ciphers needs to be accessed to provide enough leakage to recover the quarter of the total entries. This enables us to estimate how often the table should be updated when the space-hard cipher is practically used on IoT devices.

Proposal of a new family of space-hard ciphers We propose a new family of space-hard ciphers dubbed Cubicle opted for the implementation in processors of the ARM Cortex-M. Cubicle is a 4-line target-heavy Feistel cipher, where each input is consists of 4 32-bit words. We show that Cubicle can provide sufficient security against various types of well-known black-box attacks, key extraction, and code-lifting in the white-box. In addition to these evaluations, we also evaluate the security of Cubicle in the grey-box based on the table leakage bound that we derived. For the practical application of Cubicle, we show the frequency of nonce update when Cubicle is used as a key derivation function. Besides, we evaluate the performance of Cubicle and SPACE in some development boards powered by the Cortex-M3 or the Cortex-M4 processor. The result shows that the performance of Cubicle is much higher than that of SPACE.

2. Space-hard Ciphers

2.1 Space hardness

While protecting the confidentiality of secret keys in the white-box environment is the major goal of white-box cryptography, the main goal of a white-box attacker is to extract the secret key given the full control over the execution environment of a cipher. Thus the typical security requirements in this setting are follows: *key extraction security* and *code lifting security*. Key extraction security implies that the extraction of the secret key should be computationally hard. Code lifting is a threat in which the attacker can isolate the program code from the implementation environment and can use it as a larger effective key. The success of a code lifting attacks has the almost same advantage as key extraction, i.e. the attacker can encrypt/decrypt any plaintext/ciphertext.

To evaluate the difficulty of code lifting attacks, Bogdanov and Isobe [1] introduced a new security notion called *(M,Z)-space hardness*. Space hardness measures the difficulty of compressing a white-box implementation of any cipher and quantifies security against code lifting by the amount of code that needs to be extracted from the implementation by a white-box adversary to maintain its functionality. This notion of security assesses the difficulty of isolating the code form the execution environments by the amount of data required to do so and thus, it covers a wide class of attackers.

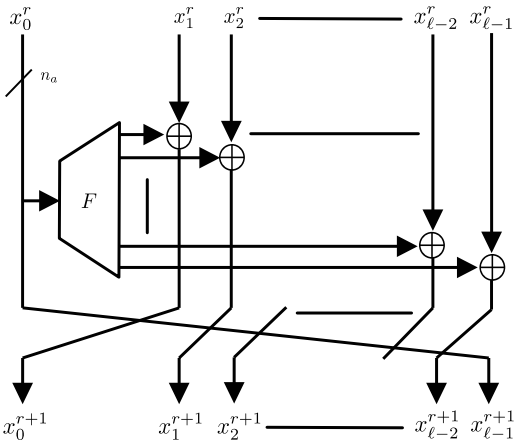


Fig. 1: One round of SPACE
($n_a \in \{8, 16, 24, 32\}$, $\ell = 128/n_a$)

Definition 1 ((M, Z)-space hardness [1])**.** The implementation of a block cipher E_K is (M, Z)-space hard if it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability of more than 2^{-Z} given any code (table) of size less than M.

(M, Z)-space hardness estimates the code size M to be isolated from white-box environments to encrypt (decrypt) any plaintext (ciphertext) with a success probability larger than 2^{-Z}. For instance, if the white-box implementation of a block cipher E_K requires a look-up table of total size T, then $(T/4, 128)$-space hardness implies that if an adversary has obtained information equivalent to one fourth of T, it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability higher than 2^{-128}.

2.2 SPACE

SPACE is a family of space-hard ciphers proposed by Bogdanov and Isobe [1] at ACM CCS 2015. SPACE employ a target-heavy generalized Feistel structure as the underlying construction and the internal F-function contains AES-128 encryption. The overview of SPACE is shown in Table 1.

In the white-box implementation, the AES-128 encryption of F-function is implemented as the keyed lookup table. Therefore, the security against the key extraction in the white-box setting reduces to the well-studied problem of the security against the key recovery in the standard black-box setting. Besides, for the code lifting attacks, SPACE has a $(T/4, 128)$-space hardness.

3. Security Model

3.1 Grey-box/side-channel attack

The main assumption of computer systems is that processed secrets are inaccessible for an attacker due to security measures in software and hardware. However, side-channel attacks allow an attacker to still deduce the secrets by observing certain side effects of a computation. A side-channel is where information leaks accidentally via some medium that was not designed or intended for communication. These attacks can be broadly classified into two axes:

Invasive vs Non-invasive: Invasive attacks require opening the device under attack. This usually refers to the chip level, where depackaging of the chip might be needed. Invasive attacks can be further divided into semi-invasive and fully-invasive attacks. The difference is that with semi-invasive attacks the passivation layer of the chip stays intact whereas with fully invasive attacks, the chip is further deprocessed depending on the requirements of the particular attack.

Active vs Passive: While passive attacks restrict themselves to only observe the device's behavior, an active attack also manipulates the device's operation e.g. by injecting various types of faults (electrical, optical, etc.) or by employing glitching attacks.

In the light of these attacks, attempts are made to implement the cryptographic algorithms in such a way that the cost of retrieving the key for an adversary is too high to interest him/her. Many side-channel attacks have been devised and equally many countermeasures have been published. These countermeasures against side-channel attacks can be categorized into the following two families. The first one, called *masking*, modifies the algorithm in order to make the sensitive variables manipulated independently from the secret key. The second one, called *hiding*, covers the processing of the algorithm by noise in order to make the exploitation of the observed signal too difficult.

However, the countermeasures are theoretically very efficient but are very hard to implement in practice. These countermeasures may guarantee the security of implementation against a specific attack, for example, first-order power analysis, but it might leave the implementation very weak against other attacks which were not covered by the countermeasure. Also, the overhead of performance required by these countermeasures is not negligible. For example, most of the countermeasures make use of randomness. Hiding countermeasures might require randomness to shuffle executions, and insert unpredictable delays. Masking countermeasures require randomness to share variables such that the manipulations are uncorrelated to the secret. Moreover, operations manipulating masked values induce an overhead in terms of needed randoms during the transcoding part. The production of these random bits is complex and costly.

3.2 Space-hard ciphers in Grey-box settings (Table Leakage)

In this section, we discuss the security of space-hard ciphers in the grey-box setting. Space-hard ciphers are constructed to make code lifting attacks difficult in the white-box setting. They are table-based ciphers whose tables, T, are composed of some pairs of input to an underlying block cipher such as AES and the corresponding output. This table is used by the ciphers as a large secret key. The ultimate goal of an adversary in the grey-box setting is to deduce some values of the secret table T from a series of information leaked from the table.

Side-channel attacks such as differential fault attack (DFA) is not feasible in space hard ciphers. The tables in these ciphers are composed of an underlying block cipher such as AES, and the internals of the underlying block cipher are not accessible to the adversary. Thus, any fault injection attack reduces to a differential attack on a small block cipher in the black-box setting. Since the underlying cipher is considered secure against a differential attack in the black-box setting space-hard ciphers are secure against differential fault attacks.

However, the side-channel attacks against key-dependent table look-ups can be mounted on these ciphers. Information may leak whenever bits of data are accessed and computed upon. The leaking information actually depends on the particular operation performed, and, more generally, on the configuration of the currently active part of the computer. Some of the operations which are commonly susceptible to side-channel leakage are: Data access to/from registers, rotations, and shifts, data-dependent offsets, bitwise boolean operations. The specific information leaked depends on the actual measurement made. Different measurements can be chosen (adaptively and adversarially) at each step of the computation. These measurements can then be analyzed to retrieve any sensitive information of the table.

3.3 Bound of table leakage

A general construction of space-hard cipher $\mathcal{C}$ can be defined as: an n-bit block cipher that encrypts/decrypts using key dependent tables T. Let us consider that the input size of the table be n_a and the output size be n_b. The number of entries in the table T will be 2^{n_a}. Let the number of rounds be R and each round uses the identical table T. Let us consider that this space-hard cipher $\mathcal{C}$ is running on a leaking device. A side-channel attack will try to exploit the leakage resulting from an intermediate computations in order to recover (a part of) of the entries of the table. Space hard ciphers makes use of the tables T as a secret key and recovery of entries of the table would mean recovery of the secret key. Thus we call the leakage in this case as *table leakage*. Some information of the entries of the table are leaked whenever the table is accessed. We now try to estimate the number of table access needed to gather the amount of leakage needed to recover a sufficient part of the complete table. An information theoretic bound of *table leakage* is

$$\#\textit{Table leakage} \leq \textit{Used Entries in execution} \qquad (1)$$

Let $f(p)$ be a function that incorporates a lookup table T and some further transformations of the value read from the table. The parameter p represents plaintext input and may be extended to a sequence of parameters without significant change in the method of attacks. Let

$$r = f(p)$$

The problem that has to be solved by an adversary is to find the content of the unknown or modified lookup table T just by observing the side-channel information. The value

of parameter p is known to the attacker, while the result r is unknown, since it is further modified during algorithm execution. While encrypting the plaintext p, the cipher needs to access the table and during each access few entries of the table is computed on. Let us assume that the adversary can retrieve one table entry every time the table is accessed while encrypting p. This is the best-case scenario for the adversary and the worst case for the cipher in terms of grey-box security. If we can secure the device in this scenario we can say that the device is side-channel resistant, no matter which side-channel attack is used. We now try to estimate the number of access to the table needed to provide enough leakage to recover $T/4$ entries of the table, i.e. 2^{n_a-2} entries. We have the following lemma.

Lemma 2. For every q access to the table T, the number of entries used can be estimated as $2^{n_a}(1-(1-\frac{1}{2^{n_a}})^q)$.

Proof. For q table accesses, an i^{th} entry of of the table is used with a probability of $(1-(1-\frac{1}{2^{n_a}})^q)$. Since there are 2^{n_a} entries the table, we obtain $2^{n_a}(1-(1-\frac{1}{2^{n_a}})^q)$. □

From Lemma 2 and Eqn. 1, we have the following:

$$\#\textit{Table Leakage} \leq 2^{n_a}(1-(1-\frac{1}{2^{n_a}})^q)$$

We can now estimate the number of times the table needs to be accessed to provide enough leakage to recover $T/4$ entries of the table, i.e. 2^{n_a-2} entries. From Lemma 2 replacing q by 2^{n_a-2} gives the estimate of table access required to retrieve $T/4$ entries of the table. Let us use the cipher SPACE as an example here. The number of table entries in SPACE-8 and SPACE-16 is 2^8 and 2^{16} respectively. Now from Lemma 2, the number of table access required to retrieve $T/4$ entries will be $\approx 2^6$ for SPACE-8 and $\approx 2^{14}$ for SPACE-16. To provide a perspective, consider that the IoT devices which use SPACE-16 perform 2^8 executions per day then, the table needs to be updated after 2^6 days. Thus if the look-up tables are updated after 2^6 or 2^{14} table access, the adversary will have to go through all the hard work to obtain the entries of a new table rendering all the previously obtained results useless. Hence we can assume that this strategy of updating the table makes the cipher implementation side-channel resistant.

3.4 Open problem of using existing space-hard ciphers in the grey-box

In 3.3, we showed the bounds of table leakage and concluded that using space-hard ciphers is an effective countermeasure against side-channel attacks. However, there several problems with using space-hard ciphers on IoT devices. Existing Space-hard ciphers such as SPACE and SPNbox are designed for software implementations on high-performance processors which support various types of SIMD instructions. Therefore, their shuffle operations can be implemented effectively by such instructions. In particular, the implementation of SPNbox requires many SIMD instructions for their table-lookup operations and matrix multiplications.

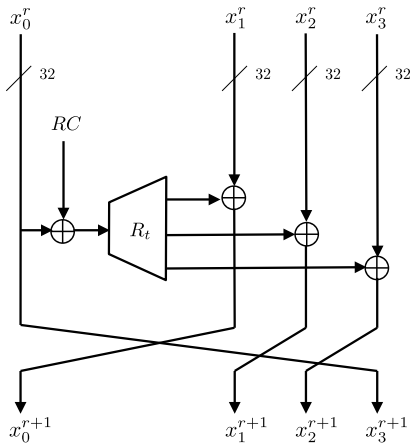


Fig. 2: One round of Cubicle

The implementation of SPACE-8, -16, -24 which are the variants of SPACE also requires SIMD instructions for their shuffle operation. However, there exist many IoT devices with processors that do not have such instructions. If we implement existing space-hard ciphers on such devices, the encryption/decryption may take a lot of time. Also, the table size of SPACE-8 and SPACE-16 is 3.84KB and 917.5KB respectively. For some IoT devices, these sizes of the table might turn out to be too expensive to be updated regularly. This leaves an area of improvement in designing space hard ciphers which have significantly reduced table sizes. In the next section, we propose a new family of space-hard ciphers with a SPACE-like construction dubbed Cubicle to deal with these problems when we implement space-hard ciphers in such environments.

4. Specification of Cubicle

Cubicle is a family of space-hard ciphers: Cubicle-8, Cubicle-16. Both variants accept a 128-bit block input and employ a 4-line target-heavy generalized Feistel structure. The overview of Cubicle is shown in Figure 2.

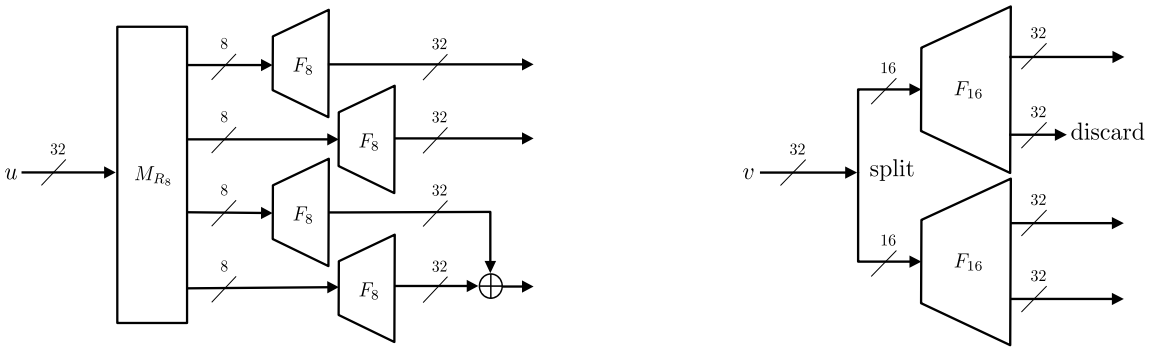
The function $R_t : \{0,1\}^{32} \rightarrow \{0,1\}^{96}$ $(t \in \{8,16\})$ contains more than once application of expanding function. We denote the 128-bit input state of the r^{th}-round by $X^r = \{x_0^r, x_1^r, x_2^r, x_3^r\}$, $x_i^r \in \{0,1\}^{32}$, $(i = 0, \cdots, 3)$. Cubicle updates the state in each round as following:

$$X^{r+1} = (R_t(x_0 \oplus RC) \oplus (x_1^r || x_2^r || x_3^r)) || x_0^r.$$

where $||$ denotes the concatenation and RC is the round constant. We define the round constant to be $RC = r$ for Cubicle-8 and $RC = (RC + 0\text{x}00020002)\%0\text{xf}000\text{f}000$ where the initial value of RC is 1 for Cubicle-16. The specification of R_t is different for $t = 8$ and $t = 16$. The overview of R_8 and R_{16} are shown in Fig. 3. R_8 is used for Cubicle-8 and R_{16} is used for Cubicle-16. For the 32-bit input $u = (u_0, u_1, u_2, u_3)$, $u_i \in \{0,1\}^8$ $(i = 0, \cdots, 3)$, R_8 is defined as follows:

$$R_8(u) = F_8(z_0) || F_8(z_1) || (F_8(z_2) \oplus F_8(z_3))).$$

where z_i are defined as



(a) R_8 **(b)** R_{16}

Fig. 3: The overview of R_t for $t \in \{8, 16\}$

$$M_{R_8} \cdot u^T = (z_0\, z_1\, z_2\, z_3)^T. \quad (2)$$

$F_8(\cdot)$ is the expanding function $F_8 : \{0,1\}^8 \to \{0,1\}^{32}$. M_{R_8} is a almost MDS defined as

$$M_{R_8} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

This matrix is the same as used in midori. On the other hand, for the 32-bit input $v = (v_0, v_1)\, v_i \in \{0,1\}^{16}\, (i = 0, 1)$, R_{16} is defined as follows:

$$R_{16}(v) = \mathrm{msb}_{32}(F_{16}(v_0))||F_{16}(v_1)$$

$F_{16}(\cdot)$ is the expanding function $F_{16} : \{0,1\}^{16} \to \{0,1\}^{64}$. Here $\mathrm{msb}_{32}(\cdot)$ denotes the most significant 32 bits of the input.

F_8 and F_{16} are restricted to be implemented by the table-lookup operation. For the 8-bit input x and 16-bit input y, the functions $F_8(x)$ and $F_{16}(y)$ are defined as follows:

$$F_8(x) = \mathrm{msb}_{32}(\mathrm{AES}_k(C_0||x)),$$
$$F_{16}(y) = \mathrm{msb}_{64}(\mathrm{AES}_k(C_0||y)).$$

where AES_k is AES-128 that takes $k \in \mathbb{F}_2^{128}$ as the secret key and C_0 is a all zero vector to pad the input for AES_k.

5. Design Rationale

5.1 Designing a space-hard cipher for ARM Cortex-M processors

Cubicle is designed to be implemented in the IoT device powered by a processor of the Cortex-. The Cortex-M is a family of 32-bit RISC processors licensed by ARM intended for controlling microcontrollers. They are designed to be low-cost and energy-efficient while remaining high performance in applications for automotive, industrial control, IoT, or other resource-constrained devices. In this paper, we specifically focus on the implementation on processors of the Cortex-M3 and the Cortex-M4. The goal of designing Cubicle is to realize the high performance and sufficient security against various attacks in processors of the Cortex-. To achieve this goal, we consider the following features of the Cortex-M and design requirements.

32-bit registers Processors of the Cortex-M3 and the Cortex-M4 have 16 32-bit registers. They support the ARMv7-M architecture and the Thumb-2 instructions. Although processors of the Cortex-M4 also support ARMv7E-M architecture and SIMD instructions, there is no complex shuffle instruction in this architecture. Therefore, we have to eliminate the transfer of data between different registers so that we do not need additional instructions for such operations. In order to implement the space-hard cipher without any additional instructions except for table-lookup, bitwise and arithmetic operations, we employ word-wise operations for the design. A word is a 32-bit data in processors of the Cortex-M.

Pipelining load instructions Bitwise and arithmetic operations take 1 cycle on this architecture while a load operation (ldr) usually takes 2 cycles. However, once ldr is executed, the consecutive execution referencing the same table as the first execution takes only 1 cycle when the target data has no dependency with the result of previous operations. Since the table-lookup operation can be executed by ldr, consecutive table-lookup operations can take advantage of this feature. Therefore, we aim to use one or more table-lookup operations in one round.

Utilizing barrel-shifting registers A distinguishing feature of the AR architecture is the availability of barrel-shifting registers. This means that we can execute rotation or shift operations without any additional cost. We consider making the most of this feature for improving the security of Cubicle.

Small table size The size of flash and RA of devices powered by processors of the Cortex-M is not enough to store the table of space-hard ciphers which requires a large number of table entries like SPACE-24 and -32 or SPNbox-24 and -32. Thus, we have to restrict the input size of the table to 8-bit or 16-bit. The number of entries of the 8-bit input table is 2^8. This can be stored in the RAM of such devices. In contrast, the number of entries of the 16-bit input table is 2^{16}. This is too large to be stored in the RAM of such devices, but there exist

some devices which can store the 16-bit input table in their flash.

AES-based construction The table of space-hard ciphers is ensuring their security under the assumption that the underlying block cipher used for the table generation is sufficiently secure. To ensure sufficient security against various cryptanalysis, we employ AES as the underlying block cipher of Cubicle since no efficient key recovery of AES has been published so far even though a lot of effort has been put into the cryptanalysis over a long period.

5.2 General construction

5.2.1 Feistel structure

We employ a 4-line target-heavy Feistel structure where the one line is a word for Cubicle. Since each word is stored in one register and there is no transfer of data between different registers, the shuffle operation can be implemented without any additional instructions.

5.2.2 Reducing the table size

Hence our target devices often have limited flash and RA sizes as we mentioned in 5.1, we have to restrict the input size of the table to 2^8 for Cubicle-8 and 2^{16} for Cubicle-16. In order to combine this transformation by 8-bit or 16-bit input table-lookups with 4-line target-heavy Feistel structure into the construction of Cubicle, we split one word into t-bit to utilize as inputs of F_8 for $t = 8$ or F_{16} for $t = 16$. F_8 and F_{16} are generate the 32-bit output and the 64-bit output, respectively. Thereby, we succeed to restrict the table size 32×2^8 bit ($\approx$ 1KB) for F_8 of Cubicle-8 and 64×2^{16} bit ($\approx$ 524.3KB) for F_{16} of Cubicle-16 while the table size of SPACE-32, whose the construction resembles Cubicle is 96×2^{32} bit ($\approx$ 51.5GB). In addition, we can also say that the table sizes of Cubicle-8 and Cubicle-16 are smaller than the table sizes of SPACE-8 ($120 \times 2^8 = 3.84$KB) and SPACE-16 ($112 \times 2^{16} \approx 917.5$KB), respectively.

5.3 Round function

5.3.1 Consecutive table-lookups

As described in 5.1, processors of Cortex- have the feature that pipelining load instructions (ldr) can reduce the latency per ldr instruction from 2 to 1 if we use the same table consecutively. We designed Cubicle so that one or more than table-lookup operations can be executed in one round to take advantage of this feature while SPACE can not utilize this feature because of the data dependency between different rounds. Specifically, we use four consecutive table-lookups by 4 8-bit inputs for Cubicle-8 and two consecutive table-lookups by 2 16-bit inputs for Cubicle-16.

5.3.2 Matrix multiplication

The matrix multiplication used in Cubicle-8 to improve the security against differential/linear cryptanalysis which ensure the minimum branch number is 4 can be implemented by utilizing barrel-shifting registers. From the equation (2), the following can be obtained:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} u_1 \oplus u_2 \oplus u_3 \\ u_2 \oplus u_3 \oplus u_0 \\ u_3 \oplus u_0 \oplus u_1 \\ u_0 \oplus u_1 \oplus u_2 \end{pmatrix} \quad (3)$$

In software implementation of Cubicle, a single register stores one value each for u_0, u_1, u_2 and u_3. Thus, the result of (3) can be expressed by the XORing three right-rotated registers. Namely, this matrix multiplication can be expressed by the XORing the following three registers: the 24 bits right-rotated register, the 16 bits right-rotated register, and the 8 bits right-rotated register. Therefore, this operation can be implemented by 2 XORs and 3 rotate right. By using barrel-shifting registers, we can implement this operation by only 2 XORs without additional isolated instructions to execute the rotate right.

6. Security Analysis

6.1 Security in the white-box

Key extraction

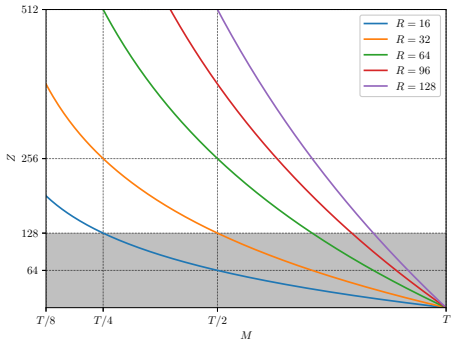
In the white-box model, attackers can fully access the internal values in any round of the algorithm. Extracting the secret key K from the keyed table of F_8 or F_{16} is equivalent to performing key recovery attacks on the underlying block cipher E_K used to generate those tables. As we mentioned in 5.1, we assume that AES is used as the E_K, and no effective key recovery attack in the black-box setting has yet been proposed against it. Therefore, even in the white-box setting, extracting the key from F_8 or F_{16} where AES is used as E_K to generate the entries, is as computationally difficult as recovering the key in the black-box setting.

Code lifting

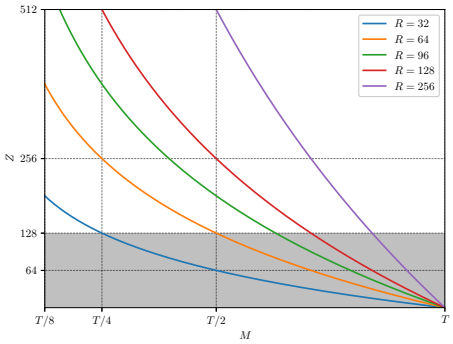
Code lifting security is an important property in environments where attackers can cause a table leakage. Let the number of extracted entries by an attacker be i, j where $i \leq 2^8$ and $j \leq 2^{16}$. The probability that the lookup of extracted entries with random input will succeed is $i/2^8$ for Cubicle-8 and $j/2^{16}$ for Cubicle-16. In Cubicle-8 and Cubicle-16, table-lookup operation execute four times and two times in a round, respectively. Therefore, the probability that the attacker will succeed to encrypt/decrypt randomly chosen plaintext/ciphertext are $(i/2^8)^{4R}$ for Cubicle-8 and $(j/2^{16})^{2R}$ for Cubicle-16 where R is the number of rounds. Fig. 4 shows the relation between M and Z in terms of space hardness for Cubicle. This results shows Cubicle-8 and Cubicle-16 achieves $(T/4, 128)$-space hardness with $R = 16$ and $R = 32$, respectively.

6.2 Security in the black-box

The result of the evaluation of the security against black-box attacks by the bit-based MILP method is summarized in Table 1. We evaluated the security against differential, linear, impossible-differential, and integral cryptanalysis. The entries of the table imply the number of rounds that requires to satisfy sufficient security against the corresponding



(a) Cubicle-8



(b) Cubicle-16

Fig. 4: The evaluation of compression attack in terms of (M, Z)-space hardness

Table 1: Summary of the result of security analysis in the black-box

Variant	Method			
	Differential	Linear	Impossible	Integral
Cubicle-8	40	52	10	11
Cubicle-16	36	40	12	11

attacks.

6.3 Security in the grey-box

Key extraction

Since Cubicle is secure against key extraction in the white-box setting, it is also secure against that in the grey-box setting, which is the weaker setting than the white-box setting. Therefore, we can say that the key extraction by various side-channel attacks such as cache attacks [3], [4], [5], differential fault attacks [6], [7], [8], and differential computation attacks [9] for Cubicle is computationally infeasible.

Table recovery attack

As described above we find that the side-channel attacks are computationally infeasible in the case of Cubicle, however using Lemma 2 we compute the bound on number of table accesses before updating the table to make Cubicle side-channel resistance.

The number of table entries in Cubicle-8 and Cubicle-16 are 2^8 and 2^{16} respectively. Using Lemma 2, the number of table access required to retrieve $T/4$ entries will be $\approx 2^6$ for Cubicle-8 and $\approx 2^{14}$ for Cubicle-16. Thus if the lookup tables are updated after every 2^6 or 2^{14} table accesses then Cubicle can be assumed to be side-channel resistant.

While this values is same as that of SPACE-8 and SPACE-16, the significant difference is the size of the table that needs to be updated, 1KB for Cubicle-8 instead of 3.84KB for SPACE-8 and 524.8KB for Cubicle-16 instead of 917.5KB for SPACE-16.

6.4 Recommended number of rounds

We summarize the rounds that are secure against the corresponding attacks based on our security analysis in Table 1. In our implementation, we choose the number of rounds for Cubicle to 64 in both variants in terms of the result of security analysis and performance.

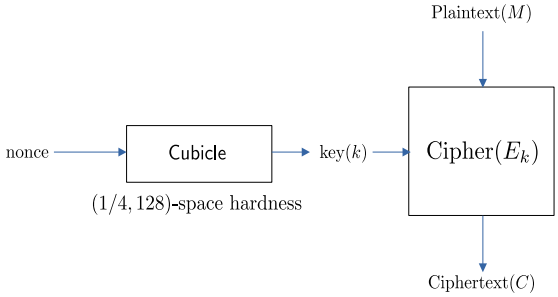


Fig. 5: Key derivation function

7. Practical Application

The strategy of updating the lookup table as described in Section 3.3 can be applied to construct a key derivation function as shown in the Figure 5. Consider that a cipher E_k, say AES for example, takes the key k as input to encrypt the plaintext M to ciphertext C. To prevent certain vulnerabilities of E_k against the certain adversary, say to protect AES-128 against differential fault attack, it becomes necessary to update the secret key k at certain intervals. Updating the secret key may not be always easy. In these cases the key derivation function defined in Figure 5 can come in handy.

From the result of the evaluation of Cubicle in the previous section, the number of table access required to retrieve $T/4$ entries will be $\approx 2^6$ for Cubicle-8 and $\approx 2^{14}$ for Cubicle-16. Thus, If Cubicle-8 is used as a key derivation function, and the key k needs to be updated after:

- every encryption, then update the nonce after 2^6 encryptions
- every hour, then update the nonce after $2^{1.4}$ days
- daily, then update the nonce after 2^6 days

On the other hand, if Cubicle-16 is used as a key derivation function, and the key k needs to be updated after:

- every encryption, then update the nonce after 2^{14} encryptions
- every minute, then update the nonce after $2^{3.5}$ days

Table 2: Specification of boards

Board	CPU	SRAM	Flash
F103RB	Cortex-M3	20KB	128KB
F411RE	Cortex-M4	128KB	512KB
F207ZG	Cortex-M3	128KB	1MB
F429ZI	Cortex-M4	260KB	2MB

Table 3: Performance evaluation (cycles)

Board	Algorithm (table size)			
	SPACE-8 (3.48KB)	SPACE-16 (917.5KB)	Cubicle-8 (1KB)	Cubicle-16 (525.3KB)
F103RB	14165	-	2614	-
F411RE	12654	-	2614	-
F207ZG	14760	10186	3196	2429
F429ZI	14760	10058	3196	2429

- every hour, then update the nonce after 2^9 days
- daily, then update the nonce after 2^{14} days

8. Performance Evaluation

In our performance evaluation, we compare the performance of Cubicle to white-box implementations of SPACE-8 and SPACE-16. We use four STM32 Nucleo boards: Nucleo-F103RB, -F411RE, -F207ZG, -F429ZI which are powered by the Cortex-M3 or the Cortex-M4 Processor for performance evaluation. The specification of these boards and the results of our evaluation are shown in Table 2 and Table 3, respectively. From the table, we can observe that the performance of Cubicle is about 4 to 5 times higher than that of SPACE. We set the number of rounds for SPACE-8 and SPACE-16 to 300 and 128, respectively, as recommended by the designers.

9. Conclusion

In this work, we show the bound of the table leakage and concluded that the implementation of space-hard ciphers on IoT devices is secure against various attacks in the grey-box setting. In addition, we propose a new family of space-hard ciphers dubbed Cubicle, which can be implemented easily and executed faster than existing space-hard ciphers on devices powered by ARM Cortex-M processors, which one of widely used processors for IoT devices. For practical application of Cubicle, we show the frequency of nonce update when we use Cubicle as a key derivation function. We hope that the results of this work would be a contribution to the practical use of space-hard ciphers on IoT devices to protect against various side-channel attacks.

Acknowledgments Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031, Grant-in-Aid for Scientific Research(B)(KAKENHI 19H02141) and SECOM science and technology foundation.

References

[1] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pp. 1058–1069. ACM, 2015.

[2] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, Vol. 10031 of *Lecture Notes in Computer Science*, pp. 126–158, 2016.

[3] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pp. 897–912. USENIX Association, 2015.

[4] Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. S$a: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pp. 591–604. IEEE Computer Society, 2015.

[5] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! A fast, cross-vm attack on AES. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*, Vol. 8688 of *Lecture Notes in Computer Science*, pp. 299–319. Springer, 2014.

[6] Eloi Sanfelix. Unboxing the white-box practical attacks against obfuscated ciphers. 2015.

[7] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, Vol. 3357 of *Lecture Notes in Computer Science*, pp. 227–240. Springer, 2004.

[8] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, Vol. 4876 of *Lecture Notes in Computer Science*, pp. 264–277. Springer, 2007.

[9] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, Vol. 9813 of *Lecture Notes in Computer Science*, pp. 215–236. Springer, 2016.