IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 4, pp. 587–617. DOI:10.46586/tches.v2021.i4.587-617

# Yoroi: Updatable Whitebox Cryptography

Yuji Koike[1] and Takanori Isobe[1,2,3]

[1] University of Hyogo, Hyogo, Japan
yuji03k.u3016@gmail.com
[2] PRESTO, Japan Science and Technology Agency, Tokyo, Japan
[3] National Institute of Information and Communications Technology, Japan
takanori.isobe@ai.u-hyogo.ac.jp

**Abstract.**
Whitebox cryptography aims to provide security in the whitebox setting where the adversary has unlimited access to the implementation and its environment. In order to ensure security in the whitebox setting, it should prevent key extraction attacks and code-lifting attacks, in which the adversary steals the original cryptographic implementation instead of the key, and utilizes it as a big key. Although recent published ciphers such as SPACE, SPNbox, and Whiteblock successfully achieve security against the key extraction attacks, they only provide mitigation of code-lifting attack by the so-called space hardness and incompressibility properties of the underlying tables as the space-hard/incompressible table might be eventually stolen by continuous leakage. The complete prevention of such attacks may need to periodically update the secret key. However, that entails high costs and might introduce an additional vulnerability into the system due to the necessity for the re-encryption of all data by the updated key. In this paper, we introduce a new property, denominated *longevity*, for whitebox cryptography. This property enhances security against code-lifting attacks with continuous leakage by updating incompressible tables instead of the secret key. We propose a family of new whitebox-secure block ciphers Yoroi that has the longevity property in addition to the space hardness. By updating its implementation periodically, Yoroi provides *constant* security against code-lifting attacks without key updating. Moreover, the performance of Yoroi is competitive with existing ciphers implementations in the blackbox and whitebox context.

**Keywords:** Whitebox cryptography, block cipher space hardness, imcompressibility

## 1 Introduction

### 1.1 Existing Whitebox Ciphers and Their Applications

Whitebox cryptography aims to protect cryptographic implementations in software under circumstances where adversaries have unlimited access to the environments for their implementation. This situation is called the whitebox setting, where adversaries are assumed to have control over the execution environment and are allowed to observe and modify internal values of the cryptographic algorithm. This setting is quite different from the standard setting called the blackbox setting where the adversaries can only observe the input and output of the cryptographic algorithm. With the whitebox adversarial capabilities, she mainly tries to mount key extraction attacks, decomposition attacks, and code-lifting attack. In the key extraction attacks, she tries to extract the secret key from the whitebox implementation. In the decomposition attacks, she tries to find a smaller implementation which maintains the exact same functionality than the original one, and in code-lifting attacks, she uses the original cryptographic implementation as a large effective


Received: 2021-04-15 Accepted: 2021-06-15 Published: 2021-08-11

secret key for encryption and decryption on a different device. The security goal for whitebox cryptography is mainly to ensure security against these attacks.

Whitebox ciphers, or precisely speaking, the implementations of DES and AES in the whitebox context, were first introduced by Chow et al. [CEJvO02a, CEJvO02b] in 2002. Their method was to represent algorithms of DES and AES in the form of continual lookups in the tables which are created by the secret key and components of cryptographic operation (i.e. DES or AES). After these pioneering works, several derived whitebox implementations were proposed. However all of them, up to date, were broken by key extraction and decomposition attacks [BHMT16, BBIJ17, DFLM18, BU18, RW19, BRVW19, GPRW20, GRW20].

On the other hand, recent published ciphers take another approach [BI15, BIT16, CCD+17, FKKM16, CCD+17, KLLM20]: they are table-based block ciphers dedicated to the whitebox implementation whose table is constructed from a well-understood standard block cipher (AES for example) by constraining the plaintext and truncating the ciphertext. By this approach, security against key extraction and decomposition attacks in the whitebox setting is reduced to the well studied problem of key recovery for block ciphers in the standard black-box setting. To mitigate code-lifting, a security notion called space hardness was introduced [BI15]. It quantifies security against code-lifting attacks by the amount of code that needs to be extracted from an implementation by a whitebox adversary to maintain its functionality with a certain probability. The ciphers having these properties are called *space-hard ciphers*. This notion is similar to incompressibility [DLPR13] and weak white-box security [BBK14].

There exist several applications of whitebox cryptography. In addition to typical applications such as Digital Rights Management (DRM) and Host Card Emulation (HCE), the leakage-resilient system is a promising application as discussed in [BIT16] and [BKR16]. One of the major problems in the current computer system is memory corruption such as Heartbleed vulnerability [DLK+14] or malware which undetectably stays in a system and extracts data such as a secret key from it. Considering a large table of space-hard ciphers as a big key [BKR16, BD17], it ensures that even if the adversary successfully steals a part of the big key (e.g. one-fourth of the big key) along with the encrypted data, it is computationally difficult to correctly decrypt the encrypted data with high probability. Thus, it can restrain the damage of data leakage in the system. However, when considering the continuous leakage, it causes several problems such as re-encryption problem.

## 1.2 Our Contribution

In this paper, we introduce the new property *longevity* to solve the re-encryption problem for exiting space-hard ciphers. This property ensures that the update of the implementation can increase the space hardness without key rotation, i.e. enhance security against code-lifting attacks while keeping the functionality of the cipher.

**Longevity:** Our idea to achieve longevity is to add an encryption operation of a small-scale block cipher $E_{K_i}$ to an output of space-hard table $T$, and in order to keep the same functionality, the corresponding inverse function $D_{K_i}$ is added to the input for the table in the next round (See Fig. 1). We treat them including new input/output tables as a new table. When updating the implementations, we update only the secret keys $K_i$ for $E_{K_i}$ and $D_{K_i}$ of a small block cipher and do not change the secret key of the original space-hard tables $T$. This approach enables us to add new space hardness by updated tables of $E_{K_i}$ and $D_{K_i}$ while keeping the functionality of the encryption function.

**Instantiation:** We propose a new family of whitebox-secure block ciphers Yoroi that has the

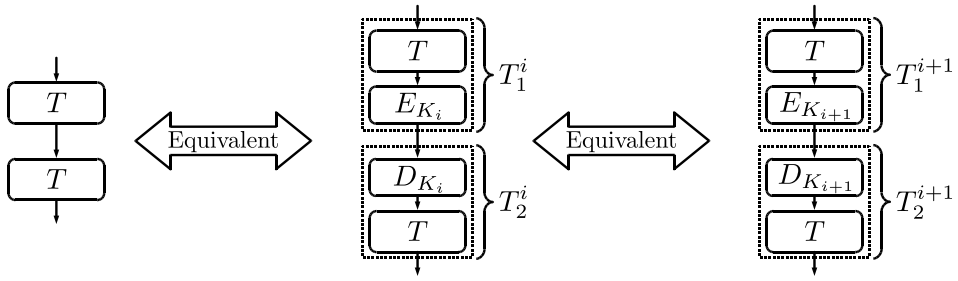


**Figure 1:** Updating Tables for Longevity

longevity property on the top of the space hardness. Yoroi employs the SPN structure as it can provide sufficient security in a smaller number of rounds. In whitebox implementations, the encryption and decryption algorithm of small-scale variants of block ciphers of AES [CMR05] or PRESENT [Lea10] are applied before and after the table lookup as new tables $T$ and $T^{-1}$ for the longevity property. By updating its implementation within the proper period, Yoroi provides *constant security* against code-lifting attack for the first time. Moreover, even with the new property, the performance of Yoroi is very competitive with existing ciphers implementations in the black-box and whitebox context.

### 1.3 Organization

In Section 2, we explore actual applications for space-hard ciphers and discuss a new property which is vital in some contexts. Then, we discuss the basic approach and design choice for our construction Yoroi to ensure this property in Section 3. In the following section, we give the specification of Yoroi and two instantiations of our construction. In Section 5, we evaluate security of Yoroi in the whitebox setting. We give the security bound on Yoroi and show the tightness of this security bound by taking actual attacks as examples. In Section 6, we present the security evaluation in the black-box setting. Based on the security evaluation in Section 5 and Section 6, we provide the implementation evaluation of all the instantiations for Yoroi in section 7. In Section 8, we give a concluding remark on our work.

## 2 Target Applications

Since the introduction of whitebox cryptography, a series of researches have revealed that it has the potential for more real-world applications. In this section, we explore some applications in which whitebox cryptography ciphers, especially space-hard ciphers, are useful. In particular, we discuss why the properties of space-hard ciphers called incompressibility/space-hardness are useful and consider their limitations and drawbacks in the real-world setting. Based on that, we discuss the required property in some contexts for space-hard ciphers.

### 2.1 DRM

Whitebox cryptography originally aimed to provide security of cryptographic key against the adversary in the whitebox context. The original application of whitebox cryptography in their mind was Digital Rights Management (DRM) software so that it can prevent illegal distribution of encrypted digital contents such as music or ebooks along with the

cryptographic key. In order to prevent illegal distribution, whitebox ciphers are required to ensure security against key extraction attacks and code-lifting attacks.

Recent published ciphers [BI15, BIT16, CCD+17, FKKM16, CCD+17, KLLM20] successfully prevent key extraction from the implementation by utilizing the table based on the inputs and outputs of symmetric primitive. The property of incompressibility/space-hardness ensures that if the program is compressed or if fragments of the program are removed, the program loses its functionality, which means that it is difficult to successfully perform a cryptographic operation on arbitrary data with only a part of the program. Thus, as discussed in [BABM20], incompressibility/space-hardness could be useful in the situation where the incompressible large program is delivered via memory hard disc while re-distribution of it through the Internet might be difficult. For instance, if the program size is incompressible and very large e.g. 30 GB, it might be difficult for an adversary to re-distribute it online [DLPR13, BI15, FKKM16, BABM20]. Although it can compromise on encryption performance, we believe incompressible large table increase difficulty of redistribution and thus security level.

However, in terms of security against the code-lifting attacks, they just mitigate the impact of such attacks and it is difficult to ensure security against them. More precisely, since the algorithms of these ciphers are based on the table lookups in the whitebox implementation, by repeating the encryption/decryption of data in the environments where the adversary has full access, she can eventually obtain all the information about the table. Thus, by gradually sending the obtained information to the internet, the adversary can eventually mount code-lifting attacks.

## 2.2 Software Replacements of HSM/TPM for HCE and IoT

HSM (Hardware Secure Module) and TPM (Trusted Platform Module) allow secure cryptographic operations by creating a secure and isolated memory area for the operation and storage of the secret keys for it. This memory area is inaccessible by anyone, and in order to conduct a cryptographic operation, the user has to send the data to operate on the HSM or TPM through the API. Hence, even if the adversary which is typically malware procures unauthorized access to the device, she can not exfiltrate the secret key from HSM or TPM.

However, some devices cannot support such hardware-secure modules due to the constraint of hardware resources, limitation of user environments, and application requirements by IoT devices and Host Card Emulation. In this case, a space-hard cipher [BI15, BIT16, CCD+17, FKKM16, KLLM20], which is a class of whitebox-secure ciphers, can be a software replacement of HSM and TPM.

For example, as discussed by Alpirez Bock et al. [BBF+20], in the scenario of mobile payment applications or HCE where hardware-secure modules are not considered, an adversary who exists in the user's phone (e.g. in the form of malware) might attempt to extract the decryption key and use it for recovering transaction credentials. In addition, the adversary might attempt to simply copy the entire application and run it on a phone of their choice, communicating with a payment terminal of their choice. Thus, mobile payment applications also need protection against code-lifting attacks.

In this context, the security properties of incompressibility/space-hardness are crucial, because they make it hard or even impossible to construct the same functionality in other devices. Even in the situation where the malware stealthily stays and insidiously steals data in the system, typically the data would be gradually leaked and the amount of leaked data would be limited [ADW10]. This assumption is reasonable, as previous works practically demonstrate only limited leakage of credentials. For instance, the adversary tries to steal credentials by mounting memory attacks [HSH+08] to obtain the key stored in RAM, microwave attacks to steal from a smart-card, or power attacks. All of them can expose only partial credentials [ADW10]. These security properties intuitively ensure the

minimum data size needed to realize cryptographic functionality with a certain probability. As a result, due to these properties and the aforementioned assumption [ADW10], the adversary can construct functionality only with a limited probability. Without these properties, the minimum data size for the functionality could be decreased, making it easier or even possible to construct the exact same functionality with a higher probability or probability of 1. Thus, a large amount of leakage (e.g. 20MB for payment terminal with limited GSM bandwidth) is required to copy the functionality, and it increases detection of an attack due to unusually high data transmission. In addition, if the total amount of daily transactions to the outside is estimated or measured in target applications, table leakage can be bounded by it because table leakage never exceeds it. A problem with the incompressibility/space-hardness approach is that regarding security against code-lifting attacks, they only provide mitigation for them. More precisely, since malware possibly staying in a device is the adversary we seek to protect against, by continuously and gradually stealing and sending out the table data to the external network, she can eventually mount a code-lifting attack. To prevent this attack, it requires the update of tables which are the secret keys in the whitebox context, but it could need a lot of computational cost to encrypt data with the new key again.

Besides, there exists a big difference in the key update between the dedicated hardware and the system employing a space-hard cipher. When updating the secret key, it is necessary to re-encrypt the data which is stored in the server, i.e. decrypt the ciphertext with the old key and encrypt it with the new key again. While dedicated hardware can update the secret key and re-encrypt data just within the secure memory area, a system with a whitebox cipher does not have such an area. Thus, the problem for this system is that during the re-encryption of data (i.e. specifically, right after the decryption of the ciphertext with the old key and before the encryption of the decrypted data with a new key), it is prone to the exfiltration of data loaded on the memory by the snapshot adversary [NKW15]. Recently, a new security notion of application (hardware) binding is proposed for DRM and mobile payments [BABM20].

## 2.3 Leakage Resilient System

A promising application is a leakage resilient system employing whitebox cryptography. As discussed in [BIT16] and [BKR16], one of the major problems in the current computer system is memory corruption such as Heartbleed vulnerability [DLK+14] or malware which undetectably stays in a system and extracts data such as a secret key from it. Bellare et al. [BKR16, BD17] show that instead of strengthening the system security in order to protect from malware which sends out data, making the cryptographic key so big that they can not easily exfiltrate the key is another approach. This property is also called *big key encryption*. In this context, a space-hard cipher is also promising, as it employs a large table as the incompressible cryptographic key in the whitebox setting [BIT16].

Specifically, the space-hard ciphers have a security property called space hardness which is also known as incompressibility [DLPR13] and weak white-box security [BBK14]. In this paper, we utilize the definition of space hardness [BIT16]. The definition of space hardness is given in the following.

**Definition 1** ($(M, Z)$-space hardness [BI15])**.** *Block cipher $E_K$ is an $(M, Z)$-space hard cipher if it is computationally difficult to encrypt (decrypt) randomly chosen plaintext (ciphertext) with the probability of more than $2^{-Z}$ in the situation where the adversary is given code (table) size of less than $M$.*

Space hardness aims at quantitatively measuring security against code-lifting attacks (i.e. big key stealing) by showing the relationship between the amount of stolen incompressible table data and the corresponding security level. For example, $(T/4, 128)$-space hard cipher $E_K$ ensures that even if the adversary obtains one fourth of $T$, she can not

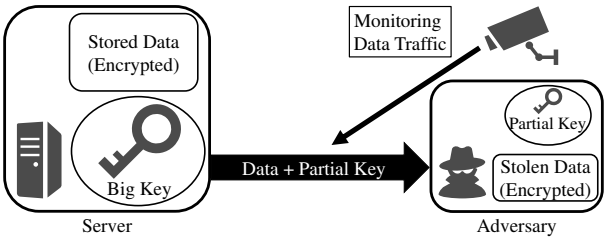


**Figure 2:** Leakage Resillient System with Whitebox Cipher and Network Monitoring

correctly decrypt the randomly drawn ciphertext with the probability of more than $2^{-128}$, where $T$ is the code size (table size) of block cipher $E_K$, and recent published ciphers [BI15, BIT16, CCD+17, FKKM16, KLLM20] provide $(T/4, 128)$-space hardness or a similar level of it.

Suppose that a system carries out a cryptographic operation on accessible memory by authorized users to encrypt data by a space-hard cipher and store it in the server. The encryption by a space-hard cipher can provide a sufficient level of security against snapshot adversary [NKW15], who has only access to information about the content loaded on the memory of the server at a certain moment, by monitoring the size of stolen data to outside of the network, and updating the secret key when the amount of stolen data exceeds a certain point, e.g. 1/4 of data size for the total table, as shown in Fig. 2.

The drawback of this leakage resilient system is a periodic key updating after the leakage of $T/4$ data to keep security. It causes the re-encryption of all stored data. Obviously, it is an expensive computational cost in real-world applications. In addition, it might have a security risk such that during the re-encryption of data, it is possible for the snapshot adversary [NKW15] to read the plaintext loaded in the memory.

## 2.4 Required Property for Whitebox Cryptography

As the applications indicate, updating the table (i.e. secret key for the underlying cipher) might prevent a code-lifting attack relying on continuous leakage. However, that entails the re-encryption of data. This could be an expensive cost or even vulnerability in some contexts.

To address this issue, we introduce the new property called *longevity* as a resistance of cryptographic functions against continuous data leakage which leads to code-lifting attacks. In this context, the adversarial model we assume is that the adversary can specify a leakage function of bounded length. The function is then evaluated in the environment where the incompressible program is located, and the adversary obtains the results. The adversary then tries to perform decryption operations using the bounded leakage which is the results obtained from the leakage function. The adversary does not have encryption and decryption oracles. Note that throughout this paper, we discuss security against code-lifting attacks based on this model.
Within the model above, we now define *$Z$-longevity*, where $Z$ denotes an upper bound on the number of leaked bits per phase.

**Definition 2** (($Z$)-longevity)**.** *Cryptographic function has $(Z)$-longevity if it is computationally difficult to encrypt (decrypt) randomly-chosen plaintext (ciphertext) with probability of more than $2^{-Z}$ in the situation where the functionality remains constant, and code (table) is continuously leaked to the adversary.*

A space-hard cipher with this property persistently guarantees *constant* security of a cryptographic function against code-lifting attacks while keeping the same functionality,

which means it can consequently avoid the re-encryption of stored data. Thus, in the case of a continuous leakage, even after the leakage of $(T/4)$ data, by properly updating incompressible tables, the probability that a randomly-drawn plaintext can be correctly computed can be constant, e.g. $2^{-128}$ in the case of (128)-longevity.

**Remark.** When evaluating $(Z)$-longevity of a new construction, some parameters, such as how much data is leaked and how often the leakage itself happens, should be considered so that the parameters can represent the continuous leakage which is mentioned in our definition. However, we do not include these parameters in our definition, because we believe these parameters heavily depend on an application where whitebox cipher is deployed, not on the cipher itself. So, in the evaluation of $(Z)$-longevity it is important to consider these parameters.

# 3 Our Approach for Longevity

In this section, we first give a basic idea about how to ensure the property of longevity, i.e. update space hardness without updating the secret key of the underlying cipher. After that, we show our fundamental construction that has longevity on the top of space-hard property.

## 3.1 How to Update Tables while Keeping the Functionality

Our idea to add longevity into space-hard ciphers which are based on the incompressible table $T$ is to apply encryption operation of small-scale block cipher $E_{K_i}$ to the output of the incompressible table $T$ when updating the tables as shown in Fig. 1. We treat them as new tables. Intuitively, this enables adding a new source of the space hardness into the implementation.

Here, in order to keep the original functionality, we have to apply the corresponding inverse table (i.e. inverse function) $D_{K_i}$ to the input for the table in the next round (See Fig. 1). We repeatedly apply incompressible table $E_{K_i}$ and its inverse function $D_{K_i}$ to outputs and inputs of all tables, respectively.

When updating the implementations, we update the secret key $K_i$ for these small block ciphers $E_{K_i}$ and $D_{K_i}$ and do not change the secret key of the underlying ciphers for the original table. Thus, it can keep the same functionality even after updating the implementation.

## 3.2 Our Design: SPN with Partial MDS Layer

As an underlying structure, we choose a substitution-permutation network (SPN) structure as is the case with SPNbox [BIT16], because it can offer sufficient security in a smaller number of rounds than a Feistel structure. Following the design of SPNbox [BIT16], each Sbox is realized by a small block cipher which is represented as tables in the whitebox implementation. Also, we use an MDS matrix in linear layers to achieve a full diffusion in the small number of rounds.

However, if this MDS layer takes the whole outputs from each table lookup, it is impossible to ensure the same functionality after updating $E_{K_i}$, because the corresponding decryption function $D_{K_i}$ in the next rounds cannot be constructed due to the matrix operation between these two tables. Although it could be possible if the encryption function has a homomorphic property, its performance is obviously unacceptable in our target applications.

To address this issue, we employ an SPN with the partial MDS layer. In our structure, an output of tables $T$ is divided into two parts. The MDS layer takes one part, and the

function $E_{K_i}$ takes the other part as their inputs. The output from MDS layer is a direct and partial input for the S layer in the next rounds as shown in Fig. 3

Interestingly, this construction provides the same functionality after updating its tables and achieves the 2 round full diffusion property as well as the SPN layer with the MDS matrix assuming $T$ has the full diffusion property.

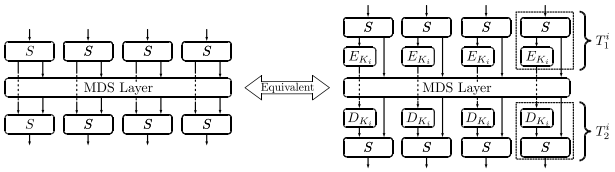


**Figure 3:** Underlying Construction: SPN with partial MDS layer

# 4 Specification of Yoroi

Yoroi, which is an $n$-bit block cipher with a $k$-bit secret key, employs an SPN with the partial MDS layer. The state in $r$-th round $X^r$ is expressed as $\ell$ $(= n/n_{in})$ elements of $n_{in}$ bits, i.e., $X^r = \{x_0^r, x_1^r, ..., x_{\ell-1}^r\}, x_i^r \in \{0,1\}^{n_{in}}$ for $1 \leq r \leq R$, where $R$ is the number of total rounds of Yoroi. Each element $x_i^r$ $(1 \leq r \leq R,\ 0 \leq i \leq \ell - 1)$ is represented as $(msb_m(x_i^r) \parallel lsb_t(x_i^r))$ where $m + t = n_{in}$, $msb_m$ and $lsb_t$ denote the most significant $m$ bits and the least significant $t$ bits of the element respectively, and $\parallel$ denotes concatenation.

In this section, we will explain the specification of Yoroi in blackbox and whitebox implementations, respectively. Note that these functionality are the same, however, for the whitebox implementation, the small scale block ciphers $E_K$ and its inverse functions $D_K$ are added to achieve the property of longevity as discussed in Section 3.

## 4.1 Blackbox Implementation

The encryption of a plaintext $X^0$ to a ciphertext $X^R$ is accomplished by applying $R$ rounds of the following round transformation to the plaintext:

$$X^R = \mathcal{A} \circ \gamma^R \circ \left(\bigcirc_{i=1}^{R-1} \left(\theta \circ \sigma^i \circ \gamma^i\right)\right)(X^0).$$

### 4.1.1 S-layer $\gamma^i$.

The S-layer $\gamma^i$ consists of $\ell$ key-dependent $n_{in}$-bit bijective functions, and is applied to the whole state: in the first round and last round, $S_1(x)$ and $S_3(x)$ are applied as $n_{in}$-bit bijective functions, and for the rest of rounds, $S_2(x)$ is applied, respectively:

$$\begin{aligned} \gamma^i : \mathrm{GF}(2^{n_{in}})^\ell &\rightarrow \mathrm{GF}(2^{n_{in}})^\ell \\ (x_0, \ldots, x_{\ell-1}) &\rightarrow (S_j(x_0), \ldots, S_j(x_{\ell-1})) \end{aligned}$$

where $j = \{1, 3\}$ when $i = \{1, R\}$ and for the rest of cases (i.e. $2 \leq i \leq R-1$), $j = 2$, respectively.

### 4.1.2 Linear layer $\theta$.

The linear layer $\theta$ consists of an $\ell \times \ell$ MDS matrix over $\mathrm{GF}(2^t)$, and is applied to the part of the state. Specifically, the $\ell \times \ell$ MDS matrix $M_t$ takes the least significant $t$ bits of each $n_{in}$-bit element as inputs. The linear layer $\theta$ is defined as follows.

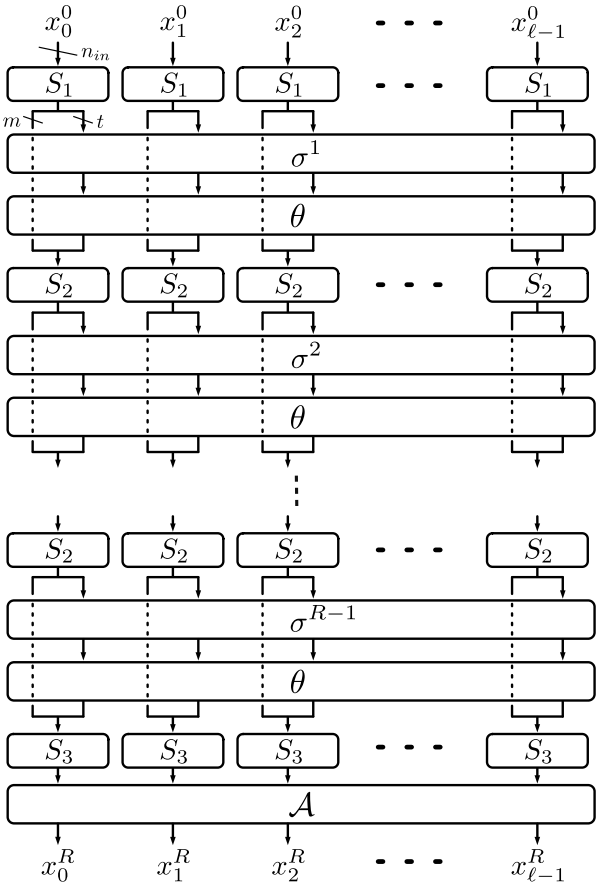


**Figure 4:** Algorithm of Yoroi (Blackbox Implementation)

$$\begin{aligned} \theta : \mathrm{GF}(2^t)^\ell &\rightarrow \mathrm{GF}(2^t)^\ell \\ (lsb_t(x_0^r), ..., lsb_t(x_{\ell-1}^r)) &\rightarrow (lsb_t(x_0^r), ..., lsb_t(x_{\ell-1}^r)) \cdot M_t \end{aligned}$$

#### 4.1.3 Affine layer $\sigma^i$.

$\sigma^i$ is an affine layer which adds constant $C_i$ to the state in $i$-th round:

$$\begin{aligned} \sigma : \mathrm{GF}(2^{n_{in}})^\ell &\rightarrow \mathrm{GF}(2^{n_{in}})^\ell \\ (x_0, \ldots, x_{\ell-1}) &\rightarrow (x_0 \oplus C_i, \ldots, x_{\ell-1} \oplus C_i) \end{aligned}$$

where $C_i = i$ for $1 \leq i \leq R-1$.

Note that the constant $C_i$ has to be smaller than $2^t$ in order to maintain the same functionality of blackbox implementation and whitebox implementation.

Fig. 4 illustrates the algorithm of Yoroi in the blackbox context, and the detailed algorithm is given in the Appendix A.

#### 4.1.4 AES layer $\mathcal{A}$.

Following a diffusion method as in previous works [FKKM16, CCD+17], Yoroi employs 10 *round AES* $\mathcal{A}$ with a fixed key $K_{\mathcal{A}}$ which is generated independently of the keys used in S-layers, so that it can eliminate possible cryptographic characteristics. Note that in Yoroi, AES is employed only in the last round. This approach can reduce security in the blackbox context to the well-studied problem of key recovery for AES.

## 4.2 Whitebox Implementation

The main difference of the algorithm of Yoroi in the whitebox implementation from the one in the blackbox implementation is the application of $m$-bit $E_K$ and $D_K$ to the bijective functions $S_1(x)$, $S_2(x)$, and $S_3(x)$. $E_K$ and $D_K$ are $m$-bit encryption and decryption algorithms of small-variant of block ciphers and takes the most significant $m$ bits of the element $x_i^r$ as an input, respectively. Specifically, we use small variants of AES [CMR05] or PRESENT [Lea10] as an instantiation of $E_K$ and $D_K$.

By alternating $E_K$ and $D_K$ between the bijective functions, Yoroi in the whitebox implementation keeps the exact same functionality as the one in the blackbox context. In Yoroi, there are the following three types of tables $T_1$ and $T_2$,

$$\begin{aligned} T_1 &= E_K(msb_m(S_1(x)))||lsb_t(S_1(x)), \\ T_2 &= E_K(msb_m(S_2(D_K(msb_m(x))||lsb_t(x))))|| \\ &\quad lsb_t(S_2(D_K(msb_m(x))||lsb_t(x))), \\ T_3 &= S_3(D_K(msb_m(x))||lsb_t(x)). \end{aligned}$$

Yoroi employs $T_1$, $T_2$, and $T_3$ as shown in Fig. 5. These are implemented by tables.

The full algorithm of Yoroi in the whitebox context is given in Appendix A.

## 4.3 Updating Whitebox Implementation

When updating the whitebox implementation, $E_K$ and $D_K$ in $T_1$, $T_2$, and $T_3$ are replaced with new ones, i.e. the key of underlying ciphers of $E_K$ and $D_K$ is updated. Specifically, we compute new tables of $T_1$, $T_2$, and $T_3$ with a new key of $E_K$ and $D_K$, while original functions $S_1$, $S_2$, and $S_3$ are unchanged.

For simplicity, let expressions of $E_K(msb_m(S_1(x)))||lsb_t(S_1(x))$, $E_K(msb_m(S_2(D_K(msb_m(x))||lsb_t(x))))||lsb_t(S_2(D_K(msb_m(x))||lsb_t(x)))$, and $S_3(D_K(msb_m(x))||lsb_t(x))$ be $E_K(S_1(x))$, $E_K(S_2(D_K(x)))$ and $S_3(D_K(x))$, respectively.

## 4.4 Instantiations

For concrete instantiations, we propose two variants with the following specification.

- Yoroi-16 : $n = 128$, $\ell = 8$, $n_{in} = 16$, $m = 12$, $t = 4$, $R = 8$, $S_i(x) : \{0,1\}^{16} \to \{0,1\}^{16}$,
- Yoroi-32 : $n = 128$, $\ell = 4$, $n_{in} = 32$, $m = 28$, $t = 4$, $R = 16$, $S_i(x) : \{0,1\}^{32} \to \{0,1\}^{32}$.

For Yoroi-16 and Yoroi-32, $S_1(x)$, $S_2(x)$, and $S_3(x)$ are instantiated by 16 and 32-bit key-dependent Sboxes for SPNbox [BIT16] with different keys, respectively, which are obtained from the $k$-bit master key by any generic KDF function.

For the concrete linear diffusion layer $\theta$ for Yoroi-16 and Yoroi-32, we employ the $8 \times 8$ and $4 \times 4$ MDS matrixes used in Whirlwind [BNN+10] and Piccolo [SIH+11], respectively. The irreducible polynomial for these matrixes is $x^4 + x + 1$.

The size of table $T_i$ $(1 \le i \le 3)$ is estimated as $(2^{n_{in}} \times n_{in})$ bits. Thus, the actual size of each table for Yoroi-16 and Yoroi-32 is $(2^{16} \times 16)$ and $(2^{32} \times 32)$ bits, respectively,

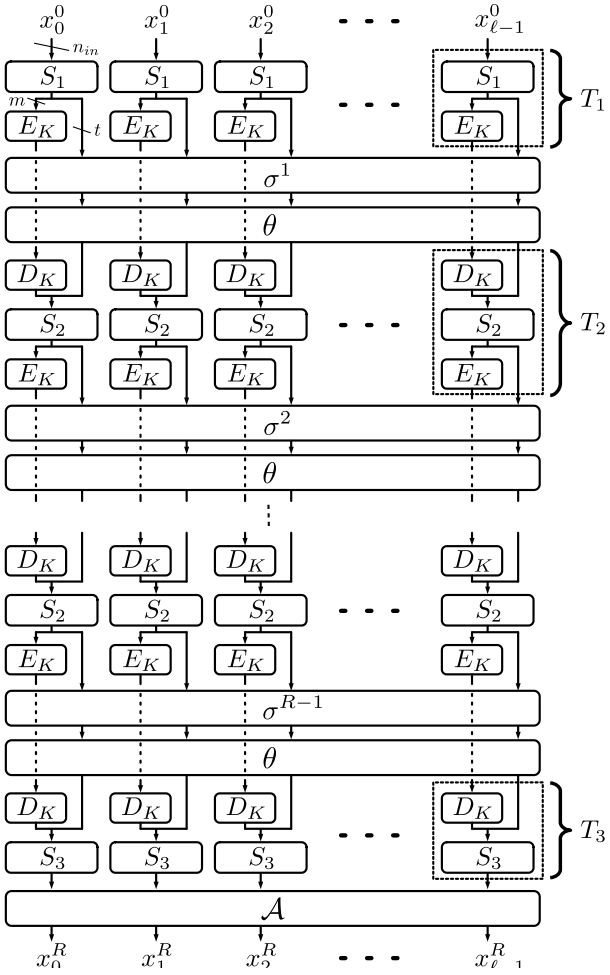


**Figure 5:** Algorithm of Yoroi (Whitebox Implementation)

and total table size for Yoroi-16 and Yoroi-32 is $2^{20} \times 3$ ($= (2^{16} \times 16) \times 3$) and $2^{37} \times 3$ ($= (2^{32} \times 32) \times 3$) bits.

# 5 Whitebox Security

Here, we analyze the security of Yoroi in the whitebox setting. Recall that for the table-base implementation, tables of $T_1$, $T_2$, and $T_3$ are composed of $n_{in}$-bit bijective functions of $S_1(x)$, $S_2(x)$, and $S_3(x)$, and $m$-bit bijective functions of $E_K$ and $D_K$ where $m < n_{in}$.

In the whitebox setting, it should provide sufficient security against the key extraction attack and code-lifting attack. In Section 5.1 and Section 5.2, we give the security evaluation against these attacks in the standard setting, i.e. there is no update of implementation/tables. Then in Section 5.3, we explore the security after the table update to show how to ensure the longevity for Yoroi.

## 5.1 Security against Key Extraction

In the whitebox setting, the adversary can observe not only the inputs and outputs of the cryptographic function, but also its internal values. Hence, she can fully access all entries

of the table of $T_1$, $T_2$, and $T_3$, i.e., all pairs of inputs and the corresponding outputs in the table.

In order to extract the secret key in the whitebox setting, she has to at least recover the secret key from the underlying cipher for $S_1(x)$, $S_2(x)$, and $S_3(x)$ in the blackbox setting. Therefore, the security of Yoroi against the key extraction attack in the whitebox setting is reduced to the key recovery problem for the underlying cipher in the blackbox setting as with existing space-hard schemes [BI15, BIT16, CCD+17, FKKM16, KLLM20].

As a corollary, if the underlying cipher used to generate the tables $T_1$, $T_2$, and $T_3$ is secure against the key recovery attacks in the blackbox setting, it is computationally infeasible to extract the secret key from the tables $T_1$, $T_2$, and $T_3$ in the whitebox context.

## 5.2 Security against Code-Lifting Attack

In this section, we evaluate the difficulty of code-lifting attacks on Yoroi. In this context, we follow the adversarial model discussed in Section 2.4. Namely, the adversarial goal is to obtain the functionality of the implementation code and use it in a stand-alone manner. For instance, the whitebox adversary who controls its execution environment and can see the full program tries to achieve this goal by stealing the implementation code (ex. by sending out information the implementation code from the environment and reconstructing the same functionality as the original code with that stolen information in another environment of her choice). In order to evaluate security against code-lifting attacks, we will utilize the space hardness [BI15], which represents the probability that the adversary can reconstruct the same functionality as the original one with the stolen and limited information. Additionally, in our analysis, we assume just as in [BI15, BIT16, CCD+17] that the adversary's memory contains only secret table entries. Note that our evaluation assumes as a limitation that the adversary has bounded computational resources, and does not consider chosen-ciphertext attacks. Regarding the space hardness of Yoroi, we give the following intuitive theorem.

**Theorem 1.** *When $S_1$, $S_2$, $S_3$, $E_K$, and $D_K$ are assumed to be pseudorandom functions, given $i$, $j$, and $k$ table entries for the table $T_1$, $T_2$, and $T_3$, respectively, the probability $p$ that the adversary can correctly encrypt (decrypt) a randomly chosen plaintext (ciphertext) is upper bounded by*

$$p = \left(\frac{i}{2^{n_{in}}}\right)^{\ell} \times \left(\frac{j}{2^{n_{in}}}\right)^{\ell\times(R-2)} \times \left(\frac{k}{2^{n_{in}}}\right)^{\ell},$$

*where $\ell$ and $R$ denote the number of table lookups in one round and the total round of the cryptographic algorithm, respectively, and $0 \leq i, j, k \leq 2^{n_{in}}$.*

*Proof.* Given $i$, $j$, and $k$ table entries for the table $T_1$, $T_2$, and $T_3$ respectively, the probability that each entry for the table $T_1$, $T_2$, and $T_3$ is included in the known entries can be estimated as $i/2^{n_{in}}$, $j/2^{n_{in}}$, and $k/2^{n_{in}}$, respectively. Hence, the adversary can derive the correct intermediate value in one round transformation by looking up the known entries with the probability of $(i/2^{n_{in}})^{\ell}$, $(j/2^{n_{in}})^{\ell}$, or $(k/2^{n_{in}})^{\ell}$, depending on which table is used in the round transformation. According to the specification in Section 4.2, $T_1$ and $T_3$ are used for the first and last round in Yoroi, respectively, and for the rest of the rounds $T_2$ is used. Therefore, the probability $p$ that the adversary can correctly encrypt (decrypt) a randomly chosen plaintext (ciphertext) is bounded by

$$p = \left(\frac{i}{2^{n_{in}}}\right)^{\ell} \times \left(\frac{j}{2^{n_{in}}}\right)^{\ell\times(R-2)} \times \left(\frac{k}{2^{n_{in}}}\right)^{\ell},$$

□

Recent published ciphers [BI15, BIT16, CCD+17, FKKM16, KLLM20] provide $(T/4, 128)$-space hardness, meaning that even if the adversary successfully steals one fourth

of the entire table, she can not correctly encrypt (decrypt) a randomly drawn plaintext (ciphertext) with the probability of more than $2^{-128}$. According to Theorem 1, Yoroi-16 and Yoroi-32 can also provide $(T/4, 128)$-space hardness when $R = 8, 16$, respectively. Note that our proof is based on the known/chosen space attack model defined in [BIT16], and if we consider adaptively chosen space attack in [BIT16], the security bound for Yoroi-$n$ would be $(T/4, 128 - n)$-space hardness as in [CCD+17, FKKM16], where $n \in \{16, 32\}$. Moreover, as tables of Yoroi are based on bijective functions, it is actually possible to construct the same functionality with a slightly smaller size of the whole table by exploiting the property of bijection functions as discussed in [CCD+17], e.g. 16-bit permutation can be constructed by $14.56 \times 2^16$ bits instead of $16 \times 2^16$ bits. However, the impact arising from that on the security bound is marginal, so we omit it in the proof above for the sake of simplicity.

## 5.3 Longevity: Space Hardness After the Table Update

This section shows that Yoroi can provide constant security against code-lifting attacks by updating the tables within a proper period. First of all, we explore the incompressibility of Yoroi after the table update and then give the security bound of the space hardness by taking continuous table updating into consideration.

### 5.3.1 Incompressibility of Updated Tables.

Let $T^j$ be a family of tables $T_1^j$, $T_2^j$, and $T_3^j$ after $j$ times updating. As the incompressibility of updated tables $T^j$, we aim to obtain the lower bounds on the data size to be stolen from $T^j$ to copy the full functionality of the encryption/decryption functions.

In our evaluation, we consider the worst case where the adversary has the knowledge about all the previous tables $T^i$ where $0 < i < j - 1$. We show that even under such a strong case, the updated table has a significant incompressible property. It means that table updating can inject a new source of the space hardness into the implementation. The lower bound is given in the following Lemma under the assumption where $E_K$, $D_K$, $S_1$, $S_2$, and $S_3$ are pseudorandom permutations.

**Lemma 1.** *Given all the previous tables of $T^i$ ($\in T_1^i$, $T_2^i$, $T_3^i$) for $0 < i$, if $E_K$ and $D_K$ are $m$-bit pseudo random permutations, it is computationally infeasible to find the equivalent functionality of tables $T^j$ ($\in T_1^j$, $T_2^j$, $T_3^j$) whose total size is less than $(2^m \times m) \times 2$ bits.*

To prove lemma 1, we show the proof by contra-position. The contraposition of the lemma 1 is given in the following.

**Contraposition 1.** *Given all the previous tables of $T^i$ ($\in T_1^i$, $T_2^i$, $T_3^i$) for $0 < i$, if it is computationally feasible to find the equivalent functionality of table $T^j$ ($\in T_1^j$, $T_2^j$, $T_3^j$) whose total size is less than $(2^m \times m) \times 2$ bits, $E_K$ and $D_K$ are not $m$-bit pseudo random permutations.*

*Proof.* In order to compress the tables $T^j$, the adversary at least has to know about how elements in the tables $T^j$ are updated from those in the tables $T^{j-1}$. In other words, considering the design of tables $T^i$, she has to collect information about $E_{K_j}$ and $D_{K_j}$ whose size is $(2^m \times m) \times 2$ bits in total. If it is computationally feasible to compress the tables $T^j$ into smaller equivalent functionality whose size is less than $(2^m \times m) \times 2$ bits, it means that the adversary can guess how some of the values for elements in the tables $T^j$ are changed from those in the tables $T^{j-1}$ with higher probability than randomly guess that. This simply indicates that particular outputs of $E_{K_j}$ and $D_{K_j}$ are more predictable than the others and that demonstrates $E_{K_j}$ and $D_{K_j}$ are not pseudo random permutations. □ Note that this proof might be intuitive, as we are not strongly convinced that there is no

correlation between $E_{K_i}$ and $D_{K_i}$ which are included in $T^i$, and we heuristically treated these functions as independent permutations

In order to show the tightness of lemma 1, we consider and describe compression attacks on each table $T_1^j$, $T_2^j$, and $T_3^j$ under the assumption that the adversary has the knowledge about all the previous tables $T^i$ where $0 < i < j - 1$ in Appendix C D E. This section actually gives a glimpse at how much data the adversary has to obtain after the table update to copy the full functionality.

### 5.3.2 Main Conclusion of Incompressibility of Updated Tables.

These compression attacks on $T_1$, $T_2$, and $T_3$ show that the bound of required data to copy the functionality of each individual table after table updating in Lemma 1 is tight. Besides, they show that the table data that the adversary previously obtains does not affect the lower bound of the updated implementation which lemma 1 provides. It means that updating tables in this manner can add a new source of space hardness into the implementation. Hence, we can evaluate the security bound of the updated implementation against code-lifting attacks by using the security notion of space hardness, without having to consider the influence of the previously stolen data.

### 5.3.3 Space Hardness with Updating Tables

In order to evaluate the space hardness on Yoroi after the table update, we introduce the following theorem.

**Theorem 2.** *Given $g$, $x$ and $h$ entries for table $T_1^j$, $T_2^j$ and $T_3^j$ respectively, the probability that the adversary can compute a randomly chosen plaintext into a correct ciphertext is upper bounded by*

$$\left(\frac{g \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^l \times \left(\frac{x \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l\times(R-2)} \times \left(\frac{h \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^l, \tag{1}$$

*assuming that the adversary does not gain enough table entries for any version of the table $T_1^i$, $T_2^i$, and $T_3^i$ to successfully encrypt (decrypt) plaintaxt (ciphertext) with the probability of more than probability of (2), where $0 \leq i \leq j-1$, $g, h < 2^m$ and $x = min(g, h)$.*

*Proof.* Because of differences in the input (output) size between $S_1(x)$ ($S_2(x)$, or $S_3(x)$) and $E_K$ (or $D_K$), if she obtains one entry of the table $T_1^j$ (or $T_3^j$), she can infer at most $(2^{n_{in}-m} - 1)$ entries. That is virtually equivalent to the situation where she knows at most $2^{n_{in}-m}$ entries just by obtaining one entry of the table. So, given the case where the adversary steals a part of the table, i.e., $g \leq 2^m$ entries for $T_1^j$ are leaked to her, the number of total entries that are virtually leaked to her is up to $(g \times 2^{n_{in}-m})$.

When estimating the total entries of a whole table as $2^{n_{in}}$, the probability that each entry for table $T_1^j$ is included in the $(g \times 2^{n_{in}-m})$ leaked entries is $(g \times 2^{n_{in}-m})/2^{n_{in}}$. Thus, the probability that she can derive the correct output by looking up the leaked entries of the table is $(g \times 2^{n_{in}-m})/2^{n_{in}}$. Considering the fact that Yoroi uses table $T_1^j$ $\ell$ $(= n/n_{in})$ times to look up values in one round where $n$ is the block size for Yoroi, the probability that the adversary can obtain the correct intermediate value by looking up the tables $l$ times is given as:

$$\left(\frac{g \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^\ell.$$

Besides, with the same logic, by using $h$ entries for table $T_3^j$, the probability that the adversary can compute the correct intermediate value is:

$$\left(\frac{h \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^\ell.$$

In order to infer some table entries of $T_2^j$, the adversary at least has to know about $E_{K_j}$ and $D_{K_j}$. Information about $E_{K_j}$ tells her what each output of the table $T_2^{j-1}$ is updated to, and that of $D_{K_j}$ tells what each index of the table $T_2^{j-1}$ is updated to, i.e., where each updated output is moved to. As mentioned, leakage of one entry leads to that of $2^{n_{in}-m}$ entries. Specifically, with the information about transition from $E_{K_{j-1}}$ to $E_{K_j}$, one leaked entry enables the adversary to infer $2^{n_{in}-m}$ outputs of table $T_2^j$ and with that the information about transition from $D_{K_{j-1}}$ to $D_{K_j}$ each leaked entry possibly enables her to infer indexes, i.e., positions for $2^{n_{in}-m}$ outputs in the table $T_2^j$. Because of the design of table $T_2$, the adversary has to gain both information about these types of transition.

In the case of $g < h$ the adversary can possibly gain the indexes for $h \times 2^{n_{in}-m}$ outputs. However, she can infer what only $g \times 2^{n_{in}-m}$ outputs of the table $T_2^{j-1}$ are updated to. This means that she can infer only the indexes for $(h-g)\times 2^{n_{in}-m}$ $(= (h\times 2^{n_{in}-m})-(g\times 2^{n_{in}-m}))$ outputs and can not know the actual values for them.

In the case of $g > h$, the adversary can infer $g \times 2^{n_{in}-m}$ output values for the table $T_2^j$, while she can infer the indexes for only $h \times 2^{n_{in}-m}$ outputs. Thus, even though she can infer infer $g \times 2^{n_{in}-m}$ output values, she can not know the indexes for $(g-h) \times 2^{n_{in}-m}$ $(= (g \times 2^{n_{in}-m}) - (h \times 2^{n_{in}-m}))$ outputs for the table $T_2^j$

Given the discussion above, the number of table entries (i.e. updated table outputs and their correct indexes) the adversary can gain is up to $x \times 2^{n_{in}-m}$, where $x = min(g, h)$. Also with the same logic discussed in table $T_1^j$ and $T_3^j$, the adversary can obtain the intermediate value by using leaked information about $T_2^j$ with the probability of up to:

$$\left(\frac{x \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{\ell}.$$

In order to compute a correct output, the adversary has to obtain a correct intermediate value in any round. In $R$ rounds, Yoroi uses the table $T_1^j$ and $T_3^j$ for the first and last round respectively, and for the rest of rounds $T_2^j$ is employed, where $R$ is the total rounds for Yoroi. Therefore, after $R$ rounds, the adversary can compute the correct output from the random-drawn input with the probability of

$$\left(\frac{g \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l} \times \left(\frac{x \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l\times(R-2)} \times \left(\frac{h \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l},$$

□

For Yoroi-16 and Yoroi-32 where $R = 8, 16$ respectively, they can ensure that after the table update, the probability that the adversary successfully encrypts (decrypts) randomly drawn plaintext (ciphertext) by using the stolen table entries is less than $2^{-128}$ until the number of stolen table entries for each table $T_1^j$ and $T_2^j$ reaches $2^{m-2}$, i.e., $2^{10}$ and $2^{26}$ entries each for Yoroi-16 and Yoroi-32, respectively. Thus, these variants provide $(T/64, 128)$-space hardness as a whole.

Note that the incompressibility property shown in Section 5.3.1 and Section 5.3.2 demonstrates that the space hardness of the current implementation can be evaluated independently from previously stolen tables. Thus, the space hardness of Yoroi is bounded by the amount of leaked data in the period during each table updating. In other words, with the Theorem 2 and incompressibility property, if we update tables before the amount of leaked data reaches a certain threshold, we can keep the constant space hardness persistently without changing the master key. We will explain it by exploring the possibility of the same functionality with different versions of tables and example cases for Yoroi-32.

## 5.4 Possibility of Functionality Replication with Combined Tables of Different Versions

In the previous section, we discuss space hardness based on the adversary who aims at code-lifting attacks by using updated tables of a specific version. In other words, we did not consider the adversary who might attempt to mount code-lifting attack with higher probability by combining updated tables of several versions (e.g. $T^{j-1}$ and $T^j$) and using it to realize the same functionality.

In this case, however, the adversary using the tables of different versions can not even keep the same functionality. This is because by using tables of $T^{j-1}$ and $T^j$ for example, data to be processed has to pass through $E_{K_j}$ and $D_{K_{j-1}}$ (or $E_{K_{j-1}}$ and $D_{K_j}$) from the algorithm viewpoint. This discussion strengthens the statement that the space hardness of the current implementation can be evaluated independently from previously stolen tables.

## 5.5 Examples Study

According to Theorem 1, Yoroi-32 has $(T/4, 128)$-space hardness such that after 1/4 of each table $(T_1^0, T_2^0, T_3^0)$ is stolen by the adversary, the probability that she correctly computes the encryption or decryption function is bounded by $2^{-128}$. However, after that, by a continuous leakage, the probability increases and eventually reaches the one, i.e. copy of the full functionality, As discussed before, the update of tables $(T_1^0, T_2^0, T_3^0)$ to $(T_1^1, T_2^1, T_3^1)$ can keep this security level of the space hardness even after $T/4$ of leakage.

**Longevity.** According to Theorem 2, after the table update, Yoroi-32 can ensure $(T/64, 128)$-space hardness independently from the previous leakage. Thus, the adversary cannot decrypt (encrypt) a randomly chosen ciphertext (plaintext) with the probability of more than $2^{-128}$ until the amount of leaked data for $(T_1^1, T_2^1, T_3^1)$ reaches $T/64$. Therefore, by continuously updating the table before reaching this threshold, i.e., before the leaked data for the updated table $(T_1^1, T_2^1, T_3^1)$ amounts to $T/64$, Yoroi-32 achieves (128)-longevity such that the adversary can not successfully encrypt or decrypt with the probability of more than $2^{-128}$ persistently. This gives the new property called longevity to Yoroi-32.

**Example.** Figure 6 illustrates the security level of space hardness for Yoroi-32 with table update and that without table update over time, assuming 16 MB of data is leaked per day for an illustrative purpose. In this case, the amount of leaked data reaches $T/4$ in 512 days and at this point, the probability that the adversary can successfully encrypt (decrypt) a randomly chosen data is bounded by $2^{-128}$. By updating the table at this point, Yoroi-32 can stop the increase in the adversarial advantage (i.e success probability that she can correctly encrypt (decrypt) a randomly drawn data), and as shown in Theorem 2, the updated implementation of Yoroi-32 provides $(T/64, 128)$-space hardness. Considering the leakage of 16MB data on a daily basis, it takes 32 days for the amount of leaked data for the updated table to reach $T/64$. Therefore, after the update of the original table, updating the table every 32 days can constantly ensure that the success probability for correct computation by the adversary is at most $2^{-128}$, namely (128)-longevity.

**Remark.** If the adversary successfully obtains full information for a certain version of the tables, this scenario will not work and she no longer would need to gain more information for the table from that point. Thus, it is important to properly update the tables at the right moment. Besides, there could be more data leakage or less depending on the application. So, we set the leakage size to 16MB as an example. In the case where there is greater leakage amount, it could be a good idea to use tables with a bigger size by using different bijective functions in S-layers.

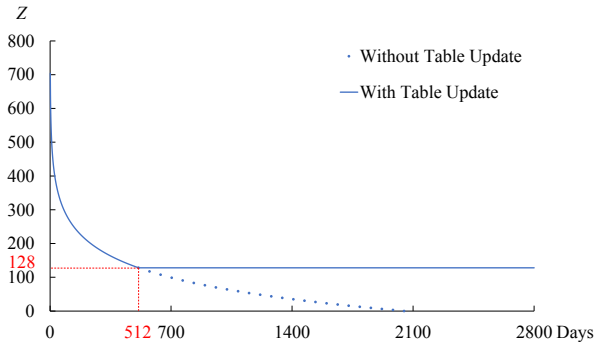


**Figure 6:** Comparison of $(M, Z)$-space hardness on Yoroi-32 with table update and that without table lookup considering the leakage of 16MB data per day

**Limitation.** Our security model [BI15, BIT16, CCD+17] does not assume the whitebox adversary who can have access to the blackbox implementations after obtaining a part of the table from the target device by the code lifting attack, i.e. after code lifting, the adversary only do the game in which she tries to compute valid ciphertexts given randomly-chosen plaintexts using the part of tables. Such a stronger adversary might recover additional table information from not only target device but also blackbox access. It is an interesting problem to evaluate the rigorous security of the leakage setting.

## 5.6 How to Update Tables

The table updates will be triggered at some point before the amount of leakage data which is monitored exceeds this point (i.e. less than $T/4$ or $T/64$). By the table updates, all tables in devices are overwritten by new ones. Note that depending on the applications, the tables can be updated locally or by communicating with a server which generates and distributes tables.

**Leakage Resilient System:** A monitor which observes how much data is leaked so far tells that the amount of leakage is close to the threshold to another server which generates and distributes updated tables, which is the trigger for a table update. After the communication, it distributes the precomputed table to the server, which we call table update. Note that the server generates another new table right after the table update to prepare for the another table update. Besides, it is possible to integrate monitor and server generating and distributing tables into one table to remove unnecessary communication between them. Figure 7 illustrates the table update flow. Note that in the case of the leakage resilient system, it is still possible to locally update the tables.

**HCE:** Assuming that the total amount of daily transactions to the outside is estimated or measured in target applications, the upper bound of table leakage can be estimated because table leakage never exceeds it. Thus, when the upper bound of leakage gets close to the threshold (i.e. less than $T/4$ or $T/64$), tables are replaced with new tables which are sent from the server. The trigger and method for table updates are the same as in the leakage resilient system.

In the case of table updates via communication with the server, we assume just as in [LER+18] that the communication and interaction between server and device are secure. In addition, the adversary might try to block the communication channel to prevent the table updates. However, we believe this is out of our scope, because if she blocks the

communication and that ends up the failure of updates, it would be easier to detect the adversary, which is not what cryptography deals with. In the case of local table updates, we assume blackbox adversary during table updates as in [BU21]. When it comes to how often the table needs to be updated in practice, we leave it as an open problem. In addition, we just showed some examples about how to update tables, and we cannot say for sure that they are recommendations. This is because there could exist better methods to trigger the update and to update tables depending on the use cases.

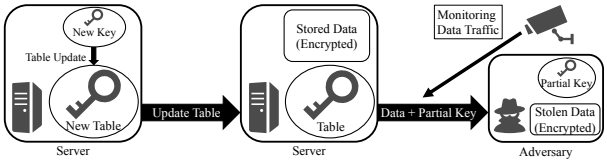


**Figure 7:** How to actually update table

### 5.7 Side Channel and Differential Fault Attacks

Recently, several new side channel attacks have been proposed [GRW20, BHMT16]. As discussed in [BI15], these attacks exploit the fact that each table depends on only a fraction of the key, e.g. 8 and 16 bits of the key. A small part of the key is efficiently extracted using side-channel leakages. On the other hand, any table of Yoroi contains full 128-bit key information. Thus, even if the adversary can fully obtain any side channel information (e.g. memory access patterns) for the target key-dependent table, there are $2^{128}$ possible candidates for each key value. Thus, these are computationally infeasible against Yoroi.

Regarding differential fault attacks [SMdH15], the tables of Yoroi compose of small block ciphers, and the internals of the small block ciphers are inaccessible in the whitebox setting. Thus, any fault injection attack reduces to a differential attack on a small block cipher in the blackbox setting. Since the underlying cipher is secure against a differential attack in the blackbox setting, Yoroi is secure against these attacks.

## 6 Blackbox Security

We evaluate the general construction of Yoroi, modeling the underlying small block cipher as pseudorandom permutation. Specifically, we evaluate security of the general construction of Yoroi, assuming $S_1$, $S_2$, and $S_3$ are independent pseudo random permutations.

### 6.1 Key Recovery Attacks

In the blackbox setting, the adversary has only access to input and the corresponding output of a cryptographic algorithm, and she is unable to collect any pairs of inputs and outputs in the table of $S_1$, $S_2$, and $S_3$. Thus, it is more computationally difficult to mount key recovery attacks on Yoroi in the blackbox setting than to recover the secret key from the underlying block cipher.

### 6.2 Distinguishing attacks

Following the approach in the previous work [FKKM16], Yoroi employs AES at the end of its algorithm. Therefore, security of Yoroi against distinguishing attacks in the blackbox context is reduced to that of AES.

# 7 Implementation

In this section, we present experimental measurements for Yoroi, SPACE [BI15], WhiteBlock [FKKM16], SPNbox [BIT16], WEM [CCD+17], and Galaxy [KSHI20] both in the blackbox setting and the whitebox setting.

We evaluate the implementations of Yoroi and compare the encryption performance of Yoroi with those of SPACE [BI15], WhiteBlock [FKKM16], SPNbox [BIT16], WEM [CCD+17], and Galaxy [KSHI20]. The basis of our comparison is the input space for the table. Specifically, we compare Yoroi-32 with SPACE-32 [BI15], WhiteBlock-32 [FKKM16], SPNbox-32 [BIT16], and Galaxy-32 [KSHI20], and Yoroi-16 with SPACE-16 [BI15], WhiteBlock-16 [FKKM16], SPNbox-16 [BIT16], WEM [CCD+17], and Galaxy-16 [KSHI20], respectively.

We have measured performance for single and parallel encryptions of messages of 2048 bytes and conducted all the performance measurements on a single core with Turbo Boost disabled and over 100000 repetitions. Regarding single encryption operation, we show the performance of implementations which encrypt one 128-bit data block of the message sequentially. Concerning parallel encryption operation, we show the implementation performance in which multiple data blocks are processed simultaneously. For each repetition, a new message was used to avoid unnecessary cache locality. We have taken time measurements from the beginning of the repetition to its ending, and we have also measured message loading and storing time.

We have conducted all the experiments with a machine which has Intel Xeon-8260 2.40GHz and 256GB DDR4 RAM. The processor on the machine has 32KB L1 data cache 32KB L1 instruction cache, 1MB L2 cache, and 35MB L3 cache, respectively. Moreover, it supports the AES instruction set [Gue10] and the SSE instructions up to AVX512.

We have employed AES implemented with AES-NI [Gue10] as the underlying block cipher for SPACE [BI15] and WhiteBlock [FKKM16], and WEM [CCD+17], and use chacha [Ber08] for Galaxy [KSHI20]. For the small-scale block cipher $E_K$ and $D_K$, we have utilized SmallPresent-3 [Lea10] and SmallPresent-7 [Lea10] for Yoroi-16 and Yoroi-32, respectively. In particular, we have followed the implementation method for the underlying cipher of SPNbox and Yoroi in the blackbox context as shown in [BIT16]. Namely, whenever possible, we have employed AES-NI [Gue10] to implement the underlying cipher. For the implementation of linear layer $\theta$ in Yoroi, we have employed an efficient technique for constant-time parallel multiplication operation which is illustrated in [BIT16] for SPNbox and for Yoroi with a slight adjustment. Especially, we have implemented constant-time multiplication of register `%xmm0` by two in the following.

```
vpcmpgtb CONST, %xmm0, %xmm1
vpslld $1, %xmm0, %xmm0
vpand REDPOLY, %xmm1, %xmm1
vpxor %xmm0, %xmm1, %xmm0
```

where `CONST` contains four 32-bit or eight 16-bit copies of value $7_x$ and `REDPOLY` contains four 32-bit or eight 16-bit copies of reduction polynomial, i.e. $13_x$ for Yoroi-32 and Yoroi-16, respectively. We have compiled the source codes with GCC 9.3.0 in O3 optimization level.

The evaluation results in the blackbox setting and whitebox setting are summarized in Table 1 and Table 2, respectively. They show that Yoroi provides sufficient security both in the blackbox setting and whitebox setting, and ensures the longevity without the significant performance loss. Note that we exclude WEM [CCD+17] and Galaxy [KSHI20] from our comparison in the blackbox setting, as they use Fisher-Yates shuffle and chacha as underlying ciphers, which is significantly slow in the blackbox implementation.

*Discussion.* Table 1 shows that Yoroi in the blackbox context is competitive with other existing schemes. In particular, Yoroi-16 in the blackbox context is faster than SPNbox-16

**Table 1:** Evaluation of encryption performance and comparison with existing algorithms in the blackbox setting where the encryption performance is given in cycle per byte

| Algorithm | Round $R$ | Single Encryption | Parallel Encryption |
|---|---|---|---|
| Yoroi-32 | 16 | 56.28 | 16.39 |
| Yoroi-16 | 8 | 108.71 | 36.58 |
| SPNbox-32 [BIT16] | 10 | 35.04 | 10.45 |
| SPNbox-16 [BIT16] | 10 | 135.86 | 42.59 |
| WhiteBlock-32 [FKKM16] | 34 | 133.53 | 70.96 |
| WhiteBlock-16 [FKKM16] | 18 | 81.86 | 55.37 |
| SPACE-32 [BI15] | 128 | 289.95 | 124.41 |
| SPACE-16 [BI15] | 128 | 290.15 | 124.58 |

**Table 2:** Evaluation of encryption performance and comparison with existing algorithms in the whitebox setting where the encryption performance is given in cycle per byte

| Algorithm | Round $R$ | Table size $T$ | Single Enc | Parallel Enc |
|---|---|---|---|---|
| Yoroi-32 | 16 | 48 GB | 803.57 | 275.98 |
| Yoroi-16 | 8 | 384 KB | 33.53 | 12.44 |
| SPN-32 [BIT16] | 10 | 16 GB | 474.39 | 156.01 |
| SPN-16 [BIT16] | 10 | 128 KB | 36.68 | 13.42 |
| WhiteBlock-32 [FKKM16] | 34 | 64 GB | 1114.52 | 334.34 |
| WhiteBlock-16 [FKKM16] | 16 | 2 MB | 156.05 | 39.22 |
| SPACE-32 [BI15] | 128 | 48 GB | 2259.92 | 606.09 |
| SPACE-16 [BI15] | 128 | 896 KB | 243.58 | 59.49 |
| Galaxy-32 [KSHI20] | 32 | 16 GB | 767.90 | 244.85 |
| Galaxy-16 [KSHI20] | 20 | 128 KB | 25.63 | 9.21 |
| WEM [CCD+17] | 12 | 13 MB | 232.84 | 48.29 |

and SPACE-16 both in single encryption operation and parallel encryption operation. This is basically because Yoroi-16 has a smaller number of total rounds $R$ than SPNbox-16 and SPACE-16. The reason why Yoroi-16 is slower in single encryption and faster in parallel operation than WhiteBlock-16 lies in the difference of underlying cipher and diffusion layer. Yoroi-16 employs a dedicated small block cipher and $4 \times 8$ MDS layer as an underlying cipher and its diffusion layer, while WhiteBlock-16 uses AES both for the underlying cipher and diffusion layer. Above all, WhiteBlock-16 makes 4 calls to full-round AES as the underlying cipher based on different partial inputs in one round, which enables independent computation of AES. As a result, WhiteBlock-16 can make the most of the instruction-level pipeline even in single encryption. On the other hand, due to the design of a dedicated small block cipher, Yoroi-16 has to make 2 calls of `aesenc` instruction with some overhead 32 times sequentially in the situation where there is unavoidable latency caused by data dependency between `aesenc` instruction and the overhead. That prevents the full potential of the instruction-level pipeline, and consequently, in single encryption, Yoroi-16 is slower than WhiteBlock-16. However, in parallel encryption where multiple data blocks are processed simultaneously, the latency caused by data dependency between the instruction in Yoroi-16 is diminished to the minimum. WhiteBlock-16 does not benefit from parallel encryption more than Yoroi-16, because WhiteBlock already takes full advantage of parallelism in single encryption. What's more, $4 \times 8$ MDS layer in Yoroi-16 is obviously faster than full-round AES used as diffusion in WhiteBlock-16, and Yoroi-16 has smaller rounds than WhiteBlock-16. Therefore, Yoroi-16 outperforms WhiteBlock-16 in parallel encryption. Concerning Yoroi-32, it is faster than SPACE-32 and WhiteBlock-32

and slower than SPNbox-32, mainly because of the number of total rounds.

Table 2 demonstrates that Yoroi in the whitebox context competes with other ciphers too. Particularly, compared with SPNbox-16, Yoroi-16 is faster because it has smaller rounds than SPNbox-16. Regarding WhiteBlock-16, SPACE-16, and WEM, they are slower than Yoroi-16 largely because their tables are so large that they can be loaded in the L3 cache, while tables of Yoroi-16 may be loaded on the L2 cache. The reason why Galaxy-16 is faster than Yoroi-16 is the difference in the diffusion layer between these two. Galaxy-16 uses a simple shuffle layer which costs much less clock cycle than the $4 \times 8$ MDS layer employed in Yoroi-16. As a result, the encryption performance of Galaxy-16 is better than that of Yoroi-16, even though the number of table lookups in the algorithm in Galaxy-16 is larger than that of Yoroi-16. In the comparison with 32-bit variants where table lookups are the dominant cost for their performance, Yoroi-32 performs quite well. Specifically, Yoroi-32 performs faster than WhiteBlock-32 and SPACE-32 because the number of table lookups in Yoroi-32 (i.e. 64 times of table lookups) is smaller than those of WhiteBlock-32 (68 times)and SPACE-32 (128 times). Moreover, WhiteBlock-32 employs AES as a diffusion layer which costs much more than $4 \times 4$MDS layer in Yoroi-32, which leads to an overall performance gap in the encryption performance between Yoroi-32 and WhiteBlock-32. When it comes to Galaxy-32, it is slightly faster. This is because of the difference of diffusion layer: simple shuffle operation in Galaxy-32 and $4 \times 4$ MDS layer in Yoroi-32, while the number of table lookups in the algorithm is the same. Also, note that Yoroi-16 is deployable on IoT devices such as Raspberry pi whose RAM size is from 512 MB to 8 GB.

# 8 Conclusion

In this work, we introduced the new property *longevity* for the whitebox ciphers which is vital in some cases. This property ensures *constant* security against code-lifting attacks. With this property in our mind, we designed a family of whitebox ciphers Yoroi. Not only does Yoroi provide the same security level both in the blackbox and whitebox settings with the existing whitebox ciphers, Yoroi ensures longevity by applying a small-scale block cipher to the incompressible tables used in ciphers with the secret key for the cipher continuously updated. Moreover, we show that the encryption performance of Yoroi is competitive with existing ciphers both in the blackbox and whitebox setting.

# Acknowledgement

The authors would like to thank Chris Brzuska and the anonymous TCHES reviewers for the valuable comments and suggestions. 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

[ADW10] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In Kaoru Kurosawa, editor, *Information Theoretic Security*, pages 1–18, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BABM20] Estuardo Alpirez Bock, Alessandro Amadori, Chris Brzuska, and Wil Michiels. On the security goals of white-box cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):327–357, 2020.

[BBF+20] Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, and Wil Michiels. Security reductions for white-box key-storage in mobile payments. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 221–252. Springer, 2020.

[BBIJ17] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, and Martin Bjerregaard Jepsen. Analysis of software countermeasures for whitebox encryption. *IACR Trans. Symmetric Cryptol.*, 2017(1):307–328, 2017.

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the asasa structure: black-box, white-box, and public-key (extended abstract). In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 63–84, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BD17] Mihir Bellare and Wei Dai. Defending Against Key Exfiltration: Efficiency Improvements for Big-Key Cryptography via Large-Alphabet Subkey Prediction. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 923–940, New York, NY, USA, 2017. Association for Computing Machinery.

[Ber08] D.J. Bernstein. ChaCha, A Variant of Salsa20. In *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers (2008)*, 01 2008.

[BHMT16] 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*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[BI15] Andrey Bogdanov and Takanori Isobe. White-Box Cryptography Revisited: Space-Hard Ciphers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1058–1069, 2015.

[BIT16] 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*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-Key Symmetric Encryption: Resisting Key Exfiltration. In *CRYPTO*, pages 373–402. Springer, 2016.

[BNN+10] Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptogr.*, 56(2-3):141–162, 2010.

[BRVW19] Andrey Bogdanov, Matthieu Rivain, Philip S. Vejre, and Junwei Wang. Higher-order DCA against standard side-channel countermeasures. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 118–141. Springer, 2019.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2018.

[BU21] Alex Biryukov and Aleksei Udovenko. Dummy shuffling against algebraic attacks in white-box implementations. Cryptology ePrint Archive, Report 2021/290, 2021. https://eprint.iacr.org/2021/290.

[CCD+17] Jihoon Cho, Kyu Young Choi, Itai Dinur, Orr Dunkelman, Nathan Keller, Dukjae Moon, and Aviya Veidberg. WEM: A new family of white-box block ciphers based on the even-mansour construction. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2017.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A White-Box DES Implementation for DRM Applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-Box Cryptography and an AES Implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[CMR05] C. Cid, S. Murphy, and M. J. B. Robshaw. Small Scale Variants of the AES. In *Proceedings of the 12th International Conference on Fast Software Encryption*, FSE'05, page 145–162, Berlin, Heidelberg, 2005. Springer-Verlag.

[DFLM18] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):121–149, 2018.

[DLK+14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, page 475–488, New York, NY, USA, 2014. Association for Computing Machinery.

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In Tanja Lange,

Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2013.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and Provable White-Box Primitives. 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*, volume 10031 of *Lecture Notes in Computer Science*, pages 159–188, 2016.

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *J. Cryptogr. Eng.*, 10(1):49–66, 2020.

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):454–482, 2020.

[Gue10] Shay Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set. https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf, 5 2010.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60, 2008.

[KLLM20] Jihoon Kwon, ByeongHak Lee, Jooyoung Lee, and Dukjae Moon. FPL: White-Box Secure Block Cipher Using Parallel Table Look-Ups. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 106–128. Springer, 2020.

[KSHI20] Yuji Koike, Kosei Sakamoto, Takuya Hayashi, and Takanori Isobe. Galaxy: A Family of Stream-Cipher-Based Space-Hard Ciphers. In Joseph K. Liu and Hui Cui, editors, *Information Security and Privacy - 25th Australasian Conference, ACISP 2020, Perth, WA, Australia, November 30 - December 2, 2020, Proceedings*, volume 12248 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2020.

[Lea10] Gregor Leander. Small Scale Variants Of The Block Cipher PRESENT. Cryptology ePrint Archive, Report 2010/143, 2010. https://eprint.iacr.org/2010/143.

[LER+18] Russell W. F. Lai, Christoph Egger, Manuel Reinert, Sherman S. M. Chow, Matteo Maffei, and Dominique Schröder. Simple password-hardened encryption services. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1405–1421, Baltimore, MD, August 2018. USENIX Association.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the*

*22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 644–655, 2015.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):225–255, 2019.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2011.

[SMdH15] Eloi Sanfelix, Cristofaro Mune, and Job de Haas. Unboxing the white-box practical attacks against obfuscated ciphers. Black Hat Europe 2015, 2015.

# A Algorithm of Yoroi

In this section, we give the full algorithms of Yoroi both in the blackbox and whitebox setting. Algorithm 1 and algorithm 2 describe Yoroi in the blackbox context and whitebox context, respectively.

**Algorithm 1** Algorithm of Yoroi in the Blackbox Implemetation

$X^1(=\{x_0^1, x_1^1, ..., x_{\ell-1}^1\}) \Leftarrow$ INPUT
$i \Leftarrow 1$ // initialize (the number of round, counter) to 1
**while** $i \leq R-1$ **do**
  $\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \gamma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
  $\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \sigma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
  $\{lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i)\} \Leftarrow \theta(lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i))$
  $X^{i+1} \Leftarrow X^i$
  $i \Leftarrow i+1$
**end while**// out of loop when $i = R$
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \gamma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \mathcal{A}(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
OUTPUT $\Leftarrow \{x_0^i, x_1^i, ..., x_{\ell-1}^i\}$

# B Proof of Theorem 1

*Proof.* Given $i$, $j$, and $k$ table entries for the table $T_1$, $T_2$, and $T_3$ respectively, the probability that each entry for the table $T_1$, $T_2$, and $T_3$ is included in the known entries can be estimated as $i/2^{n_{in}}$, $j/2^{n_{in}}$, and $k/2^{n_{in}}$, respectively. Hence, the adversary can derive the correct intermediate value in one round transformation by looking up the known entries with the probability of $(i/2^{n_{in}})^\ell$, $(j/2^{n_{in}})^\ell$, or $(k/2^{n_{in}})^\ell$, depending on which table is used in the round transformation. According to the specification in Section 4.2, $T_1$ and $T_3$ are used for the first and last round in Yoroi, respectively, and for the rest of rounds $T_2$ is used. Therefore, the probability that the adversary can correctly encrypt (decrypt) a randomly chosen plaintext (ciphertext) is bounded by

$$Pr. = \left(\frac{i}{2^{n_{in}}}\right)^\ell \times \left(\frac{j}{2^{n_{in}}}\right)^{\ell\times(R-2)} \times \left(\frac{k}{2^{n_{in}}}\right)^\ell,$$

□

**Algorithm 2** Algorithm of Yoroi in the Whitebox Context

$X^1(=\{x_0^1, x_1^1, ..., x_{\ell-1}^1\}) \Leftarrow$ INPUT
$(i, j) \Leftarrow (1, 0)$ // initialize (the number of round, counter) to 1 and 0
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \gamma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
**while** $j \leq \ell - 1$ **do**
  $msb_m(x_j^i) \Leftarrow E_K(msb_m(x_j^i))$
  $j = j + 1$
**end while**
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \sigma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
$\{lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i)\} \Leftarrow \theta(lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i))$
$X^{i+1} \Leftarrow X^i$
$(i, j) \Leftarrow (i + 1, 0)$
**while** $i \leq R - 1$ **do**
  **while** $j \leq \ell - 1$ **do**
    $msb_m(x_j^i) \Leftarrow D_K(msb_m(x_j^i))$
    $j = j + 1$
  **end while**
  $\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \gamma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
  $\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \sigma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
  $\{lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i)\} \Leftarrow \theta(lsb_t(x_0^i), lsb_t(x_1^i), ..., lsb_t(x_{\ell-1}^i))$
  $j \Leftarrow 0$
  **while** $j \leq \ell - 1$ **do**
    $msb_m(x_j^i) \Leftarrow E_K(msb_m(x_j^i))$
    $j = j + 1$
  **end while**
  $X^{i+1} \Leftarrow X^i$
  $(i, j) \Leftarrow (i + 1, 0)$
**end while**// out of loop when $i = R$
**while** $j \leq \ell - 1$ **do**
  $msb_m(x_j^i) \Leftarrow D_K(msb_m(x_j^i))$
  $j = j + 1$
**end while**
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \gamma^i(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
$\{x_0^i, x_1^i, ..., x_{\ell-1}^i\} \Leftarrow \mathcal{A}(x_0^i, x_1^i, ..., x_{\ell-1}^i)$
OUTPUT $\Leftarrow \{x_0^i, x_1^i, ..., x_{\ell-1}^i\}$

# C Compression Attack on $T_1$.

The update from $T_1^{j-1}$ to $T_1^j$ requires the computation of $E_{K_j}(S_1(x))$ in which all the $2^{n_{in}}$ output values for table $S_1$ are updated while inputs of tables do not change. As described in Section 4, $E_{K_j}$ takes the most significant $m$ bits and ignore the least significant $(n_{in} - m)$ bits of output value $v_k$ from $S_1(x)$ where $0 \leq k \leq 2^{n_{in}} - 1$. Hence, $2^{n_{in}-m}$ different values of $v_k$ are treated as the same value of $v^{'}$ as inputs of $E_{K_j}$. Namely, the computation of $E_{K_j}$ on $2^{n_{in}-m}$ different outputs from $S_1(x)$ produces the outputs whose upper $m$ bits are exactly the same. This enables the adversary to compress the table and find the smaller equivalent implementation of $T_1^j$ by the following method which is also described in Fig. 8 in the case where $n_{in} = 5$ and $m = 4$.

**a)** First of all, the adversary obtains information about the changes in the upper $m$-bit value of $v_k$ for outputs in the same input (i.e. index) of $T_1^{j-1}$ and $T_1^j$ by comparing the outputs (the index is "00000" in Fig. 8.)

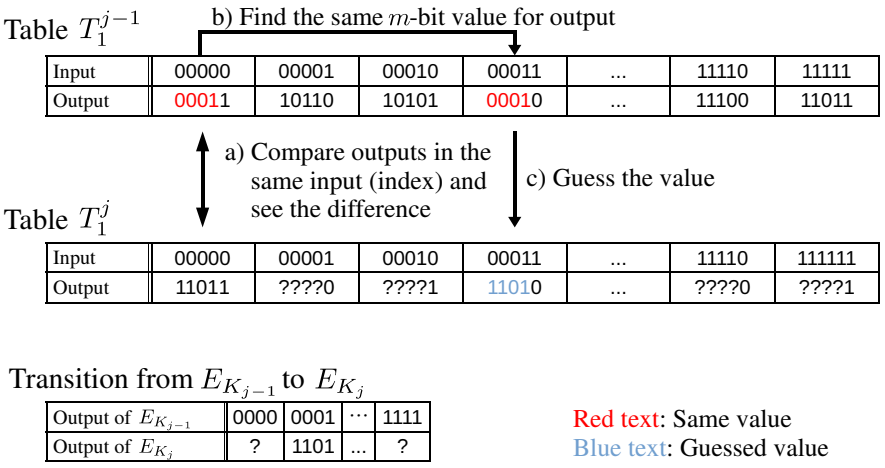


**Figure 8:** Compression Technique for $T_1$ in the case where $n_{in} = 5$ and $m = 4$

**b)** She finds an output in another index, which is "00011" in Fig. 8, for table $T_1^{j-1}$ whose upper $m$-bit value is exactly the same as the one in the index "00000" of table $T_1^{j-1}$ which she already knows.

**c)** In this case, since she knows the transition from an output value in an index "00000" for table $T_1^{j-1}$ to that in the same index for table $T_1^j$, she can correctly guess an output in the other index "00011" for table $T_1^j$ by using the output in the same index "00011" for table $T_1^{j-1}$ and the transition information.

As shown in the above example, once she knows one output of table $T_1^j$, she can correctly guess some corresponding outputs of updated table $T_1^j$. Specifically, if she has the knowledge about one output of table $T_1^j$, she can accurately guess $(2^{n_{in}-m} - 1)$ different outputs of table $T_1^j$.

Therefore, in order to maintain the same functionality of $T_1^j$, she needs at least $2^m$ outputs of table $T_1^j$. With those information, she can infer $(2^{n_{in}} - 2^m)$ $(= 2^m \times (2^{n_{in}-m} - 1))$ different entries.

Now, each entry of the table is $n_{in}$ bits in size. However, only the upper $m$ bits of value $v$ are updated by applying $E_{K_j}$ to function $S_1(x)$ in order to update the table $T_1^{j-1}$ to $T_1^j$, and the lower $n_{in} - m$ bits of $v$ are constant. Thus, due to the assumption that the adversary knows about all the previous tables, she already has the knowledge about the lower $n_{in} - m$ bit of the value $v$ for every entry she only has to obtain the information about the upper $m$ bits of $n_{in}$-bit value $v$. As a result, the total size which she has to obtain in order to find the smaller equivalent representation of Yoroi is $2^m \times m$ bits.

# D Compression Attack on $T_3$.

In order to update from $T_3^{j-1}$ to $T_3^j$, it is necessary to compute $S_3(D_{K_j}(x))$. What this computation actually does is just to change all the $2^{n_{in}}$ indexes of table $T_3^{'}$, and it does nothing to all the output values for $T_3^{'}$, where $T_3^{'}$ is a table comprising pairs of all the input and corresponding outputs of $S_3$. In other words, it just permutes all the $2^{n_{in}}$ entries of $T_3^{'}$. However, since $D_{K_j}$ takes the most significant $m$ bits and ignore the $2^{n_{in}-m}$ bits of the input for $S_3$, this computation is not a permutation of $2^{n_{in}}$ table entries, but that of

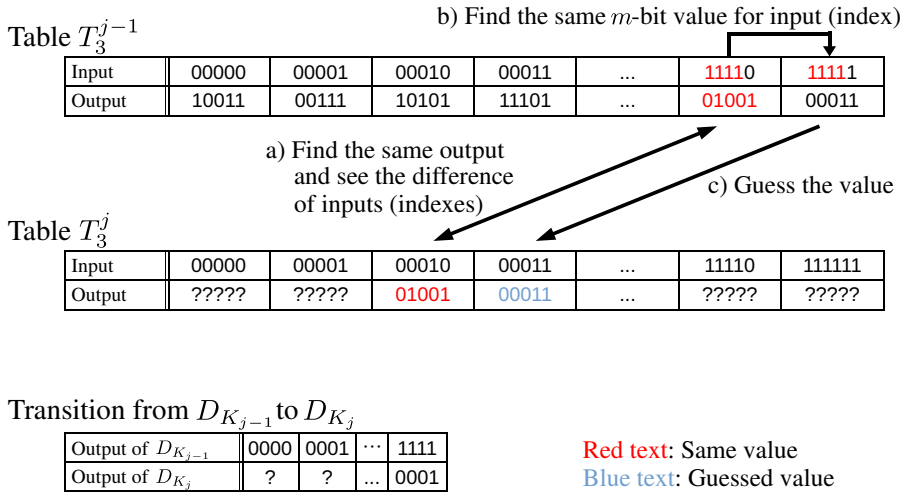


**Figure 9:** Compression Technique for $T_3$ in the case where $n_{in} = 5$ and $m = 4$

$2^m$ table entries. As a result, the order for each $2^{n_{in}-m}$ of entries in the tables remains constant. This gives the possibility for the table compression by the following method which is also described in Fig. 9.

**a)** To begin with, the adversary finds the same output from tables $T_3^{j-1}$ and $T_3^j$, and gains the knowledge about a difference of indexes for the output in tables $T_3^{j-1}$ and $T_3^j$.

**b)** She finds some indexes in table $T_3^{j-1}$ whose upper $m$-bit value is the same.

**c)** Since the order for each $2^{n_{in}-m}$ of entries in $T_3^j$ and the outputs for these entries are constant, she can guess the outputs for them.

As long as she can accurately guess $(2^{n_{in}-m} - 1)$ different entries of table $T_3^j$ with the knowledge about one entry of table $T_3^j$, she does not have to gain these information, which means she can reduce the amount of information about the table $T_3^j$ to steal so that she can find the same functionality. As a result, in order to find the same functionality of $T_3^j$, she needs at least $2^m$ entries of table $T_3^j$, because with those information, she can infer $(2^{n_{in}} - 2^m)$ $(= 2^m \times (2^{n_{in}-m} - 1))$ different entries. This compression method implicitly demonstrates that after the table update, the adversary can find the equivalent function of updated table $T_3^j$ by using previous table $T_3^{j-1}$ and collecting information about the transition from outputs of $D_{K_{j-1}}$ to those of $D_{K_j}$ whose size is $2^m \times m$.

# E Compression attack on $T_2$.

Compared with compression attacks on tables $T_1$ and $T_3$, it may seem relatively difficult to compress table $T_2$ merely by compareing $T_2^{j-1}$ and $T_2^j$ just as we did on tables $T_1$ and $T_3$. This is because the update from $T_2^{j-1}$ to $T_2^j$ requires the computation of $E_K(S_2(D_K(x)))$. Namely, the output values for table $S_2$ are updated by $E_{K_j}$, and the table entries are shuffled by $D_{K_j}$.

However, information about $E_{K_j}$ and $D_{K_j}$ can be collected by mounting compression attacks on $T_1$ and $T_3$. If the adversary analyzes and compress tables $T_1$ and $T_3$, she can gain the information about what each output value for table $T_2^{j-1}$ is updated to and where each entry of $T_2^{j-1}$ is moved to. Therefore by using the information about transion

from outputs of $E_{K_{j-1}}$ to those of $E_{K_j}$ and that from outputs of $D_{K_{j-1}}$ to those of $D_{K_j}$ which are the actual information about $E_{K_j}$ and $D_{K_j}$, and the previous table $T_2^{j-1}$, the adversary can realize the same functionality of $T_2^j$. As a result, if the adversary analyzes tables $T_1$ and $T_3$, she does not have to gain any information $T_2^j$ at all, which means she can compress table $T_2^j$ to zero.

# F Proof of Theorem 2

*Proof.* As mentioned, if she obtains one entry of the table $T_1^j$ (or $T_3^j$), she can infer at most $(2^{n_{in}-m}-1)$ entries. That is virtually equivalent to the situation where she knows at most $2^{n_{in}-m}$ entries just by obtaining one entry of the table. So, given the case where the adversary steals a part of the table, i.e., $g \leq 2^m$ entries for $T_1^j$ are leaked to her, the number of total entries that are virtually leaked to her is up to $(g \times 2^{n_{in}-m})$.

When estimating the total entries of a whole table as $2^{n_{in}}$, the probability that each entry for table $T_1^j$ is included in the $(g \times 2^{n_{in}-m})$ leaked entries is $(g \times 2^{n_{in}-m})/2^{n_{in}}$. Thus, the probability that she can derive the correct output by looking up the leaked entries of table is $(g \times 2^{n_{in}-m})/2^{n_{in}}$. Considering the fact that Yoroi uses table $T_1^j$ $\ell$ $(= n/n_{in})$ times to look up values in one round where $n$ is the block size for Yoroi, the probability that the adversary can obtain the correct intermediate value by looking up the tables $l$ times is given as:

$$\Big(\frac{g \times 2^{n_{in}-m}}{2^{n_{in}}}\Big)^{\ell}.$$

Besides, with the same logic, by using $h$ entries for table $T_3^j$, the probability that the adversary can compute the correct intermediate value is:

$$\Big(\frac{h \times 2^{n_{in}-m}}{2^{n_{in}}}\Big)^{\ell}.$$

In order to infer some table entries of $T_2^j$, the adversary at least has to know about $E_{K_j}$ and $D_{K_j}$. Information about $E_{K_j}$ tells her what each output of the table $T_2^{j-1}$ is updated to, and that of $D_{K_j}$ tells what each index of the table $T_2^{j-1}$ is updated to, i.e., where each updated output is moved to. As mentioned, leakage of one entry leads to that of $2^{n_{in}-m}$ entries. Specifically, with the information about transition from $E_{K_{j-1}}$ to $E_{K_j}$, one leaked entry enables the adversary to infer $2^{n_{in}-m}$ outputs of table $T_2^j$ and with that the information about transition from $D_{K_{j-1}}$ to $D_{K_j}$ each leaked entry possibly enables her to infer indexes, i.e., positions for $2^{n_{in}-m}$ outputs in the table $T_2^j$. Because of the design of table $T_2$, the adversary has to gain both information about these types of transition.

In the case of $g < h$ the adversary can possibly gain the indexes for $h \times 2^{n_{in}-m}$ outputs. However, she can infer what only $g \times 2^{n_{in}-m}$ outputs of the table $T_2^{j-1}$ are updated to. This means that she can infer only the indexes for $(h-g) \times 2^{n_{in}-m}$ $(= (h \times 2^{n_{in}-m}) - (g \times 2^{n_{in}-m}))$ outputs and can not know the actual values for them.

In the case of $g > h$, the adversary can infer $g \times 2^{n_{in}-m}$ output values for the table $T_2^j$, while she can infer the indexes for only $h \times 2^{n_{in}-m}$ outputs. Thus, even though she can infer infer $g \times 2^{n_{in}-m}$ output values, she can not know the indexes for $(g-h) \times 2^{n_{in}-m}$ $(= (g \times 2^{n_{in}-m}) - (h \times 2^{n_{in}-m}))$ outputs for the table $T_2^j$

Given the discussion above, the number of table entries (i.e. updated table outputs and their correct indexes) the adversary can gain is up to $x \times 2^{n_{in}-m}$, where $x = min(g, h)$. Also with the same logic discussed in table $T_1^j$ and $T_3^j$, the adversary can obtain the intermediate value by using leaked information about $T_2^j$ with the probability of up to:

$$\left(\frac{x \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{\ell}.$$

In order to compute a correct output, the adversary has to obtain a correct intermediate value in any round. In $R$ rounds, Yoroi uses the table $T_1^j$ and $T_3^j$ for the first and last round respectively, and for the rest of rounds $T_2^j$ is employed, where $R$ is the total rounds for Yoroi. Therefore, after $R$ rounds, the adversary can compute the correct output from the random-drawn input with the probability of

$$\left(\frac{g \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l} \times \left(\frac{x \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l\times(R-2)} \times \left(\frac{h \times 2^{n_{in}-m}}{2^{n_{in}}}\right)^{l},$$

□