IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 3, pp. 368–402. DOI:10.46586/tosc.v2022.i3.368-402

Hybrid Code Lifting on Space-Hard Block Ciphers

Application to Yoroi and SPNbox

Yosuke Todo[1] and Takanori Isobe[2]

[1] NTT Social Informatics Laboratories, Tokyo, Japan
yosuke.todo.xt@hco.ntt.co.jp
[2] University of Hyogo, Kobe, Japan
takanori.isobe@ai.u-hyogo.ac.jp

Abstract. There is a high demand for whitebox cryptography from the practical use of encryption in untrusted environments. It has been actively discussed for two decades since Chow et al. presented the whitebox implementation of DES and AES. The goal is to resist the *key extraction* from the encryption program and mitigate the *code lifting* of the program. At CCS2015, Bogdanov and Isobe proposed space-hard block ciphers as a dedicated design of whitebox block ciphers. It ensures that the key extraction is as difficult as the key recovery in the standard blackbox model. Moreover, to mitigate code lifting, they introduce space hardness, a kind of leakage-resilient security with the incompressibility of a huge program. For space-hard ciphers, code lifting (a partial leakage of the entire program) is useless to copy the functionality. In this paper, we consider a new attack model of space-hard block ciphers called *hybrid code lifting.* Space-hard block ciphers are intended to ensure security under a size-bounded leakage. However, they do not consider attackers (in the standard blackbox model) receiving the leakage by code lifting. If such attackers can recover the encryption program of a space-hard block cipher, such a cipher does not always satisfy the intention. We analyze Yoroi proposed in TCHES 2021. We introduce the canonical representation of Yoroi. Using the representation enables the recovery of the programs of Yoroi-16 and Yoroi-32 with 2^{33} and $2^{65.6}$ complexities, respectively, in spite of slight leakage. The canonical representation causes another attack against Yoroi. It breaks an authors' security claim about the "longevity". We additionally analyzed SPNbox proposed in Asiacrypt 2016. As a result, considering security on the hybrid code lifting, the original number of rounds is insufficient to achieve 128-bit security under quarter-size leakage.

Keywords: Whitebox cryptography · Space-hard block cipher · Code lifting · Blackbox analysis · Truncated differential · Secret S-box recovery · Longevity

1 Introduction

The use of block ciphers has become common in various environments. If block ciphers work in unreliable environments, attackers can access or modify their implementations. They exploit unavailable information for attackers in the blackbox model. In particular, attackers being allowed unlimited access or modification of their implementations are the strongest that can be assumed. We call such an attack model a *whitebox model. Whitebox cryptography* aims to ensure security against attackers in the whitebox model. The high demand for whitebox cryptography has been discussed, particularly in the software environment [All14, int18].

Chow et al. introduced whitebox cryptography two decades ago [CEJvO02a, CEJvO02b]. They provided *whitebox implementations* of block ciphers DES and AES. The primary goal is to make it difficult for an attacker in the whitebox model to extract the secret

Received: 2022-06-01 Accepted: 2022-08-01 Published: 2022-09-09

key from the implementation. The basic idea is to implement DES or AES by only continual lookups in several tables embedded with round keys. Random linear and nonlinear transformations are applied before and after each table to hide round keys from tables. Since the seminal paper by Chow et al., many other whitebox implementations have been proposed [BCD06, Kar10, LN05]. Unfortunately, almost all have been broken [BGE04, WMGP07, MWP10, MRP12, LRM$^+$13, con17]. Recently, Bock et al. pointed out that even attackers in a graybox model (limited whitebox model) are sufficient to extract the secret key from some whitebox implementations [BHMT16, BBB$^+$19]. Therefore, state-of-the-art whitebox implementations aim to ensure security against attackers in such a limited whitebox model [BBIJ17, BU18, CC19, BU21].

Another direction is designing dedicated whitebox block ciphers, whose whitebox implementations are easy [BBK14, BI15, BIT16, FKKM16]. A space-hard block cipher proposed by Bogdanov and Isobe [BI15] is one of the successful ciphers in this direction. Like whitebox implementations, they use a table, but the table is generated by a secure block cipher (such as AES). A whitebox attacker can observe the table, but extracting the secret key is equivalently difficult to the key-recovery attack in the blackbox model. Thus, we expect that space-hard block ciphers are secure against the key extraction.

Therefore, the main interest of space-hard block ciphers moves to mitigate code lifting, another whitebox attack model. The goal is to isolate the program and copy it instead of the secret key. To mitigate the code lifting, Bogdanov and Isobe introduced *space hardness* [BI15]. When the size of the program (table) is T, it guarantees that the probability that random plaintexts are successfully encrypted is at most 2^{-Z} by using $M(\ll T)$ partial table entries. The intuitive understanding is leakage-resilient security. Even if a whitebox attacker looks at the table and extracts M-bit information from the table, the extracted data does not help to copy the encryption program. Considering the table size ranges from KB to GB orders in common space-hard block ciphers, even the partial data (usually, quarter-size, i.e., $M/T = 2^{-2}$) is large, and leaking them is not easy to hide from users. Nowadays, many *space-hard block ciphers* have been proposed [BIT16, FKKM16, CCD$^+$17, KSHI20, KLLM20].

At TCHES 2021 [KI21], a new dedicated space-hard block cipher, Yoroi, was proposed. Yoroi has a new functionality called *longevity* beyond conventional space-hard block cipher. It enables us to update the table, and the functionality as block cipher is compatible before and after updating the table. Specifically, ciphertexts (generated by the old table) can be decrypted using the updated table. The goal is to ensure security against the following attack. An attacker leaks slight data about the table over a long time to avoid being found by users. For example, assuming the attacker leaks 16MB every day, 1600MB of data can be collected in 100 days. Eventually, the attacker can collect all table entries. An updatable table (but the secret key is not updated) is promising to address this attack. Once the table of Yoroi is updated, a whitebox attacker needs to restart leaking table entries from the beginning. To our knowledge, Yoroi is the only such cipher with this functionality.

1.1 Our Contribution

Hybrid Code Lifting. Considering the intention of the leakage-resilient security of space-hard block ciphers, we introduce a new attack model called *hybrid code lifting*. Our attack model is regarded as the hybrid of blackbox and whitebox models.

In the first phase, an attacker is in the whitebox model, looks at and analyzes the implementation, and leaks size-bounded arbitrary data. In the second phase, a collaborative blackbox attacker receives the leakage and analyzes the block cipher in the standard blackbox model. We say that the block cipher is insecure against the hybrid code lifting if the collaborative attacker can recover the encryption program faster than an exhaustive search of the secret key by exploiting the leakage.

Table 1: Summary of hybrid code lifting on Yoroi and SPNbox.

target	code-lifting phase		blackbox phase	remark	reference
	time	leak bit size (ratio)	complexity‡		
Yoroi-16	$2^{18.8}$	800 ($2^{-11.94}$)	2^{33}	verified practically	Sect. 5
Yoroi-32	$2^{35.9}$	3008 ($2^{-27.03}$)	$2^{65.5}$		Sect. 5
SPNbox-16	2^{14}	16×2^{14} (1/4)	$2^{124.09}$		Sect. 7
SPNbox-24	2^{22}	24×2^{22} (1/4)	$2^{102.27}$		Sect. 7
SPNbox-32	2^{30}	32×2^{30} (1/4)	$2^{95.84}$		Sect. 7

Complexity‡ represents the time and data complexities to recover the encryption program from the leaked information.

Table 2: Summary of attacks on the longevity of Yoroi.

target	code-lifting phase			complexity‡	remark	reference
	model	time	#updates			
Yoroi-16	arbitrary†	$2^{18.8}$	171	negl.		Sect. 6.2
Yoroi-32	arbitrary†	$2^{35.9}$	342	negl.		Sect. 6.2
Yoroi-16	known space	-	$2^{35.97}$	$2^{48.78}$	break claimed security	Sect. 6.3
Yoroi-32	known space	-	$2^{68.95}$	$2^{98.86}$	break claimed security	Sect. 6.3

Arbitrary† represents a whitebox attacker w/o nonvolatile memory.
Complexity‡ represents the time complexity to recover the encryption program from collected leakages, and a query is not required.

At first glance, space-hard block ciphers look secure against hybrid code lifting. However, this intuition is not valid. First, the space hardness assumes leaking table entries directly. Next, it does not assume a blackbox attacker receiving the leakage. We believe that our attack model is natural and not extraordinarily strong for the security of space-hard ciphers. In practice, the *strong incompressibility* [FKKM16], which is a security model of whitebox encryption scheme, assumes similar attackers. Note that, as the authors of [FKKM16] already say, the strong incompressibility is not introduced for space-hard block ciphers. In practice, any space-hard block cipher does not satisfy the strong incompressibility because of a trivial attack. Our attack model can be regarded as the revision of the strong incompressibility so that it is compatible with space-hard block ciphers and still can be demanded.

Applying Hybrid Code Lifting to Yoroi and SPNbox. To discuss the impact of our attack model, we apply this attack to Yoroi [KI21], which was recently proposed in TCHES 2021, in Sect. 5. Table 1 summarizes our attacks. If we expect Yoroi to be secure until a quarter-size leakage of the encryption program, 96 KB and 12 GB in Yoroi-16 and Yoroi-32, respectively, we fall short of the expectation by our attacks. Our leakage size is significantly smaller than the quarter size. Specifically, only 800-bit leakage, whose ratio is $800/(3 \times 2^{16} \times 16) \approx 2^{-11.94}$, is sufficient to recover the encryption program of Yoroi-16. Besides, the time complexity to attack Yoroi-16 is even practical. We need to say that this attack is outside the authors' security claims. However, we believe that their claimed security is too optimistic to claim that the practical use case of Yoroi can be secure.

We applied the hybrid code lifting to another space-hard block cipher SPNbox [BIT16] in Sect. 7. Unlike Yoroi, we do not find an extreme vulnerability. However, the original number of rounds would not achieve 128-bit security under quarter-size leakage.

Breaking Claimed Security of Yoroi. We also show another attack against Yoroi in Sect. 6. This attack breaks one of the authors' security claims. The attack target is the longevity of Yoroi. An attacker collects many leakages every table update. The attacker analyzes these leakages and tries to recover the encryption program without querying plaintexts/ciphertexts to encryption/decryption oracles. Table 2 summarizes our attacks. We can recover the encryption program by leakage based on the known-space attack model [BIT16, KI21], which the authors of Yoroi claimed as infeasible. Thus, this attack breaks one of the authors' security claims.

Organization. The rest of this paper is organized as follows. Section 2 introduces whitebox cryptography and space-hard block ciphers. We introduce a hybrid code lifting, a new attack model on space-hard block ciphers, in Sect. 3. The preliminaries for our attacks against Yoroi are summarized in Sect. 4. Sections 5 and 6 show the hybrid code lifting and attacks against the longevity, respectively. The hybrid code lifting on SPNbox is introduced in Sect. 7. Finally, Section 8 concludes the paper.

2 Whitebox Cryptography and Space-Hard Block Cipher

2.1 Block Cipher and Its Whitebox Security

Definition 1 (Block cipher)**.** A block cipher is a function $E : \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^{n} \to \mathbb{F}_2^{n}$, where κ and n denote a key length and block length, respectively. The function E is invertible. Then, there is the decryption function $D : \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^{n} \to \mathbb{F}_2^{n}$ such that $D(K, E(K, P)) = P$ for all $P \in \mathbb{F}_2^{n}$. We denote by E_K and D_K as encryption and decryption of the block cipher with the fixed secret key $K \in \mathbb{F}_2^{\kappa}$.

We introduce the term *program* inherited from [DLPR13]. We use a program as the word in the language-theoretic sense. A program is interpreted in the explicit context of programming and execution models. Successive executions are stateless, i.e., it returns a deterministic output given a fixed input.

There is an *efficient program* to implement the encryption/decryption of the block cipher, where an efficient program denotes a program that is implementable by reasonable resources and returns output with reasonable time on a modern computer. An attacker can access an encryption (and/or decryption) oracle in the blackbox model. In contrast, in the whitebox model, an attacker can additionally access the program of the block cipher unlimitedly. Specifically, supposing that lookup tables are continuously used in the encryption, such as Chow et al.'s implementation or space-hard block ciphers, the attacker can look at the entire table. We introduce two well-discussed security goals for the whitebox security of the block cipher.

2.1.1 Key Extraction

The first-priority goal is to resist the key extraction. The goal of the key extraction is to extract K from the program, where a whitebox attacker can unlimitedly access and modify the program. The straightforward implementation does not resist the key extraction because the whitebox attacker can observe and extract the input of the key schedule. Chow et al. and many subsequent researchers proposed whitebox implementations of DES or AES to resist the key extraction [CEJvO02a, CEJvO02b, BCD06, Kar10, LN05]. Unfortunately, many attacks have been proposed, and almost all implementations were eventually broken [BGE04, WMGP07, MWP10, MRP12, LRM+13]. The state-of-the-art is how to resist attacks in a limited whitebox model such as differential computation analysis [BHMT16, BBB+19] and its generalization [BBIJ17, BU18, BU21].

One of the successful directions resisting the key extraction is the dedicated design of whitebox block ciphers [BBK14, BI15]. For example, a space-hard block cipher [BI15], which is our focus, is such a cipher and guarantees that the key extraction is as hard as the key recovery attack against a common block cipher in the blackbox model. Thus, the main interest of dedicated design moves to the security against the code lifting.

2.1.2 Code Lifting and Related Works for Its Mitigation

The code lifting is to extract the program directly instead of extracting K. Attackers can easily encrypt any message once they successfully extract it. A *space hardness*, described later in detail, is introduced as a mitigation of code lifting in a space-hard block cipher. There are other mitigations, such as *external encoding* [CEJvO02a], *binding* [BBF+20], and *incompressibility* [DLPR13]. Although understanding other mitigations is not always necessary to understand our paper, we briefly introduce them.

The external encoding was suggested by Chow et al [CEJvO02a]. It provides a program of $E' = Q_l \circ E \circ Q_f$ instead of E, where the block cipher is masked by secret functions Q_f and Q_l. We cannot evaluate E using E' without Q_f and Q_l. Note that valid users need to use them when they want to use the block cipher. Therefore, the external encoding is helpful in the environment using the trusted hardware, where Q_f and Q_l are not exposed to attackers in even the whitebox model.

When the execution of block ciphers is bounded by trusted hardware or application, the encryption program does not work outside of the bound environment. Recently, binding using the technique of public-key cryptography have been discussed, e.g., a scheme using indistinguishability obfuscation [BBF+20] or LWE [ABCW21]. Note that binding requires the trusted hardware that attackers never touch in even the whitebox model.

To the best of our knowledge, Delerablée et al. first introduced incompressibility as a security notion for whitebox cryptography in [DLPR13][1]. Given an encryption program, incompressibility says that an attacker cannot compress the encryption program to a program whose size is significantly smaller in the whitebox model. They supposed the security risk in digital rights management (DRM), which is one of the most typical applications of whitebox cryptography. In DRM, attackers own the decryption program to decrypt protected contents, and the risk (of a content provider) is the re-distribution of the decryption program. Assuming the encryption program is large, the re-distribution may be somewhat discouraged due to the huge size. Note that the naive incompressibility is not always useful for the use case of whitebox cryptography except for the DRM [BABM20].

The mitigations above are not the only ones. For example, there are traceability [DLPR13], one-wayness [DLPR13], strong whitebox security [BBK14], and so on. Although we concentrate on the space hardness in this paper, we stress that there are some cases in that other mitigations are more helpful. Space hardness is superior to or inferior to other mitigations in some respects. For example, large program size is necessary for a space-hard block cipher, which is a disadvantage. On the other hand, it does not require trusted hardware, which is an advantage.

2.2 Space-Hard Block Cipher

2.2.1 Space Hardness and Space-Hard Block Cipher

Bogdanov and Isobe introduced the space hardness [BI15], which is similar to the incompressibility but intends more leakage-resilient security than the incompressibility.

Definition 2 ((M, Z)-space hardness [BI15])**.** The implementation of a block cipher E_K is (M, Z)-space hard if it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability higher than 2^{-Z} given any code (table) of size less than M.

[1] Biryukov et al. introduced a similar security notion as *weak whitebox security* in [BBK14].

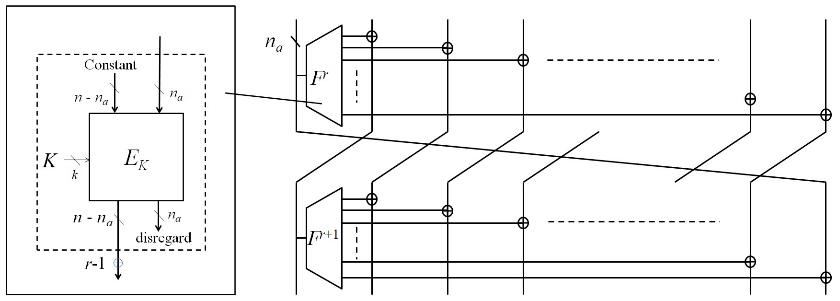

Figure 1: Overview of SPACE.

As shown in existing works (e.g., §5.3.2 in [BI15] or §2.3 in [KI21]), the space hardness is the expected security notion for a leakage-resilient system. Even if an attacker successfully steals part of the entire table, the attacker cannot correctly encrypt (decrypt) a randomly drawn plaintext (ciphertext) with a high probability by using the stolen table entries when the space hardness is guaranteed.

A *space-hard block cipher* is the block cipher satisfying space hardness. The whitebox implementation (program) of a space-hard block cipher is generated by the following compiler.

Definition 3 (Compiler of space-hard block ciphers)**.** A compiler of space-hard block ciphers is a function $\mathbf{C_E} : \mathbb{F}_2^{\kappa} \times \mathcal{R} \to \mathcal{T}$, taking a key $k \in \mathbb{F}_2^{\kappa}$ and possibly a randomness $r \in \mathcal{R}$ drawn from some randomness[2] space $\mathcal{R}$. It outputs a table $T \in \mathcal{T}$. Then, there is a program $\tilde{E}_T$, which has the same functionality as the original block cipher E_K, namely, $\tilde{E}_T(P) = E_K(P)$ for all $P \in \mathbb{F}_2^n$. Note that the size of the table denoted by $size(T)$ is much larger than the size of the original key.

A space-hard block cipher is specified by continual lookups in several tables. The idea is to make tables from well-analyzed block ciphers such as AES for the whitebox implementation.

A block cipher family, SPACE, is the first instantiation [BI15]. SPACE is based on a target heavy Feistel construction in which the F function is generated by a well-analyzed block cipher (AES in their example) with the secret key by constraining the plaintext and truncating the ciphertext. Figure 1 shows the rth and $(r+1)$th round functions of SPACE. The compiler loads the secret key, calculates 2^{n_a} table entries, and outputs the program using the table. In contrast, another implementation uses a short key, where AES runs every round instead of the table. Then, it does not satisfy the space hardness.

To be precise, space hardness depends on the attack mode in the whitebox. Bogdanov et al. introduced three attack models in [BIT16].

Definition 4 (Attack Models [BIT16])**.** Let F be a table used in the whitebox implementation of a space-hard block cipher.

- *Known-Space Attack (KSA)* extracts M pairs of inputs and corresponding outputs of tables $(x_i, F(x_i))$ for $i \in \{1, 2, \ldots, M\}$.
- *Chosen-Space Attack (CSA)* extracts M pairs of inputs and corresponding outputs of tables $(x_i, F(x_i))$ for $i \in \{1, 2, \ldots, M\}$, where x_i is chosen in advance by attackers.
- *Adaptive Chosen-Space Attack (ACSA)* extracts M pairs of inputs and corresponding outputs of tables $(x_i, F(x_i))$ for $i \in \{1, 2, \ldots, M\}$, where x_i is adaptively chosen by attackers. Specifically, x_a can be chosen after obtaining $(x_{a-1}, F(x_{a-1}))$.

[2]Many space-hard block ciphers such as SPACE [BI15], SPNbox [BIT16], Whiteblock [FKKM16] or WEM [CCD+17] do not accept this randomness, but Yoroi [KI21] can accept it to achieve *longevity*.

Cho et al. introduced a more straightforward extraction, where an attacker leaks ciphertexts of chosen plaintexts [CCD$^+$17]. Then, it cannot hope for a space hardness better than $Z = n - \log(T)$ when the attacker can leak $(T \times n)$ bits. Recently published space-hard block ciphers such as Galaxy [KSHI20] or Yoroi [KI21] mainly focus on the space hardness against the KSA/CSA.

Space-hard block ciphers are promising as the ciphers in the leakage-resilient system. However, the space hardness and existing attack models are not sufficient to claim such security. First, we should suppose attackers extracting arbitrary leakage rather than extracting table entries or plaintexts because there is no reason for attackers to restrict their actions in the whitebox model. Moreover, we need to assume a collaborative blackbox attacker receiving the leakage. If the blackbox attacker can recover the encryption program, we cannot say that such a cipher is leakage-resilient secure. These two insufficiencies motivate us to consider a new extended attack model for space-hard block ciphers in Sect. 3.

2.2.2 Strong Incompressibility [FKKM16]

Fouque et al. introduced *strong incompressibility (IND-COM and ENC-COM)* in [FKKM16]. The strong incompressibility treats a symmetric-key encryption scheme rather than a block cipher. Unlike the space hardness, it supposes a blackbox attacker receiving the output of an arbitrary leakage function f. The only limitation imposed to f is that the min-entropy of the key remains sufficiently large after the leakage. Then, IND-COM ensures indistinguishability, and ENC-COM ensures that plaintexts are not successfully encrypted without the encryption oracle. We refer to [FKKM16] for the detailed definitions.

The strong incompressibility provides strong security under the leakage by a whitebox attacker. The authors of [FKKM16] mentioned

> Note that in the following definitions, f is not computationally bounded, so generating the tables via a pseudorandom function is not possible.

before the definition of the strong incompressibility. Space-hard block ciphers generate the table via a pseudorandom function, e.g., AES. Since the function f is not computationally bounded, the attacker can choose the function that exhaustively searches the secret key by checking the consistency of the stored table of space-hard block ciphers. It implies that space-hard block ciphers never satisfy the strong incompressibility.

Remember that strong incompressibility is introduced to obtain provable security. On the other hand, a space-hard block cipher ensures its security based on the analysis of the best attack algorithm like the security of block ciphers.

3 Hybrid Code Lifting on Space-Hard Block Ciphers

As discussed in Sect. 2, the intention of the space hardness is to ensure security under the leakage by a whitebox attacker. However, the space hardness does not suppose a blackbox attacker receiving the leakage. We believe that supposing the blackbox attacker is necessary to satisfy the intent of the secure leakage-resilient system. Therefore, we evaluate space-hard block ciphers from a similar aspect to the strong incompressibility. Unfortunately, any space-hard ciphers never satisfy the strong incompressibility as discussed in Sect. 2. Therefore, we need to introduce a similar but different attack model to be sound for space-hard ciphers.

3.1 Attack Model

Figure 2 shows the high-level overview of our attack model. The hybrid code lifting consists of two phases: the 1st and 2nd phases suppose whitebox and blackbox attackers,

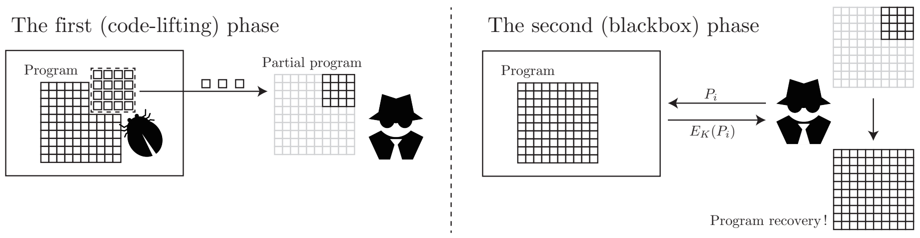

Figure 2: Hybrid code lifting.

respectively. Whether the attacker wins or not is finally determined in the 2nd phase. The attack is parameterized by $(\lambda, \tau_w, q, \tau_b)$, where λ and τ_w are parameters for the 1st phase, and q and τ_b are parameters for the 2nd phase.

Definition 5 (Hybrid code lifting with parameter $(\lambda, \tau_w, q, \tau_b)$)**.** Let $\tilde{E}_T : \mathbb{F}_2^n \to \mathbb{F}_2^n$ denote a program of a space-hard block cipher. We assume the attacker consists of two phases.

In the 1st (code-lifting) phase, we assume an attacker who hacks into the encryption program and can do everything against $\tilde{E}_T$. Specifically, the attacker can read the whole table entries of the space-hard block cipher and perform arbitrary computations on it. The goal is to generate and leak at most λ bits. Note that the time complexity of this phase is bounded by 2^{τ_w}. In practice, the attacker in this phase may be malware or the attacker who temporarily steals the device.

In the 2nd (blackbox) phase, we assume an attacker receiving the leakage generated in the 1st phase. The attacker no longer analyzes $\tilde{E}_T$ in the whitebox model but analyzes $\tilde{E}_T$ in the blackbox model. Specifically, the attacker queries plaintexts/ciphertexts to the encryption/decryption oracles up to q times. The goal is to recover the encryption program. In other words, the goal is to successfully construct an efficient program $\mathbf{Q} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $\mathbf{Q}(P) = \tilde{E}_T(P)$ for all $P \in \mathbb{F}_2^n$. Note that the time complexity of this phase is bounded by 2^{τ_b}.

Remark on Parameters. There are four parameters $(\lambda, \tau_w, q, \tau_b)$. The parameter λ is the size bound of the leakage. When $\lambda = size(T)$, the attacker always wins by leaking the table T itself. Thus, we consider $\lambda < size(T)$. Some space-hard block ciphers are expected to be secure even if a quarter of table entries is leaked [BI15, BIT16, KI21]. When we inherit this heuristic bound, $\lambda \leq size(T)/4$ would be required. The parameters τ_w and τ_b are the bounds of the time complexity in the 1st and 2nd phases, respectively. To avoid a trivial exhaustive search, $2^{\tau_w} + 2^{\tau_b} < 2^{\kappa}$ is necessary. The parameter q must be lower than 2^{τ_b}. Note that constructing an efficient program is non-trivial, even using the full codebook. Namely, when $\kappa > \tau_b > n$, $q = 2^n$ is a possible parameter. It is similar to the non-triviality of recovering the secret key of the block cipher with the full codebook.

3.2 Motivation of Hybrid Code Lifting

Hybrid code lifting is a hybrid of blackbox and whitebox models. We discuss why our attack model reflects the intention of space-hard block ciphers. We also show the difference from previous models.

Blackbox model after receiving leakage. We mainly focus on an attacker in the blackbox model, where the attacker receives the leakage generated by a whitebox attacker. Considering such an attacker is not new in whitebox cryptography. For example, Fouque et al. already supposed such an attacker in the strong incompressibility [FKKM16]. We discuss the necessity to consider such an attacker from the practical motivation of space-hard block ciphers.

The space hardness intends that ignoring slight leakage does not cause any security risk. Reflecting such an intention, the authors of [KI21] suggested the combined use with an anomaly detection system that detects huge leakage by monitoring process or outgoing packets. In other words, they suppose that detecting slight leakage is difficult, and it is not convincing that the attack model changes before and after slight leakage. If we do not need to consider a blackbox attacker after the leakage, the use case of the space-hard block cipher is limited, e.g., there are no blackbox attackers in the first place, or leakage is detectable despite the leakage size. The former is unlikely as a model for whitebox cryptography, and the latter loses the advantage of using space-hard block ciphers.

Hybrid Code Lifting and Longevity. Yoroi can update the table entries to enhance the security against the code lifting. Therefore, the authors suggested the following use case: users monitor the total amount of data traffic sent from the encryption device or the number of executions of the encryption program. If these numbers reach the threshold, which is equivalent to $(size(T)/64)$-bit leakage, users update their own table. If Yoroi is secure up to $(size(T)/64)$-bit leakage every table update, this use case is promising.

To provide protection in the use case above, the security claimed by the space hardness is too weak. Assuming that the cipher is vulnerable against the hybrid code lifting with parameters $(\lambda, \tau_w, q, \tau_b)$, where $\lambda < size(T)/64$, the use case above would be insecure because the implementation method does not matter for the blackbox attacker. Even if each table entry is updated, it does not contribute to security once we go to the blackbox phase.

Difference from Strong Incompressibility. As discussed in Sect. 2, space-hard block ciphers do not satisfy the strong incompressibility [FKKM16] because of a trivial attack. However, the attack model behind the strong incompressibility is crucial for the leakage-resilient system. Therefore, we revisit a similar security in space-hard block ciphers. Due to the similarity of the motivation, hybrid code lifting is similar to the strong incompressibility.

Considering the table entries of space-hard block ciphers are generated by block ciphers, only limiting leakage size is not sufficient. To avoid an exhaustive search by the leakage function f, we introduce a bound for the time complexity of the leakage function (whitebox attacker) instead of restriction by a min-entropy. We believe this revision is natural because code lifting is regarded as noise-free leakage by a whitebox attacker with limited running time in practice.

The other main difference is the attack goal. Unlike the strong incompressibility, the goal of the hybrid code lifting is the program recovery, which is the most powerful attack. It is interesting to discuss similar, stronger security (like indistinguishability) for space-hard block ciphers. However, since our primary focus is the attack, we do not discuss such security in our paper.

3.3 General Attack Idea for Program Recovery

In our hybrid scenario, the collaborative blackbox attacker exploits size-bounded leakage generated by the whitebox attacker. However, since the target cipher is a space-hard block cipher, it still contains large secret information after leakage. For example, the code (table) size of SPNbox-16 is $2^{16} \times 16$ bits. Assuming a quarter-size leakage, the remaining is $3/4 \times 2^{16} \times 16$ bits. It is significantly larger than a usual block cipher implementation loading 128 or 256-bit secret key. Thus, it is not easy for the collaborative blackbox attacker to recover the full program even after the leakage.

To recover the vast secret information (table entries), we present attack procedures that recover a part of (unleaked) table entries. This procedure highly depends on the target cipher: we use a truncated differential in Yoroi but a more straightforward guess-and-determine approach in SPNbox. In many cases, we recover table entries picked randomly. Namely, the procedure often reveals table entries known already. This is very similar to the so-called coupon collector's problem. The goal is to collect all table entries (coupons).

Theorem 1 (Coupon Collector's Problem)**.** *There are n coupons. We already have k_0 coupons. When one trial randomly opens 1 coupon, the expected number of trials to collect all n coupons is $nH_{n-k_0} \approx n(\ln(n-k_0)+\gamma)$, where H_n is the nth harmonic number, and γ is the Euler's constant, i.e., $\gamma \approx 0.577$.*

Theorem 1 is available to estimate the required number of procedures to collect all table entries. Often, there is a case that we cannot collect all coupons (table entries) by simply repeating the same attack procedure. Then, we want to estimate how many the limited trials can recover coupons (table entries). For that, we present the following corollary, a variant of the coupon collector's problem.

Corollary 1. *There are n coupons. We already have k_0 coupons. When one trial randomly opens one coupon, the expected total number of coupons collected by t trials is*

$$k + (t - n \times (H_{n-k_0} - H_{n-k})) \times \frac{n-k}{n},$$

where $k = \lfloor n - e^{\ln(n-k_0) - \frac{t}{n}} \rfloor$.

Proof. The first trial opens a new coupon with a probability of $\frac{n-k_0}{n}$. Therefore, $\frac{n}{n-k_0}$ trials are required to collect an additional coupon (k_0+1 coupons in total). The expected number of the remaining trials is $t - \frac{n}{n-k_0}$ after k_0+1 coupons are collected. After repeating the procedure up to collecting k coupons, the expected number of the remaining trials is

$$t - \frac{n}{n-k_0} - \frac{n}{n-k_0-1} - \cdots - \frac{n}{n-(k-1)} = t - n \times (H_{n-k_0} - H_{n-k})$$

after k coupons are collected.

When $t - n \times (H_{n-k_0} - H_{n-k}) = 0$, the expected number of the remaining trials is 0. Therefore,

$$\begin{aligned} t &= n \times (H_{n-k_0} - H_{n-k}) \approx n \times (\ln(n-k_0) - \ln(n-k)) \\ k &\approx n - e^{\ln(n-k_0) - \frac{t}{n}}, \end{aligned}$$

and we expect $k = \lfloor n - e^{\ln(n-k_0) - \frac{t}{n}} \rfloor$. The probability that we can collect the $(k+1)$th new coupon is estimated by

$$(t - n \times (H_{n-k_0} - H_{n-k})) \times \frac{n-k}{n}.$$

Thus, the expected number is

$$k + (t - n \times (H_{n-k_0} - H_{n-k})) \times \frac{n-k}{n}.$$

□

4 Preliminaries for Our Attacks against Yoroi

Before demonstrating the concrete attacks on Yoroi, we present the specification of Yoroi and some technical preliminaries used in our attacks shown in Sects. 5 and 6.

We provide some important theorems and corollaries. The coupon collector's problem and its variants are often used in our analysis. Moreover, we introduce a *perfect decomposition*, which is based on random graph theory, to estimate the attack complexity. We finally show a *canonical representation* of Yoroi. Our attack recovers the table of the canonical representation. We emphasize that the designers of Yoroi overlooked the existence of this canonical representation, and it causes the critical flaw of Yoroi.

4.1 Yoroi: Updatable Space-Hard Block Cipher

The updatable space-hard block cipher Yoroi was proposed at TCHES 2021 [KI21].

Partial table-entry leakage is not critical for space-hard block ciphers. It enables to mitigate the risk of code lifting by monitoring processes and/or outgoing packets. However, what about attackers leaking slight information from the table over a long time so as not to be found out by users? This is unlikely to be detectable. The designers of Yoroi tackled this problem and introduced a new unique functionality called *longevity*, where the table can be updated while maintaining the functionality of block ciphers.

4.1.1 Specification of Yoroi

Yoroi is a space-hard block cipher. There are two variants: Yoroi-16 and Yoroi-32 adopt key-dependent 16- and 32-bit tables (bijective functions), respectively, to build 128-bit block ciphers. Let $P \in (\mathbb{F}_2^{n_{in}})^\ell$ and $C \in (\mathbb{F}_2^{n_{in}})^\ell$ be a plaintext and ciphertext, respectively, and C is computed from P as $C = \mathcal{A} \circ \gamma^R \left(\bigcirc_{r=1}^{R-1} (\theta \circ \sigma^r \circ \gamma^r) \right) (P)$. We define the *Yoroi-core part* as $\gamma^R \left(\bigcirc_{r=1}^{R-1} (\theta \circ \sigma^r \circ \gamma^r) \right)$. Each component is defined as follows:

S-layer γ^r. The S-layer $\gamma^r : (\mathbb{F}_2^{n_{in}})^\ell \to (\mathbb{F}_2^{n_{in}})^\ell$ consists of ℓ key-dependent n_{in}-bit bijective functions. S_1 and S_3 are applied for the first and last rounds, respectively, and S_2 is applied for the rest of the rounds.

$$\gamma^r : (x_1, \ldots, x_\ell) \to (S_j(x_1), \ldots, S_j(x_\ell)),$$

where $j = 1$ for $r = 1$, $j = 3$ for $r = R$, and $j = 2$ for the rest of r.

Linear layer θ. The linear layer $\theta : (\mathbb{F}_2^{n_{in}})^\ell \to (\mathbb{F}_2^{n_{in}})^\ell$ consists of an MDS matrix $M \in (\mathbb{F}_2^t)^{\ell \times \ell}$. The ith n_{in}-bit output of S_j, denoted by x_i, is divided into the top m bits, x_i^L, and last t bits, x_i^R, i.e., $n_{in} = m + t$. The MDS matrix is multiplied with the last t bits of ℓ elements.

$$\theta : (x_1, \ldots, x_\ell) \to (x_1^L \| x_1'^R, \ldots, x_\ell^L \| x_\ell'^R),$$

where $x_i = x_i^L \| x_i^R$ and $(x_1'^R, \ldots, x_\ell'^R) = (x_1^R, \ldots, x_\ell^R) \times M$.

Affine layer σ^r. In the add-constant layer $\sigma^r : (\mathbb{F}_2^{n_{in}})^\ell \to (\mathbb{F}_2^{n_{in}})^\ell$, t-bit constants are added in the lsb t bits of each element of the state.

$$\sigma^r : (x_1, \ldots, x_\ell) \to (x_1^L, \ldots, x_\ell^L) \oplus (x_1^R + C_r, \ldots, x_\ell^R + C_r),$$

where $C_r = r$.

AES layer $\mathcal{A}$. Finally, the AES with a fixed key $K_{\mathcal{A}}$ is applied.

4.1.2 Updating Tables and Longevity

Yoroi has a unique feature called *longevity*, where updating the table is possible while maintaining the functionality. It intends to ensure security even if partial table entries are leaked every table update, and in total, it accepts massive leakage beyond the program size.

To achieve the longevity, Yoroi prepares a secure m-bit block cipher E and updates three tables as

$$T_1 \leftarrow (E\|I) \circ S_1, \qquad T_2 \leftarrow (E\|I) \circ S_2 \circ (E^{-1}\|I), \qquad T_3 \leftarrow S_3 \circ (E^{-1}\|I),$$

where $(E\|I)(x) = E(x^L)\|x^R$. Since θ and σ^i do not change the top m-bit values of each branch, the application of $(E\|I)$ after the rth S-layer is canceled out by applying of $(E^{-1}\|I)$ before the $(r+1)$th S-layer.

As shown in Definition 3, the whitebox compiler can output a fresh table by using different randomness. To the best of our knowledge, Yoroi is the only space-hard block cipher accepting randomness. Note that updating the table of Yoroi is possible with an old table, and the secret key is unnecessary.

4.1.3 Parameters and Claimed Security

Yoroi-16 uses $(n_{in}, m, t, \ell, R) = (16, 12, 4, 8, 8)$. Yoroi-32 uses $(n_{in}, m, t, \ell, R) = (32, 28, 4, 4, 16)$. The following are the designer's security claims.

- Secure against any attack in the blackbox model.
- Secure against key extraction in the whitebox model.
- $((3 \times 2^{n_{in}})/4, 128)$-space hardness against KSA[3].
- $((3 \times 2^{n_{in}})/64, 128)$-space hardness against KSA every table update.

For example, in Yoroi-16, encrypting a random plaintext is not possible even if (3×2^{14}) table entries are leaked by the KSA. Moreover, for longevity, encrypting a random plaintext is not possible even if (3×2^{10}) table entries are leaked by the KSA every table update. In other words, it accepts massive leakages, e.g., further beyond $3 \times 2^{n_{in}}$ as long as the table is updated with the proper interval.

4.2 Canonical Representation of Yoroi

We introduce the canonical representation of Yoroi, and it plays a crucial role in our attack. Yoroi is specified by three original tables S_1, S_2, and S_3, but there are implementations using T_1, T_2, and T_3 with maintaining its functionality. The canonical representation is uniquely determined by the original three tables and easily computed by three tables T_1, T_2, and T_3.

Figure 3 shows the representation. Three tables T_1, T_2, and T_3 are table entries used in the implementation $\tilde{E}_T$. The representation additionally involves m-bit permutation E_r, but these applications are always canceled out by applying D_r in the next round[4]. One important remark is that we can assign any $(E_r, D_r = E_r^{-1})$ such that whole encryption is perfectly preserved. Then, the tables used in the 1st and last rounds are $\tilde{T}_1 = (E_1\|I) \circ T_1$ and $\tilde{T}_R = T_3 \circ (D_{R-1}\|I)$, respectively. A table used in the rth round is $\tilde{T}_r = (E_r\|I) \circ T_2 \circ (D_{r-1}\|I)$ for $r \in \{2, 3, \ldots, R-1\}$.

[3]The designers of Yoroi claimed the security against the KSA/CSA. However, we found a trivial CSA to break the security (see Sect. A in detail). We contacted the authors, and they admitted they had overlooked this. In practice, the authors discuss the security against only the KSA in [KI21].

[4]Note that this property is originally introduced for the longevity of Yoroi. When a common m-bit block cipher E is used for all $r \in \{1, \ldots, R-1\}$, it is equivalent to the table update of Yoroi.

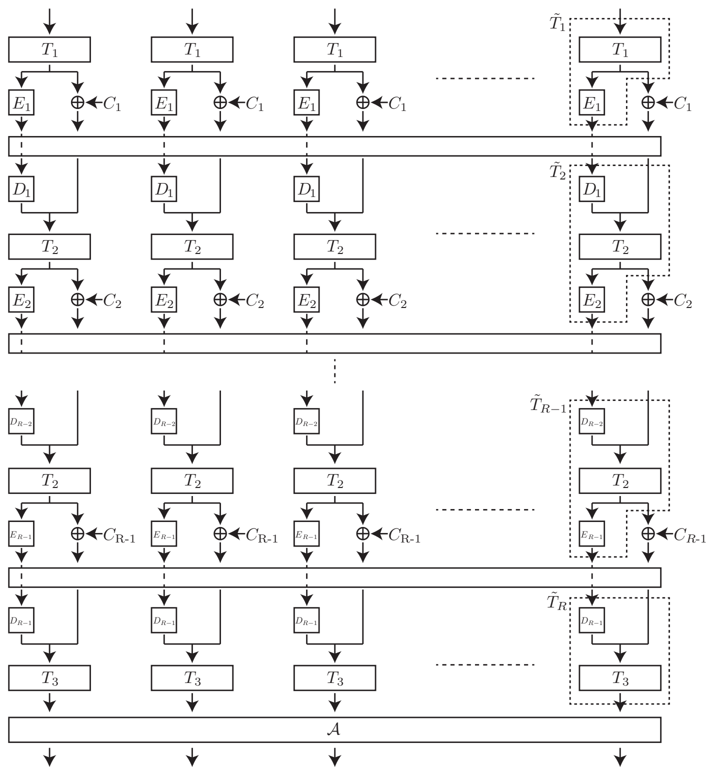

Figure 3: Canonical representation of encryption algorithm of YOROI.

We now exploit the freedom of E_r and construct $\tilde{T}_r$ for $r \in \{1, 2, \ldots, R-1\}$ satisfying the following property.

Property 1. When $\mathsf{lsb}_t(\tilde{T}_r(x)) = \mathsf{lsb}_t(\tilde{T}_r(x')) = 0$, $\mathsf{msb}_m(\tilde{T}_r(x)) < \mathsf{msb}_m(\tilde{T}_r(x'))$ holds for all $x < x'$.

Here, lsb_t denotes a t-bit string from the LSB, and msb_m denotes an m-bit string from the MSB. Given E_{r-1} and T, Algorithm 1 finds E_r satisfying Property 1 in $\tilde{T}_r$. Appendix B provides a small example to understand Property 1.

Proposition 1. *Tables $(\tilde{T}_1, \ldots, \tilde{T}_R)$ satisfying Property 1 are uniquely determined by S_1, S_2, and S_3.*

Proof. We first look at the 1st round, where $T_1 = (E\|I) \circ S_1$. Then,

$$\tilde{T}_1 = (E_1\|I) \circ T_1 = ((E_1 \circ E)\|I) \circ S_1.$$

We focus on the set $\mathbb{S}_1 := \{x | \mathsf{lsb}_t(T_1(x)) = 0\}$. Since $E_1 \circ E$ is not applied to the LSB t bits, $\mathbb{S}_1$ is determined by S_1 only. According to Property 1, $\mathbb{S}_1$ is sorted by ascending order. For any $x, x' \in \mathbb{S}_1$, we assign $(E_1 \circ E)$ such that $\tilde{T}_1(x) < \tilde{T}_1(x')$ for all $x < x'$. Such $(E_1 \circ E)$ is uniquely determined, and $\tilde{T}_1$ is uniquely determined.

We next look at the 2nd round, where $T_2 = (E\|I) \circ S_2 \circ (D\|I)$. Then,

$$\tilde{T}_2 = (E_2\|I) \circ T_2 \circ (D_1\|I) = ((E_2 \circ E)\|I) \circ S_2 \circ ((D \circ D_1)\|I).$$

Recall that $(E_1 \circ E)$ is uniquely determined by S_1. Since the inverse $(D \circ D_1)$ is also uniquely determined, $S_2 \circ ((D \circ D_1)\|I)$ is uniquely determined by S_1 and S_2. Similar to the 1st round, $(E_2 \circ E)$ is uniquely determined, and $\tilde{T}_2$ is uniquely determined.

Algorithm 1 Algorithm to determine E_r satisfying Property 1 in $\tilde{T}_r$.

```
Require: $T : \mathbb{F}_2^{n_{in}} \to \mathbb{F}_2^{n_{in}}$ , $(E_{r-1} : \mathbb{F}_2^m \to \mathbb{F}_2^m)$
Ensure: $E_r$ s.t. Property 1 holds in $\tilde{T}_r = (E_r \| I) \circ T \circ (E_{r-1}^{-1} \| I)$.
1: if $E_{r-1}$ is provided then
2:     $T \leftarrow T \circ (E_{r-1}^{-1} \| I)$
3: $cnt = 0$
4: for all $x \in \mathbb{F}_2^{n_{in}}$ do
5:     if $\mathsf{lsb}_t(T(x)) = 0$ then
6:         $E_r[\mathsf{msb}_m(T(x))] = cnt$
7:         $cnt \leftarrow cnt + 1$
8: return $E_r$
```

The iterative application shows $(E_r \circ E)$ is uniquely determined by S_1 and S_2 until $r = R - 1$. We finally look at the last round, where $T_3 = S_3 \circ (D \| I)$:

$$\tilde{T}_R = T_3 \circ (D_{R-1} \| I) = S_3 \circ ((D \circ D_{R-1}) \| I).$$

Since $(E_{R-1} \circ E)$ is uniquely determined by S_1 and S_2, the inverse $(D \circ D_{R-1})$ is also uniquely determined. Thus, $\tilde{T}_R$ is uniquely determined by S_1, S_2, and S_3. $\square$

Proposition 1 shows that Yoroi has a unique canonical representation determined by S_1, S_2, and S_3 only. Our attack recovers the canonical representation $\tilde{T}_r$ for all $r \in \{1, 2, \ldots, R\}$. In other words, we do not recover S_1, S_2, and S_3. Note that $\tilde{T}_r$ for all $r \in \{1, 2, \ldots, R\}$ (and the $K_{\mathcal{A}}$) are sufficient to encrypt (resp. decrypt) any plaintext (resp. ciphertext).

4.3 Perfect Decomposition

In our attack procedure, we sometimes divide the set $\mathbb{F}_2^n$ into 2^{n-m} subsets, where each subset contains 2^m elements, to recover the canonical representation $\tilde{T}_r$. Specifically, each subset contains all xs whose $\mathsf{lsb}_t(\tilde{T}_r(x))$ is the same. The attack procedure can detect whether two elements belong to the same subset or not probabilistically. For example, when the attack procedure detects that x_1 and x_2 belong to the same subset and x_1 and x_3 belong to the same subset, it derives that x_2 and x_3 belong to the same subset without having to detect it via the attack procedure. We regard this behavior as the *connectivity of the random graph*.

Theorem 2 (Connectivity of random graph [ER59])**.** *Let $G(n, p)$ be a random graph, where there are n vertices, and each edge is included in the graph with a probability of p. Then, the probability that the graph $G(n, p)$ is connected is estimated as $e^{-e^{-c}}$, where $c = p \times n - \ln n$.*

We consider 2^{n-m} random graphs. Each random graph contains 2^m vertices. It is assumed that one procedure detects that each edge is included in the graph with a probability of p. When we repeat the procedure s times, we simply assume that the probability that each edge is included in the graph is enhanced to $p \times s$. Then, the goal of the *perfect decomposition* is to construct 2^{n-m} disjoint connected random graphs. We call this problem (n, m, p)-perfect-decomposition.

Proposition 2 ((n, m, p)-perfect-decomposition.)**.** *The set $\mathbb{F}_2^n$ can be divided into 2^{n-m} subsets, where each subset contains 2^m elements. Let p denote the probability that one procedure detects $(x, x') \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)$ belong to the same subset, and we assume that the probability increases to $s \times p$ by s repetitions, where $s \ll 1/p$. Let p_{succ} be the probability* *that s procedures can divide the set $\mathbb{F}_2^n$ into 2^{n-m} subsets (with size 2^m). Then, the number of required repetitions to reach the probability of p_{succ} is*

$$s = p^{-1} \times 2^{-m} \times \left(\ln 2^m - \ln\left(-2^{m-n} \times \ln(p_{succ})\right)\right).$$

Proof. We first consider the probability that s procedures can find a subset whose number of elements is 2^m. Assuming each pair is independent, we can regard this problem as the connectivity problem of a random graph, where there are 2^m vertices and every possible edge occurs independently with probability $s \times p$. Due to Theorem 2,

$$e^{-e^{-c}},$$

where $c = p \times s \times 2^m - \ln 2^m$.

We need 2^{n-m} connected graph. Assuming they are independent, the probability is estimated as

$$(e^{-e^{-c}})^{2^{n-m}}.$$

The parameter c required to achieve the success probability p_{succ} is

$$\begin{aligned}
&(e^{-e^{-c}})^{2^{n-m}} = p_{succ},\\
&e^{-e^{-c}} = p_{succ}^{2^{m-n}},\\
&e^{-c} = -2^{m-n} \times \ln(p_{succ}),\\
&c = -\ln\left(-2^{m-n} \times \ln(p_{succ})\right).
\end{aligned}$$

Therefore,

$$\begin{aligned}
p \times s \times 2^m - \ln 2^m &= -\ln\left(-2^{m-n} \times \ln(p_{succ})\right),\\
s &= p^{-1} \times 2^{-m} \times \left(\ln 2^m - \ln\left(-2^{m-n} \times \ln(p_{succ})\right)\right).
\end{aligned}$$

□

We provide some examples.

Example 1 (($16, 12, 2^{-24}$)-perfect decomposition)**.** This is the parameter for Step 2-(a) to attack Yoroi-16 shown in Sect. 5.1. The number of procedures to achieve the success probability $p_{succ} = 0.5$ is estimated as

$$\begin{aligned}
s &= p^{-1} \times 2^{-m} \times \left(\ln 2^m - \ln\left(-2^{m-n} \times \ln(p_{succ})\right)\right)\\
&= 2^{24} \times 2^{-12} \times \left(\ln 2^{12} - \ln\left(2^{-4} \times \ln 2\right)\right) \approx 2^{15.52}.
\end{aligned}$$

Example 2 (($32, 28, 2^{-56}$)-perfect decomposition)**.** This is the parameter for Step 2-(a) to attack Yoroi-32 shown in Sect. 5.1. The number of procedures to achieve the success probability $p_{succ} = 0.5$ is estimated as

$$\begin{aligned}
s &= p^{-1} \times 2^{-m} \times \left(\ln 2^m - \ln\left(-2^{m-n} \times \ln(p_{succ})\right)\right)\\
&= 2^{56} \times 2^{-28} \times \left(\ln 2^{28} - \ln\left(2^{-4} \times \ln 2\right)\right) \approx 2^{32.45}.
\end{aligned}$$

5 Hybrid Code Lifting on Yoroi

We consider the security of the hybrid code lifting against Yoroi. The following is the notation used in our attack.

- n: block length, i.e., $n = 128$.

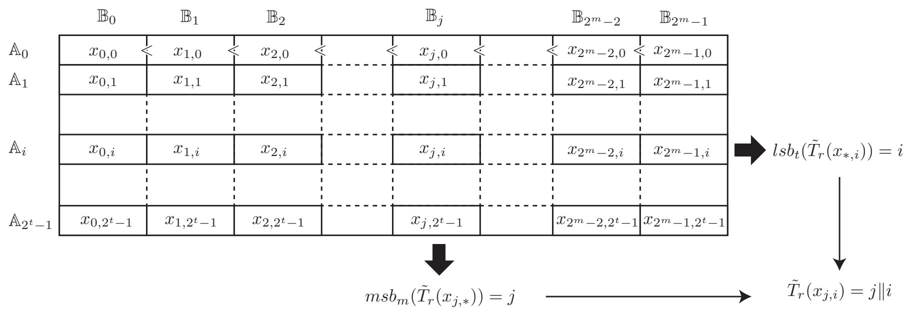

Figure 4: Structure and strategy to recover $\tilde{T}_r$.

- n_{in}: bit size of table, i.e., $n_{in} = 16$ for Yoroi-16, and $n_{in} = 32$ for Yoroi-32.
- ℓ: number of tables in every round, i.e., $\ell = 8$ and $\ell = 4$ for Yoroi-16 and Yoroi-32, respectively. Note that $n = n_{in} \times \ell$.
- m: bit size, where MDS and constant XORing are not applied in θ and σ^i, i.e., $m = 12$ and $m = 28$ for Yoroi-16 and Yoroi-32, respectively.
- t: bit size, where MDS and constant XORing are applied in θ and σ^i, i.e., $t = 4$. Note that $n_{in} = m + t$.
- $P = (P[1], P[2], \ldots, P[\ell]) \in (\mathbb{F}_2^{n_{in}})^\ell$: plaintext.
- $C = (C[1], C[2], \ldots, C[\ell]) \in (\mathbb{F}_2^{n_{in}})^\ell$: ciphertext.
- $\mathsf{lsb}_t(X)$: t-bit string from the LSB of $X \in \mathbb{F}_2^*$.
- $\mathsf{msb}_m(X)$: m-bit string from the MSB of $X \in \mathbb{F}_2^*$.

We introduce notations used in the analysis based on the canonical representation. We use the following notations to recover the table $\tilde{T}_r$ of the canonical representation.

- $\rho \in (\mathbb{F}_2^m)^{2^t}$: 2^t-dimensional vector whose elements take a value over $\mathbb{F}_2^m$.
- $\mathbb{A}_i := \{x \in \mathbb{F}_2^{n_{in}} | \mathsf{lsb}_t(\tilde{T}_r(x)) = i\}$. Since $\tilde{T}_r$ is a permutation, the number of elements of $\mathbb{A}_i$ is 2^m for any $i \in \mathbb{F}_2^t$. We sometimes use $\mathbb{A}_{\rho_i}$ when ρ has not been recovered yet. Then, when $x, x' \in \mathbb{A}_{\rho_i}$, $\mathsf{lsb}_t(\tilde{T}_r(x)) = \mathsf{lsb}_t(\tilde{T}_r(x'))$.
- $\eta \in (\mathbb{F}_2^t)^{2^m}$: 2^m-dimensional vector whose elements take a value over $\mathbb{F}_2^t$.
- $\mathbb{B}_j := \{x \in \mathbb{F}_2^{n_{in}} | \mathsf{msb}_m(\tilde{T}_r(x)) = j\}$. Since $\tilde{T}_r$ is permutation, the number of elements of $\mathbb{B}_j$ is 2^t for any $j \in \mathbb{F}_2^m$. We sometimes use $\mathbb{B}_{\eta_j}$ when η has not been recovered yet. Then, when $x, x' \in \mathbb{B}_{\eta_j}$, $\mathsf{msb}_m(\tilde{T}_r(x)) = \mathsf{msb}_m(\tilde{T}_r(x'))$.
- $x_{j,i} \in \mathbb{F}_2^{n_{in}}$: input of $\tilde{T}_r$ such that $\tilde{T}_r(x_{j,i}) = j\|i$.

Figure 4 summarizes these notations. We show the following Lemma.

Lemma 1. *The table $\tilde{T}_r$ of the canonical representation can be recovered by $\mathbb{A}_i$ and $\mathbb{B}_{\eta_j}$ for all i and j.*

Proof. In $\mathbb{A}_0$, inputs satisfying $\mathsf{lsb}_t(\tilde{T}_r(x)) = 0$ are stored. For any $x, x' \in \mathbb{A}_0$ and $x < x'$, $\mathsf{msb}_m(\tilde{T}_r(x)) < \mathsf{msb}_m(\tilde{T}_r(x'))$ holds due to Property 1 in the canonical representation. Let $x_{j',0}$ be the j'th element of the (ascending) sorted $\mathbb{A}_0$. Then, when $x_{j',0} \in \mathbb{B}_{\eta_j}$, we obtain $\eta_j = j'$. We now obtain $\mathbb{A}_i$ and $\mathbb{B}_j$ for all i and j. Thus, when $x \in \mathbb{A}_i$ and $x \in \mathbb{B}_j$, $\tilde{T}_r(x) = j\|i$. □

On the canonical representation, $\mathbb{A}_i$ and $\mathbb{B}_{\eta_j}$ for all i and j are enough to recover $\tilde{T}_r$. We do not need to know η because it can be complemented from $\mathbb{A}_0$.

5.1 Attack Procedure

We show a detailed attack procedure. For the sake of simplicity, we first show an attack whose leakage size is $128 + (R-1) \times 2^t \times n_{in}$ bits. We later reduce the leakage size to $128 + (R-1) \times 6 \times n_{in}$ bits with a negligible impact on the complexity.

5.1.1 The 1st (Code Lifting) Phase

The 1st phase is the code lifting by a whitebox attacker. The attacker generates a leakage, which is useful in the 2nd phase.

The attacker first extracts the AES key $K_{\mathcal{A}}$ from the encryption program. It might not be easy assuming there is a secure whitebox implementation of AES. However, in practice, realizing such a secure implementation is difficult [BHMT16, BBIJ17, BU18]. Therefore, we assume the AES key can be extractable. Note that the designers of Yoroi introduced the AES layer for security in the blackbox model and do not expect the layer to mitigate code lifting [KI21].

The knowledge of $K_{\mathcal{A}}$ reveals the Yoroi-core part. We notice the Yoroi-core part only is an insecure block cipher because it has a non-trivial truncated differential distinguisher (see Fig. 5 and the 2nd phase in detail). Despite such a weakness, recovering all table entries is non-trivial because we cannot guess the correct secret table due to huge search space, e.g., $2^{16}! \approx 2^{954036}$. For the practical attack, we leak small fragments about the table.

We focus on the canonical representation of Yoroi. By applying Algorithm 1 from $r = 1$ to $r = (R-1)$ iteratively, the whitebox attacker generates $(\tilde{T}_1, \ldots, \tilde{T}_R)$ of the canonical representation. We finally leak all elements in $\mathbb{B}_0$, i.e., $x_{0,i}$ satisfying $\tilde{T}_r(x_{0,i}) = (0\|i)$ for all $i \in \mathbb{F}_2^t$, of $\tilde{T}_r$ from $r = 1$ to $R-1$. Note that we do not leak $\mathbb{B}_0$ of $\tilde{T}_R$. Please refer to Sect. 5.1.2 on how to exploit the leakage.

We finally summarize the attack complexity in the code lifting phase. Generating the table of the canonical representation (and retrieving $x_{0,i}$ satisfying $\tilde{T}_r(x_{0,i}) = (0\|i)$) is possible with the complexity of $2^{n_{in}}$ for each r. Therefore, the complexity is $(R-1) \times 2^{n_{in}}$, which is about $2^{18.8}$ and $2^{35.9}$ for Yoroi-16 and Yoroi-32, respectively. The leakage size is $128 + (R-1) \times 2^t \times n_{in}$ bits, which are 1920 and 7808 bits in Yoroi-16 and Yoroi-32, respectively.

5.1.2 The 2nd (Blackbox) Phase

The 2nd phase is the differential cryptanalysis by a blackbox attacker receiving the leakage generated by the 1st phase. We ignore the last AES layer and regard the output of the Yoroi-core part as ciphertexts because the leakage includes $K_{\mathcal{A}}$.

We first show that the Yoroi-core part is easily distinguished from ideal block ciphers. Recall the linear layer θ, where the MDS matrix is only applied to the last t bits of each output of $\tilde{T}_r$. Therefore, θ does not diffuse active branches when there is no difference in the last t bits. This causes the truncated differential shown in Fig. 5, where Δ denotes any non-zero difference. The probability of satisfying this truncated differential is $2^{-t(R-1)}$. Thus, a distinguishing attack is easy. On the other hand, the attack goal is the program recovery[5]. We use the following procedure.

[5]In practice, distinguishing attacks are less interesting for the goal of the hybrid code lifting. This is because a whitebox attacker can leak any plaintext-ciphertext pairs. Therefore, a collaborative blackbox attacker can distinguish by querying the leaked plaintexts. Of course, as we demonstrated, a non-trivial distinguisher would allow the blackbox attacker to recover all table entries. Therefore, the designer should eliminate such a distinguisher.

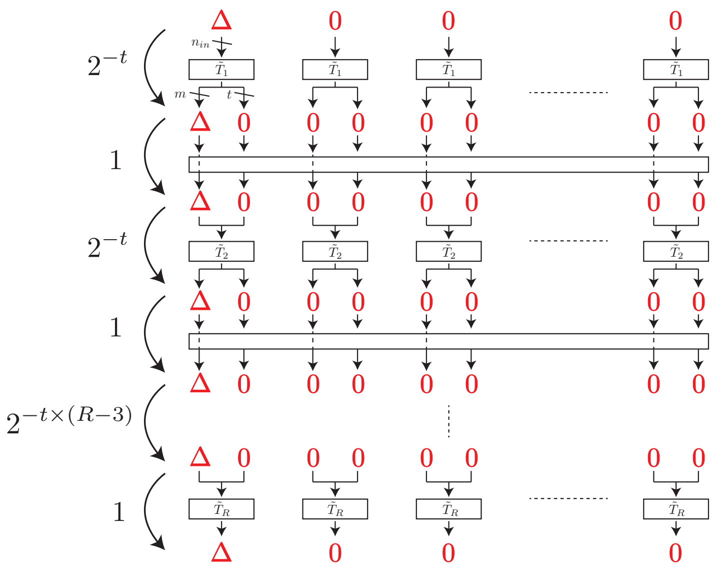

Figure 5: Truncated differential of YOROI-core part.

Step 2a. Following the truncated differential shown in Fig. 5, the blackbox attacker runs the following procedure.

1. Prepare a set with $2^{n_{in}}$ plaintexts as following: For the 1st element, we takes all values over $\mathbb{F}_2^{n_{in}}$. For the others, we take randomly-chosen fixed values. Then, we have $2^{n_{in}}$ plaintexts and obtain corresponding ciphertexts by using the encryption oracle.
2. List pair $(P[1], P'[1])$ satisfying the partial collision $\mathsf{lsb}_{n_{in}(\ell-1)}(C \oplus C') = 0$.

Every procedure requires $2^{n_{in}}$ times, data, and memory complexities, and all $(= \binom{2^{n_{in}}}{2} \approx 2^{2n_{in}-1})$ pairs are checked at the same time. Assuming that $(P[1], P'[1])$ belong to the same subset $\mathbb{A}_{\rho_i}$, the probability that such pairs can be detected by one procedure is $2^{-t\times(R-2)}$. We estimate the required number of procedures to divide $\mathbb{F}_2^{n_{in}}$ into $\mathbb{A}_{\rho_i}$. As shown in Proposition 2, it is regarded as $(n_{in}, m, 2^{-t\times(R-2)})$-perfect-decomposition. Let s be the required number of procedures to divide $\mathbb{F}_2^{n_{in}}$ into $\mathbb{A}_{\rho_i}$ with the success probability of 50%. Then, the total complexity is estimated as follows.

In YOROI-16, $s = 2^{15.52}$. Thus, the total complexity is $2^{15.52} \times 2^{16} \approx 2^{31.52}$.

In YOROI-32, $s = 2^{32.45}$. Thus, the total complexity is $2^{32.45} \times 2^{33} \approx 2^{64.45}$.

Step 2b. Step 2a does not recover ρ_i. We refer to the leakage and retrieve the data about ρ_i. The leakage includes $x_{0,i'}$ satisfying $\mathsf{lsb}_t(\tilde{T}_1(x_{0,i'})) = (0\|i')$ for all $i' \in \mathbb{F}_2^t$. Therefore, we check if the subset $\mathbb{A}_{\rho_i}$ includes $x_{0,i'}$ or not. When $x_{0,i'} \in \mathbb{A}_{\rho_i}$, $\rho_i = i'$.

Step 2c. We next recover $\mathbb{B}_{\eta_j}$. The truncated differential shown in Fig. 5 is not helpful. We use the modified truncated differentials shown in Fig. 6, where α is a m-bit arbitrary difference, and β is a t-bit non-zero difference chosen by the blackbox attacker. Then, difference $\zeta_l \in \mathbb{F}_2^t$ is a non-zero difference determined as $(\zeta_1, \zeta_2, \ldots, \zeta_\ell) = (\beta, 0, 0, \ldots, 0) \cdot M^{-1}$.

The subset $\mathbb{A}_i$ is already recovered in Step 2a and Step 2b. Moreover, the attacker knows $x_{0,i}$ that satisfies $\tilde{T}_1(x_{0,i}) = (0\|i)$ by using the leakage. By using them, the attacker runs the following procedure.

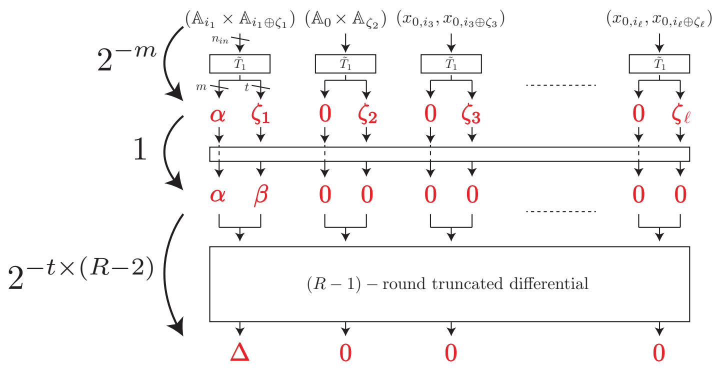
$(\mathbb{A}_{i_1} \times \mathbb{A}_{i_1 \oplus \zeta_1})$
$(\mathbb{A}_0 \times \mathbb{A}_{\zeta_2})$
$(x_{0,i_3}, x_{0,i_3 \oplus \zeta_3})$
$(x_{0,i_\ell}, x_{0,i_\ell \oplus \zeta_\ell})$
2^{-m}
n_{in}
$\tilde{T}_1$
m
t
α ζ_1 0 ζ_2 0 ζ_3 0 ζ_ℓ
1
α β 0 0 0 0 0 0
$2^{-t \times (R-2)}$
$(R-1)$ – round truncated differential
Δ 0 0 0

Figure 6: Another truncated differential of YOROI-core part.

1. Compute $(\zeta_1, \zeta_2, \ldots, \zeta_\ell)$ from a non-zero $\beta \in \mathbb{F}_2^t$.
2. Prepare a set with 2^{2m} plaintexts as follows: For the 1st element, we use a subset $\mathbb{A}_{i_1}$, where i_1 denotes a randomly chosen index. For the 2nd element, we use $\mathbb{A}_0$. For other lth elements, we randomly choose one text x_{0,i_l} stored in the leakage, i.e., $\tilde{T}_1(x_{0,i_l}) = (0\|i_l)$. Then, we have 2^{2m} plaintexts and obtain corresponding ciphertexts by using the encryption oracle.
3. Prepare another set with 2^{2m} plaintexts as follows: For the 1st element, we use $\mathbb{A}_{i_1 \oplus \zeta_1}$. For the 2nd element, we use $\mathbb{A}_{0 \oplus \zeta_2}$. For the lth elements, we choose a text $x_{0,i_l \oplus \zeta_l}$ from the leakage, where $\tilde{T}_1(x_{0,i_l}) \oplus \tilde{T}_1(x_{0,i_l \oplus \zeta_l}) = (0\|\zeta_l)$ holds. Then, we have 2^{2m} plaintexts and obtain corresponding ciphertexts by using the encryption oracle.
4. Search for pairs $(P[2], P'[2])$ satisfying the partial collision, $\mathsf{lsb}_{n_{in}(\ell-1)}(C \oplus C') = 0$.

Each procedure requires 2^{2m+1} time, data, and memory complexities. It checks all pairs $(P[2], P'[2]) \in (\mathbb{A}_0 \times \mathbb{A}_{\zeta_2})$ for a fixed $\zeta_2 (\neq 0)$. Assume that $P[2]$ and $P'[2]$ belong to the same subset $\mathbb{B}_{\eta_j}$, i.e., $\mathsf{msb}_m(\tilde{T}_1(P[2])) = \mathsf{msb}_m(\tilde{T}_1(P'[2]))$. When a pair $(P[2], P'[2])$ is checked, the probability that such a pair is detected in this procedure is $2^{-t(R-2)}$. One procedure checks this pair 2^{2m} times by trying out the value in $P[1]$. Since $2^{2m} \times 2^{-t(R-2)} = 1$ for both YOROI-16 and YOROI-32, the probability that the pair is not detected is estimated as e^{-1} due to the Poisson distribution. After d repetitions, the probability that the pair is not detected is estimated as e^{-d}.

We try out $2^t - 1$ possible β. Then, it eventually checks all pairs $(P[2], P'[2]) \in (\mathbb{A}_0 \times \mathbb{A}_{\zeta_2})$ for all $\zeta_2 (\neq 0)$. To detect all pairs, i.e., $(2^t - 1) \times 2^m \approx 2^{n_{in}}$ pairs, $d = \lceil \ln 2^{n_{in}} \rceil$ repetitions are required, which are 12 and 23 in YOROI-16 and YOROI-32, respectively. Then, the total complexity is estimated as follows:

In YOROI-16, the total complexity is $12 \times 15 \times 2^{25} \approx 2^{32.49}$.

In YOROI-32, the total complexity is $23 \times 15 \times 2^{57} \approx 2^{64.43}$.

Step 2d. The attacker has already recovered $\mathbb{A}_i$ for all $i \in \mathbb{F}_2^t$ and $\mathbb{B}_{\eta_j}$ for all $j \in \mathbb{F}_2^m$. As shown in Lemma 1, they recover η_j and $\tilde{T}_1$ thanks to the canonical representation.

Step 2e. The attacker can remove the first round because $\tilde{T}_1$ is already recovered. The attacker can use the same procedure above to recover $\tilde{T}_2, \tilde{T}_3, \ldots, \tilde{T}_{R-1}$. The probability of the truncated differential increases compared with the one to recover $\tilde{T}_1$. Thus, the attack complexity decreases. We regard these complexities as negligible.

Step 2f. The attacker already has $(\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_{R-1})$ and finally recovers $\tilde{T}_R$. It is easy to recover the table entry of $\tilde{T}_R$ by checking the consistency with the ciphertext given by the encryption oracle.

5.1.3 Reducing Leakage Size

We show an additional technique that reduces the leakage size by more than half with a negligible complexity increase.

The idea is to leak specific partial elements of $\mathbb{B}_0$ instead of all elements. Specifically, we assume to leak the following six elements

$$x_{0,0}, x_{0,1}, x_{0,2}, x_{0,4}, x_{0,8}, x_{0,15},$$

where $\tilde{T}_r(x_{0,i}) = (0\|i)$. This technique enables us to reduce the leakage size to $128 + (R-1) \times 6 \times n_{in}$ bits, which are 800 and 3008 bits of Yoroi-16 and Yoroi-32, respectively.

When we use less leakage, we cannot retrieve ten elements of ρ_i from the leakage in Step 2b. In other words, we cannot determine i such that $\rho_i \in \{3, 5, 6, 7, 9, 10, 11, 12, 13, 14\}$. Nevertheless, these ten unknown elements are efficiently recovered by tweaking the procedure of Step 2c. Referring to Fig. 6, we consider constructing a pair satisfying the truncated differential using only six limited elements. First, when $i_l, i_l \oplus \zeta_l \in \{0, 1, 2, 4, 8, 15\}$, there is a pair $(x_{0,i_l}, x_{0,i_l \oplus \zeta_l})$ for any $\zeta_l \in \mathbb{F}_2^t (= \mathbb{F}_2^4)$. For example, when $\zeta_l = 7$, a pair $(x_{0,8}, x_{0,15})$ is available, and $\tilde{T}_r(x_{0,8}) \oplus \tilde{T}_r(x_{0,15}) = (0\|7)$. Similarly, there is an i_1, where both $\mathbb{A}_{i_1}$ and $\mathbb{A}_{i_1 \oplus \zeta_1}$ are known for any $\zeta_1 \in \mathbb{F}_2^t$. Finally, $\mathbb{A}_0$ is known, and we try out all unknown $\mathbb{A}_{\rho_i}$ for $\mathbb{A}_{\zeta_2}$, where $\zeta_2 \in \{3, 5, 6, 7, 9, 10, 11, 12, 13, 14\}$. Then, we detect $\rho_i = \zeta_2$ by observing one truncated differential.

The procedure above is more efficient than Step 2c because we do not need to recover $\mathbb{B}_{\eta_j}$ completely. Only observing one truncated differential is enough to detect $\rho_i = \zeta_2$ because the probability that truncated differential holds is very low, i.e., $2^{-n_{in} \times (\ell - 1)}$ when wrong ρ_i is used. Specifically, $2^{m+m/2+1}$ time, data, and, memory complexities are required to recover one unknown ρ_i, and in total $10 \times 2^{m+m/2+1}$. The complexity is negligible compared to Step 2a and Step 2c.

5.1.4 Summary of Results

Our results are summarized in Table 1. Our attack uses 800-bit leakage and 3008-bit leakage for Yoroi-16 and Yoroi-32, respectively. These leakage sizes are significantly less than $size(T)/64$.

The time complexity of the code-lifting phase is about $2^{n_{in}}$. Thus, the complexity of the 1st phase is practical for both Yoroi-16 and Yoroi-32.

The time complexity of the blackbox phase is about 2^{33} to attack Yoroi-16. Therefore, the time complexities of both 1st and 2nd phases are practical. On the other hand, the time complexity is about $2^{65.5}$ to attack Yoroi-32. While we cannot say this is practical for a single PC, the complexity of 2^{64} is usually not recommended.

We finally remark that this attack is outside of the security claim of Yoroi. On the other hand, we need to stress that the claimed security of Yoroi is too weak to claim that the use case of Yoroi can be secure. Note that we break Yoroi's claimed security for longevity in Sect. 6.

5.1.5 Experimental Verification

We implemented our attack against Yoroi-16 for the verification. We tried to recover $\tilde{T}_1$ 10 times, and Table 3 shows the experimental results for the attack on Yoroi-16. The theoretical estimation of s in Step 2a is $2^{15.52}$, and the corresponding average value in our experiments is $48232.8 \approx 2^{15.56}$. Therefore, our theoretical estimation works well. The

Table 3: Experimental results for the attack on YOROI-16.

trial	1	2	3	4	5	6	7	8	9	10	average
s in Step 2a	46332	55660	48532	49984	44176	41712	53108	46288	47124	49412	48232.8
d in Step 2c	12	14	10	13	13	15	12	14	12	11	12.6

theoretical estimation of d in Step 2c is 12, and the corresponding average value in our experiments is 12.6. Again, our theoretical estimation works well.

5.2 Hybrid Code Lifting using More Leakage

Drawing the trade-off between the leakage size and attack complexity is interesting. We now present another simple attack using more leakage (but the size is still smaller than $\frac{3\times 2^{n_{in}}}{4}$ table entries).

The simple attack no longer exploits the specific property of YOROI, such as the canonical representation. Assuming that s_1 and s_2 table entries are leaked from T_1 and T_2, respectively, the total leakage is $s_1 + s_2 < \frac{3\times 2^{n_{in}}}{4}$. Note that we assume the AES key $K_{\mathcal{A}}$ is also leaked. Then, we can prepare s_1^ℓ chosen plaintexts, where the first round can be computed. The probability that these chosen plaintexts are successfully encrypted by one round before the last round is $(\frac{s_2}{2^{n_{in}}})^{\ell\times(R-2)}$. Ciphertexts corresponding to the chosen plaintexts are obtained by asking the encryption oracle, and the outputs of the YOROI-core part can be computed. As a result, we have $s_1^\ell \times (\frac{s_2}{2^{n_{in}}})^{\ell\times(R-2)} \times \ell$ overlapping table entries about S_3, and if the the following inequality

$$s_1^\ell \times \left(\frac{s_2}{2^{n_{in}}}\right)^{\ell\times(R-2)} \times \ell \geq 2^{n_{in}} H_{2^{n_{in}}} \tag{1}$$

holds, we can recover the full table entries of S_3 because of Theorem 1.

Equation (1) holds when $s_1 = 22192$ on YOROI-32. Then, we query $22192^4 \approx 2^{57.75}$ chosen plaintexts. The data and time complexities to recover S_3 are $2^{57.75}$, which is faster than the attack shown in Sect. 5.1. Of course, the required leakage size is much larger, i.e., $32 \times \frac{3\times 2^{32}}{4}$bits $\approx$ 12GB.

Unlike YOROI-32, this simple attack does not draw an interesting trade-off on YOROI-16. The same analysis holds when $s_1 = 24$, but the required data and time complexities are $2^{36.68}$, which is much higher than the attack shown in Sect. 5.1.

6 Attacks against Longevity of Yoroi

In this section, we provide another attack, where the target is the longevity of YOROI. An attacker collects leakage every table update and tries to recover the program without querying oracle. Figure 7 shows the overview of the attack. If the table leakage is limited by the KSA, the designers of YOROI claimed the $((3\times 2^{n_{in}})/64, 128)$-space hardness.

The canonical representation is useful in this attack too. We show three different attack models. The first one is the strongest, and it is unlikely to resist such attacks in general. The last one is the weakest, which corresponds to the known-space attack. Note that the last attack breaks one of the designers' security claims about longevity.

6.1 Attack in Whitebox Model with Nonvolatile Memory

Let us assume a whitebox model, where a nonvolatile memory is available. An attacker can memorize the old program on the encryption device and leak a little bit of the old program with each table update. Considering such a model, bounding the leakage size of

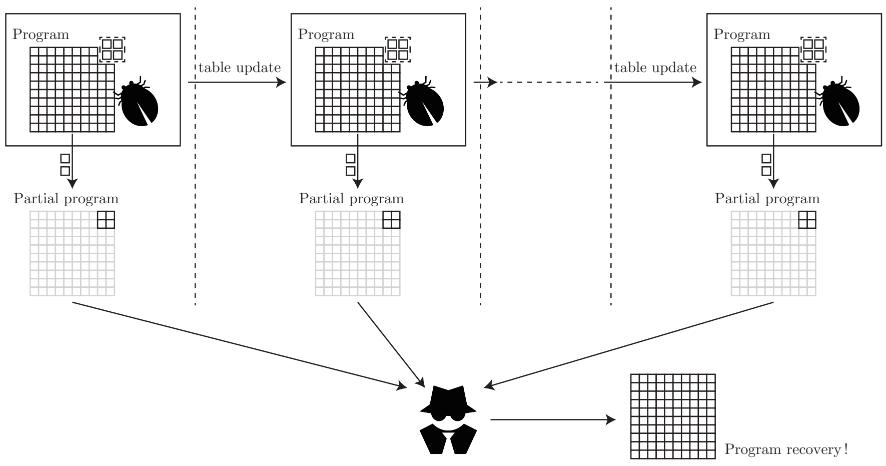

Figure 7: Attacks against longevity.

every table update no longer restricts the ability of the whitebox attacker. Therefore, such attacks are unavoidable.

6.2 Attack in Whitebox Model without Nonvolatile Memory

For more constructive discussion, we next assume a weakened whitebox model, where nonvolatile memory is not available, and all knowledge about the old program is lost after the table update. We believe that such a weakened model is still reasonable enough for the discussion of the whitebox model.

Unfortunately, Yoroi is insecure against such a weakened model. Recall that Yoroi has the canonical representation. The attacker can reconstruct the canonical representation from every updated table by using Algorithm 1. Therefore, the attacker leaks the partial data of $(\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_R)$ every table update. Specifically, it leaks $3 \times 2^{n_{in}}/64$ table entries every table update. We can collect $(\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_R)$ by receiving leakage with

$$\left\lceil \frac{R \times 2^{n_{in}}}{3 \times 2^{n_{in}}/64} \right\rceil = \lceil 64R/3 \rceil$$

table updates, which are 171 and 342 in Yoroi-16 and Yoroi-32, respectively.

6.3 Known-Space Attack: Breaking Security Claim

We finally present an attack using the known-space attack (KSA) model only. The authors claim the space hardness when at most $2^{n_{in}}/64 = 2^{n_{in}-6}$ table entries are extracted by the KSA every table update. We break this security claim. Specifically, we reconstruct the encryption program by using the leakage generated by the KSA model only. Since the attacker can encrypt any plaintext with the reconstructed program, a space-hardness is lost.

In the KSA, the attacker cannot modify table entries. The attacker extracts only partial data about T_1, T_2, and T_3 but cannot choose extracted entries. To avoid the confusion, we use $T_1^{(\mathscr{T})}$, $T_2^{(\mathscr{T})}$, and $T_3^{(\mathscr{T})}$ to represent three tables at time $\mathscr{T}$. When a table is updated, three tables are updated to $T_1^{(\mathscr{T}+1)}$, $T_2^{(\mathscr{T}+1)}$, and $T_3^{(\mathscr{T}+1)}$. The attack goal is to recover the canonical representation, i.e., we recover $\tilde{T}_1, \tilde{T}_2, \ldots \tilde{T}_R$. Note that we assume the AES key $K_{\mathcal{A}}$ is known or extractable. Otherwise, the discussion is meaningless because it is secure regardless of the Yoroi-core part.

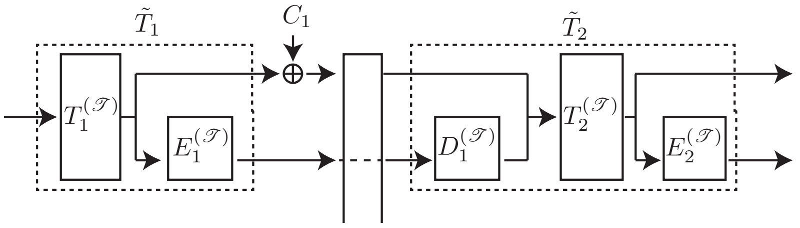

Figure 8: Relationship between $T_1^{(\mathscr{T})}$, $\tilde{T}_1$, $T_2^{(\mathscr{T})}$, and $\tilde{T}_2$.

6.3.1 Recovery of $\tilde{T}_1$

The first goal is to recover $\tilde{T}_1$, where $\tilde{T}_1 = (E_1^{(\mathscr{T})} \| I) \circ T_1^{(\mathscr{T})}$. The attacker extracts $\alpha \approx 2^{n_{in}-6}$ table entries

$$((x_1, T_1^{(\mathscr{T})}(x_1)), (x_2, T_1^{(\mathscr{T})}(x_2)), \dots, (x_\alpha, T_1^{(\mathscr{T})}(x_\alpha)))$$

and checks $\mathsf{lsb}_t(T_1^{(\mathscr{T})}(x_i))$. Since the mask $E_1^{(\mathscr{T})}$ is not applied to the last t bits,

$$\mathsf{lsb}_t(\tilde{T}_1(x_i)) = \mathsf{lsb}_t(T_1^{(\mathscr{T})}(x_i)).$$

Therefore, we can easily recover $\mathbb{A}_i := \{x \in \mathbb{F}_2^{n_{in}} | \mathsf{lsb}_t(\tilde{T}_1(x)) = i\}$ by observing $T_1^{(\mathscr{T})}(x_i)$ for all x_i.

We next recover $\mathbb{B}_{\eta_j}$. Note that the mask $E_1^{(\mathscr{T})}$ is applied, and this mask changes every $\mathscr{T}$. On the other hand, collision pairs are preserved independently of $E_1^{(\mathscr{T})}$. Therefore, we search for (x, x') satisfying

$$\mathsf{msb}_m(\tilde{T}_1(x) \oplus \tilde{T}_1(x')) = 0 \Leftrightarrow \mathsf{msb}_m(T_1^{(\mathscr{T})}(x) \oplus T_1^{(\mathscr{T})}(x')) = 0.$$

About $\binom{\alpha}{2} \times 2^{-m} \approx 2^{2n_{in}-13-m}$ pairs can be observed every $\mathscr{T}$. By collecting leakages with different $\mathscr{T}$, we recover $\mathbb{B}_{\eta_j}$ for all $j \in \mathbb{F}_2^m$.

In the canonical representation, $\tilde{T}_1$ can be recovered by $\mathbb{A}_i$ and $\mathbb{B}_{\eta_j}$ for all i and j (see Lemma 1). We omit to estimate the complexity because this step is more negligible than the following steps.

6.3.2 Recovery of $\tilde{T}_2$

The second goal is to recover $\tilde{T}_2$, where $\tilde{T}_2 = (E_2^{(\mathscr{T})} \| I) \circ T_2^{(\mathscr{T})} \circ (D_1^{(\mathscr{T})} \| I)$. Unlike the first round, we need to remove the pre-mask $(D_1^{(\mathscr{T})} \| I)$. Otherwise, we cannot compute the input of $\tilde{T}_2$ from the leaked $T_2^{(\mathscr{T})}$.

Figure 8 shows the relationship among $T_1^{(\mathscr{T})}$, $\tilde{T}_1$, $T_2^{(\mathscr{T})}$, and $\tilde{T}_2$. Some table entries of $E_1^{(\mathscr{T})}$ are constructed as

$$\mathsf{msb}_m(T_1^{(\mathscr{T})}(x_i)) \xrightarrow{E_1^{(\mathscr{T})}} \mathsf{msb}_m(\tilde{T}_1(x_i)).$$

We now have $\alpha \approx 2^{n_{in}-6}$ table entries of $T_1^{(\mathscr{T})}$ and $\tilde{T}_1$ is already recovered. In other words, we collect some table entries of $E_1^{(\mathscr{T})}$ by α trials, and Corollary 1 is available to estimate the number of recovered table entries.

We applied this analysis to Yoroi-16. We use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 2^{10})$. Then, the expected number of recovered table entries is 906.143. In other words, we know 906.143 table entries about $E_1^{(\mathscr{T})}$ (and also $D_1^{(\mathscr{T})}$).

We have $\beta \approx 2^{10}$ table entries of $T_2^{(\mathscr{T})}$. Therefore, we have $\beta \times \frac{906.143}{2^{12}} \approx 226.536$ table entries about $T_2^{(\mathscr{T})} \circ (D_1^{(\mathscr{T})} \| I)$.

The attack procedure to recover $\tilde{T}_2$ is the same as $\tilde{T}_1$, but the number of available table entries decreases from 2^{10} to 226.536. Therefore, the cost to recover $\tilde{T}_2$ is larger than the cost to recover $\tilde{T}_1$. The more rounds we analyze, the more cost increases.

6.3.3 Recovery of $\tilde{T}_{R-1}$

Similarly to the recovery of $\tilde{T}_2$, we recover partial data of $E_r^{(\mathscr{T})}$ $(D_r^{(\mathscr{T})})$ and $T_{r+1}^{(\mathscr{T})} \circ (D_1^{(\mathscr{T})} \| I)$ for each $\mathscr{T}$ and obtain partial data of $\tilde{T}_{r+1}$. By collecting many leakage every $\mathscr{T}$, we recover all table entries of $\tilde{T}_{r+1}$. The following is the summary of this analysis.

- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 2^{10})$. There are 906.143 available table entries about $E_1^{(\mathscr{T})}$. Then, 226.536 table entries about $T_2^{(\mathscr{T})} \circ (D_1^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 226.536)$. There are 220.412 available table entries about $E_2^{(\mathscr{T})}$. Then, 55.103 table entries about $T_2^{(\mathscr{T})} \circ (D_2^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 55.103)$. There are 54.741 available table entries about $E_3^{(\mathscr{T})}$. Then, 13.685 table entries about $T_2^{(\mathscr{T})} \circ (D_3^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 13.685)$. There are 13.664 available table entries about $E_4^{(\mathscr{T})}$. Then, 3.416 table entries about $T_2^{(\mathscr{T})} \circ (D_4^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 3.416)$. There are 3.415 available table entries about $E_5^{(\mathscr{T})}$. Then, 0.8537 table entries about $T_2^{(\mathscr{T})} \circ (D_5^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 0.854)$. There are 0.854 available table entries about $E_6^{(\mathscr{T})}$. Then, 0.213 table entries about $T_2^{(\mathscr{T})} \circ (D_6^{(\mathscr{T})} \| I)$ are available.

To recover $\tilde{T}_{R-1} = \tilde{T}_7 = (E_7^{(\mathscr{T})} \| I) \circ T_2^{(\mathscr{T})} \circ (D_6^{(\mathscr{T})} \| I)$, the number of available table entries is 0.213. Even finding only 1 table entry needs to rely on the probabilistic event.

Let us estimate the number of required table updates. We can use the analysis on the basis of the perfect decomposition shown in Sect. 4.3. Assuming that (x, x') satisfying

$$\mathsf{msb}_m(\tilde{T}_7(x) \oplus \tilde{T}_7(x')) = \mathsf{msb}_m((T_2^{(\mathscr{T})} \circ (D_6^{(\mathscr{T})} \| I))(x) \oplus (T_2^{(\mathscr{T})} \circ (D_6^{(\mathscr{T})} \| I))(x')) = 0,$$

the probability that x (resp. x') appears as known table entries is estimated as 0.213×2^{-16}. Thus, the probability that both x and x' appear as known table entries is $0.213^2 \times 2^{-32}$. In other words, this is $(16, 4, 0.213^2 \times 2^{-32})$-perfect-decomposition. The required number of table updates is estimated as

$$s = (0.213^2 \times 2^{-32})^{-1} \times 2^{-4} \times (\ln 2^4 - \ln(-2^{-12} \times \ln(p_{succ}))).$$

With $p_{succ} = 0.5$, $s \approx 2^{35.97}$. Therefore, we need to collect leakage of $(T_1^{(\mathscr{T})}, T_2^{(\mathscr{T})}, T_3^{(\mathscr{T})})$ with $2^{35.97}$ $\mathscr{T}$.

To recover $\tilde{T}_{R-1}$, we analyze $2^{n_{in}-6}$ leaked table entries every table update. Thus, the attack complexity for each table update is roughly estimated as $(R-1) \times 2^{n_{in}-6}$, which is $2^{12.81}$ in Yoroi-16. Thus, the total time complexity is $2^{12.81+35.97} \approx 2^{48.78}$.

6.3.4 Recovery of $\tilde{T}_R$

We finally recover $\tilde{T}_R$.

Table 4: Experimental result for the attack on the longevity of Yoroi-10.

trial	1	2	3	4	average	theoretical
s	11644781	8896042	8045264	7639131	$9056304.5 \approx 2^{23.11}$	$2^{23.26}$

- We use Corollary 1 with $(n, k_0, t) = (2^{12}, 0, 0.2134)$. There are 0.2134 available table entries about $E_7^{(\mathscr{T})}$. Then, about 0.05336 table entries about $T_3^{(\mathscr{T})} \circ (D_7^{(\mathscr{T})} \| I)$ are available.

The probability 0.05336 is lower than 0.2134, which is the case to recover $\tilde{T}_{R-1}$. However, we do not need to search for collision in this step because of no post-mask, i.e., $\tilde{T}_{R-1} = T_3^{(\mathscr{T})} \circ (D_7^{(\mathscr{T})} \| I)$. Due to the coupon collector's problem, we need to check roughly 2^{20} values. Therefore, $1/0.05336 \times 2^{20} \approx 2^{24.23}$ table updates are required to recover $\tilde{T}_R$, but it is negligible compared to $2^{35.97}$.

6.3.5 Remark on Yoroi-32

The same analysis can be applied to Yoroi-32. Similarly to Yoroi-16, the dominant part is the recovery of $\tilde{T}_{15}$. At this step,

- There are 0.853 available table entries about $E_{14}^{(\mathscr{T})}$. Then, about 0.213 table entries about $T_2^{(\mathscr{T})} \circ (D_{14}^{(\mathscr{T})} \| I)$ are available.

Assuming that (x, x') satisfies

$$\mathsf{msb}_m(\tilde{T}_{15}(x) \oplus \tilde{T}_{15}(x')) = \mathsf{msb}_m((T_2^{(\mathscr{T})} \circ (D_{14}^{(\mathscr{T})} \| I))(x) \oplus (T_2^{(\mathscr{T})} \circ (D_{14}^{(\mathscr{T})} \| I))(x')) = 0,$$

the probability that x (resp. x') appears as known table entries is estimated as 0.213×2^{-32}. Thus, the probability that both x and x' appear as known table entries is $0.213^2 \times 2^{-64}$. In other words, this is a $(32, 4, 0.213^2 \times 2^{-64})$-perfect-decomposition. The required number of table updates is estimated as

$$s = (0.213^2 \times 2^{-64})^{-1} \times 2^{-4} \times (\ln 2^4 - \ln(-2^{-28} \times \ln(p_{succ}))).$$

With $p_{succ} = 0.5$, $s \approx 2^{68.95}$.

The total time complexity is $(R-1) \times 2^{32-6} \times 2^{68.95} = 2^{29.91+68.95} = 2^{98.86}$.

6.4 Experimental Reports

Considering the attack complexity of $2^{48.78}$ for Yoroi-16, our attack is difficult to experimentally verify. Therefore, to verify the correctness of our complexity analysis, we implemented our attack with small-scaled Yoroi. We use Yoroi-10, where $n_{in} = 10$, $t = 4$, $m = 6$, and $R = 5$.

We first show the theoretical analysis.

- Use Corollary 1 with $(n, k_0, t) = (2^6, 0, 2^4)$. There are 14.266 available table entries about $E_1^{(\mathscr{T})}$. Then, 3.566 table entries about $T_2^{(\mathscr{T})} \circ (D_1^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^6, 0, 3.566)$. There are 3.494 available table entries about $E_2^{(\mathscr{T})}$. Then, 0.874 table entries about $T_2^{(\mathscr{T})} \circ (D_2^{(\mathscr{T})} \| I)$ are available.
- Use Corollary 1 with $(n, k_0, t) = (2^6, 0, 0.874)$. There are 0.8735 available table entries about $E_3^{(\mathscr{T})}$. Then, 0.218 table entries about $T_2^{(\mathscr{T})} \circ (D_3^{(\mathscr{T})} \| I)$ are available.

To recover $\tilde{T}_{R-1} = \tilde{T}_4 = (E_4^{(\mathscr{T})} \| I) \circ T_2^{(\mathscr{T})} \circ (D_3^{(\mathscr{T})} \| I)$, the number of available table entries is 0.218. The probability that x (resp. x') appears as known table entries is estimated as 0.218×2^{-10}. Thus, the probability that both x and x' appear as known table entries is $0.218^2 \times 2^{-20}$. In other words, this is $(10, 4, 0.218^2 \times 2^{-20})$-perfect-decomposition. The required number of table updates is estimated as

$$s = (0.2184^2 \times 2^{-20})^{-1} \times 2^{-4} \times (\ln 2^4 - \ln(-2^{-6} \times \ln(p_{succ}))).$$

With $p_{succ} = 0.5$, $s \approx 2^{23.26}$.

Table 4 shows the experimental results for the number of the required table updates to recover $\tilde{T}_{R-1}$. We conducted four experiments, and the average number of required table updates is about $2^{23.11}$. Therefore, our theoretical estimation works very well.

6.5 Remark on ACSA and Countermeasure

In the KSA, we need to rely on probabilistic events to recover table entries about $E_r^{(\mathscr{T})}$ and $T_2^{(\mathscr{T})} \circ (D_{r-1}^{(\mathscr{T})} \| I)$. Therefore, the increase in the number of rounds makes the KSA difficult. However, we do not think such a countermeasure is reasonable because it is unlikely to resist not only arbitrary leakage but also the ACSA. In the ACSA, the attacker can extract table entries such that as many table entries about $E_r^{(\mathscr{T})}$ and $T_2^{(\mathscr{T})} \circ (D_{r-1}^{(\mathscr{T})} \| I)$ as possible are known. Therefore, the security enhancement is slow even if the number of rounds increases.

We believe that space-hard block ciphers should be secure against not only the KSA but also ACSA. Thus, the problem of how to design space-hard block ciphers with longevity goes back to being an open problem.

7 Hybrid Code Lifting on SPNbox

The hybrid code lifting is an attack model to evaluate the blackbox security after the code lifting by a whitebox attacker. It is interesting if other space-hard block ciphers are still secure in this attack model. In this section, we discuss the security against the hybrid code lifting on SPNbox [BIT16].

SPNbox has four variants: SPNbox-32, SPNbox-24, SPNbox-16, and SPNbox-8. For each, the whitebox implementation uses a single table $T_b : \mathbb{F}_2^{n_{in}} \to \mathbb{F}_2^{n_{in}}$, where n_{in} is 32, 24, 16, and 8, respectively, and the number of rounds of all variants is $R = 10$. We denote the number of table entries by N, and the claimed security of all variants in the whitebox model is $(N/4, 64)$-space hardness.

We evaluate the security against the hybrid code lifting with parameter $(\lambda, \tau_w, \tau_b) = (T/4 \times n_{in}, 2^{64}, 2^{128})$. In other words, we evaluate if SPNbox still maintains 128-bit security against key-recovery attacks in the blackbox model even if $T/4$ table entries are leaked.

7.1 Specifications of SPNbox

SPNbox has four variants of SPNbox-8, SPNbox-16, SPNbox-24, and SPNbox-32 [BIT16]. SPNbox-n_{in} is a substitution-permutation network (SPN) with a block length of n bits, a 128-bit secret key, and based on n_{in}-bit substitution boxes. For SPNbox-8, SPNbox-16, and SPNbox-32, the block length is $n = 128$ bits, whereas SPNbox-24 has $n = 120$.

The state of SPNbox-n_{in} is organized as a vector of $\ell = n/n_{in}$ elements of n_{in} bits each:

$$X = \{X_1, \dots, X_\ell\}.$$

A plaintext X^0 is encrypted to a ciphertext X^R by applying R rounds of the following round transformation to the plaintext:

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

For all concrete proposals, SPNbox-8, SPNbox-16, SPNbox-24, and SPNbox-32, we set the number of rounds to $R = 10$. We now define each of the components γ, θ, and σ^r.

The Nonlinear Layer γ. γ is a nonlinear substitution layer, in which t key-dependent identical bijective n_{in}-bit S-boxes are applied to the state:

$$\begin{array}{rcl} \gamma : (\mathbb{F}_2^{n_{in}})^\ell & \to & (\mathbb{F}_2^{n_{in}})^\ell \\ (X_1, \ldots, X_\ell) & \to & (S_{n_{in}}(X_1), \ldots, S_{n_{in}}(X_\ell)). \end{array}$$

In SPNbox-n_{in}, the substitution $S_{n_{in}}$ is realized by a dedicated small block cipher of block length n_{in}. In this paper, we omit the specifications of small block ciphers and refer to [BIT16] for the details.

The Linear Layer θ. θ is a linear diffusion layer that applies a $t \times t$ MDS matrix to the state:

$$\begin{array}{c} \theta : (\mathbb{F}_2^{n_{in}})^\ell \to (\mathbb{F}_2^{n_{in}})^\ell \\ (X_1, \ldots, X_\ell) \mapsto (X_1, \ldots, X_\ell) \cdot M_{n_{in}}. \end{array}$$

We denote by $cir(a_0, \ldots, a_{\ell-1})$ the $\ell \times \ell$ circulant matrix A with the coefficients $a_0, \ldots, a_{\ell-1}$ in the first row; and by $had(a_0, \ldots, a_{\ell-1})$ the $\ell \times \ell$ Hadamard matrix A with coefficients $A_{i,j} = a_{i \oplus j}$, with ℓ a power of two.

For the concrete proposals SPNbox-n_{in} with $n_{in} = 32, 24, 16, 8$, the matrix $M_{n_{in}}$ is defined as follows:

$$\begin{array}{ll} M_{32} = cir(\mathtt{1}_x, \mathtt{2}_x, \mathtt{4}_x, \mathtt{6}_x) & \text{for } n_{in} = 32, \\ M_{24} = cir(\mathtt{1}_x, \mathtt{2}_x, \mathtt{5}_x, \mathtt{3}_x, \mathtt{4}_x) & \text{for } n_{in} = 24, \\ M_{16} = had(\mathtt{1}_x, \mathtt{3}_x, \mathtt{4}_x, \mathtt{5}_x, \mathtt{6}_x, \mathtt{8}_x, \mathtt{b}_x, \mathtt{7}_x) & \text{for } n_{in} = 16, \end{array}$$

and

$$\begin{array}{l} M_8 = \text{had}(\mathtt{08}_x, \mathtt{16}_x, \mathtt{8a}_x, \mathtt{01}_x, \mathtt{70}_x, \mathtt{8d}_x, \mathtt{24}_x, \mathtt{76}_x, \\ \qquad\qquad \mathtt{a8}_x, \mathtt{91}_x, \mathtt{ad}_x, \mathtt{48}_x, \mathtt{05}_x, \mathtt{b5}_x, \mathtt{af}_x, \mathtt{f8}_x) \\ \quad \text{for } n_{in} = 8. \end{array}$$

The Affine Layer σ^r. σ^r is an affine layer that adds round-dependent constants to the state:

$$\begin{array}{c} \sigma^r : (\mathbb{F}_2^{n_{in}})^\ell \to (\mathbb{F}_2^{n_{in}})^\ell \\ (X_0, \ldots, X_{\ell-1}) \mapsto (X_1 \oplus C_1^r, \ldots, X_\ell \oplus C_\ell^r), \end{array}$$

with $C_i^r = (r-1) \cdot \ell + i$ for $1 \leq i \leq \ell$.

7.2 Attack Procedure

7.2.1 Code-Lifting Phase

The hybrid code lifting allows not only table entries but also arbitrary leakage. For example, extracting plaintext-intermediatetext pairs is possible, but it is non-trivial to

Table 5: Summary of simple hybrid code lifting on SPNbox.

cipher	n_{in}	ℓ	$\mathcal{N}$	$\mathcal{M}_0$	$\mathcal{M}$	$\mathcal{M}^\ell$	$(\mathcal{M}/2^{n_{in}})^{(R-2)\times\ell}$
SPNbox-32	32	4	38612034	79	1187519920	$2^{120.66}$	$2^{-59.35}$
SPNbox-24	24	5	120662	11	4638747	$2^{110.82}$	$2^{-74.19}$
SPNbox-16	16	8	294	3	18116	$2^{113.31}$	$2^{-118.72}$
SPNbox-8	8	16	0	0	64	2^{96}	2^{-256}

recover the encryption program from them. As far as we analyze, there are no better methods than extracting a part of table entries.

Let ℓ be the number of S-boxes each round, i.e., 10ℓ table lookups are used to encrypt one plaintext. A whitebox attacker first picks $\mathcal{M}_0$ table entries, and then, $\mathcal{N} \leq \mathcal{M}_0^\ell$ plaintexts are always encrypted up to the 1st round. The attacker additionally extracts table entries that enable us to encrypt these $\mathcal{N}$ plaintexts up to one round before the last round. Then, $\mathcal{N} \times (R-2) \times \ell$ (overlapping) table entries are extracted. This is equivalent to the coupon collector's problem, where there are $2^{n_{in}}$ coupons and $2^{n_{in}-2}$ coupons are collected with $\mathcal{N} \times (R-2) \times \ell$ trials[6]. Due to Corollary 1,

$$2^{n_{in}-2} \approx 2^{n_{in}} - e^{\ln(2^{n_{in}}) - \frac{\mathcal{N}\times(R-2)\times\ell}{2^{n_{in}}}},$$

$$\mathcal{N} \approx \left\lfloor \frac{2^{n_{in}} \times (\ln 2^{n_{in}} - \ln(2^{n_{in}} - 2^{n_{in}-2}))}{(R-2)\times\ell} \right\rfloor.$$

We estimate $\mathcal{N}$ and $\mathcal{M}_0$ on each parameter, and Table 5 summarizes these results. We notice $\mathcal{N} = 0$ in SPNbox-8, but this is convincing because we need $(R-2)\times 16 = 128$ table lookups but the size of leaked table entries is only 64 $(= \frac{2^8}{4})$.

7.2.2 Blackbox Phase

The blackbox phase consists of two steps.

The 1st Step. In the 1st step of the blackbox phase, $\mathcal{N}$ chosen plaintexts are encrypted up to one round before the last round in the local of the blackbox attacker. The attacker also queries these chosen plaintexts to the encryption oracle. Then, $\mathcal{N} \times \ell$ (overlapping) table entries are extracted from the last round. This is equivalent to the coupon collector's problem, where there are $2^{n_{in}}$ coupons, $2^{n_{in}-2}$ coupons are already collected, and additional coupons are collected by $\mathcal{N} \times \ell$ trials. Due to Corollary 1, the number of finally collected table entries is estimated as follows.

$$\mathcal{M} \approx \lfloor 2^{n_{in}} - e^{\ln(2^{n_{in}} - 2^{n_{in}-2}) - \frac{\mathcal{N}\times\ell}{2^{n_{in}}}} \rfloor.$$

We estimate $\mathcal{M}$ on each parameter, and Table 5 summarizes these results.

The 2nd Step (Simple Procedure). The goal of this step is to recover the whole table entries. We already have $\mathcal{M}$ table entries. The coupon collector's problem is useful to estimate how many (overlapping) table entries are required to collect all table entries. Due to Theorem 1, the number of trials to collect all coupons is estimated as $2^{n_{in}} \times (\ln(2^{n_{in}} - \mathcal{M}) + 0.577)$, which are $2^{36.49}$, $2^{28.08}$, $2^{19.50}$, and $2^{10.54}$ for SPNbox-32, SPNbox-24, SPNbox-16, and SPNbox-8, respectively.

[6] Accurately, there are $2^{n_{in}}$ coupons, we already have $\mathcal{M}_0$ coupons, and $2^{n_{in}-2} - \mathcal{M}_0$ coupons are collected with $\mathcal{N} \times (R-2) \times \ell$ trials. However, we neglect the impact on $\mathcal{M}_0$ for simplicity. In practice, the attacker can repeat this procedure and find $\mathcal{M}_0$ table entries such that $\mathcal{N}$ plaintexts can be encrypted up to one round before the last round by using $2^{n_{in}-2}$ leaked table entries. Because $\mathcal{M}_0$ is very small, it is easy to find such table leakage for a whitebox attacker running in time 2^{τ_w}.

Whether we can obtain these overlapping table entries is important. There are $\mathcal{M}^\ell$ chosen plaintexts that are encrypted up to the 1st round. Since lookups of each S-box are known with probability of $(M/2^{n_{in}})$, the probability that we can encrypt $\mathcal{M}^\ell$ chosen plaintexts up to one round before the last round is $(\mathcal{M}/2^{n_{in}})^{(R-2)\times\ell}$. These values on each parameter are summarized in Table 5. In SPNBOX-32 and SPNBOX-24, the following

$$\mathcal{M}^\ell \times (\mathcal{M}/2^{n_{in}})^{(R-2)\times\ell} > 1$$

holds. Then, we observe $\left\lfloor \mathcal{M}^\ell \times (\mathcal{M}/2^{n_{in}})^{(R-2)\times\ell} \right\rfloor \times \ell$ (overlapping) table entries, which are $2^{63.23}$ and $2^{38.86}$ for SPNBOX-32 and SPNBOX-24, respectively. Since they are larger than $2^{36.49}$ and $2^{28.08}$, respectively, it recovers the whole table entries of both SPNBOX-32 and SPNBOX-24. The number of required chosen plaintexts is $2^{36.49}/2^{-59.35} = 2^{95.84}$ and $2^{28.08}/2^{-74.19} = 2^{102.27}$ of SPNBOX-32 and SPNBOX-24, respectively. Moreover, the time complexity is the same as the data complexity. On the other hand, this procedure cannot find any table entry of SPNBOX-16 and SPNBOX-8.

According to Table 2 in [BIT16], $N/2^{3.20}$ and $N/2^{2.57}$ random table entries are required to encrypt a plaintext with probability 2^{-128} of SPNBOX-32 and SPNBOX-24, respectively. Thus, it is natural that they are vulnerable to the hybrid code lifting when quarter-size table entries are leaked. On the other hand, $N/2^{1.61}$ table entries are required of SPNBOX-16 in [BIT16]. Thus, it is interesting whether SPNBOX-16 is secure against the hybrid code lifting or not.

The 2nd Step (Filter-then-Guess). To attack SPNBOX-16, we consider a more advanced technique, which accepts some unknown table entries[7].

When there are s unknown table entries to encrypt plaintexts, we guess s unknown table entries. However, we face two difficulties. 1) Is it possible that we guess s unknown table entries whose time complexity is faster than 2^{128}? 2) Is it possible to filter all wrong guesses? For the 1st question, the filter-then-guess procedure is useful. We first filter available data, reduce the size, and then guess unknown table entries. For the 2nd question, we use the case that some table entries are known in the last round, and wrong keys are filtered by comparing these known entries and ciphertexts given by the encryption oracle.

Let $\vec{s} = (s_1, s_2, \ldots, s_R)$ be a vector representation, where s_r denotes the number of the access to unknown table entries in the rth round. When we use $\mathcal{M}^\ell$ chosen plaintexts, we can always evaluate the first round, i.e., $s_1 = 0$, and there are $\mathcal{M}^\ell$ inputs for the 2nd round. Given $\mathcal{M}^\ell$ inputs, we accept s_2 accesses to unknown table entries. Therefore, the filter-then-guess procedure generates

$$\mathcal{M}^\ell \times \left\{ \binom{\ell}{s_2} \times \left(1 - \frac{\mathcal{M}}{2^{n_{in}}}\right)^{s_2} \times \left(\frac{\mathcal{M}}{2^{n_{in}}}\right)^{\ell - s_2} \right\} \times (2^{n_{in}} - \mathcal{M})^{s_2}$$

outputs, which becomes inputs of the 3rd round. Note that the time complexity is the same as the value above. Given $\vec{s}$, the time complexity can be evaluated by the iterative applications, and we only use $\vec{s}$'s such that the time complexity is enough lower than 2^{128}, i.e., $\leq 2^{120}$. Let $\mathbb{S}$ be the set of such an $\vec{s}$, and the total time complexity is the sum of the time complexity each $\vec{s} \in \mathbb{S}$.

The probability that plaintexts can be encrypted (accepting these guesses) up to one round before the last round is

$$p = \sum_{\vec{s} \in \mathbb{S}} \left(\prod_{r=2}^{R-1} \binom{\ell}{s_r} \right) \times \left(1 - \frac{\mathcal{M}}{2^{n_{in}}}\right)^{s} \times \left(\frac{\mathcal{M}}{2^{n_{in}}}\right)^{(R-2)\times\ell - s},$$

[7]The authors of [CCD+17] also discussed a similar analysis accepting guesses of some unknown table entries. They analyzed whether there is a plaintext that is successfully encrypted with a probability higher than 2^{-128}. The difference is that we show a concrete attack procedure whose running time is lower than 2^{128}.

Table 6: Summary of filter-then-guess technique on SPNbox.

cipher	R	s	$\mathcal{N}$	$\mathcal{M}$	p	time	p_w	$\mathcal{M}'$
SPNbox-16	10	4	294	18116	$2^{-94.41}$	$2^{124.09}$	$2^{-3.98}$	$2^{16.69}$
SPNbox-16	11	6	261	17925	$2^{-100.32}$	$2^{126.57}$	$2^{-16.61}$	$2^{4.97}$
SPNbox-24	13	3	87754	4519115	$2^{-86.05}$	$2^{123.93}$	$2^{-6.57}$	$2^{20.79}$
SPNbox-24	14	4	80441	4492370	$2^{-91.25}$	$2^{125.11}$	$2^{-16.06}$	$2^{11.74}$
SPNbox-24	15	4	74253	4469694	$2^{-100.30}$	$2^{125.69}$	$2^{-25.17}$	$2^{2.62}$
SPNbox-32	16	3	22064019	1139258434	$2^{-91.85}$	$2^{123.31}$	$2^{-12.50}$	$2^{28.49}$
SPNbox-32	17	3	20593084	1134932419	$2^{-98.60}$	$2^{123.36}$	$2^{-19.289}$	$2^{15.62}$
SPNbox-32	18	3	19306017	1131142294	$2^{-118.74}$	$2^{123.55}$	$2^{-26.51}$	$2^{8.40}$

where $s = \sum_{r=2}^{R-1} s_r$. Therefore, $\mathcal{M}^\ell \times p$ plaintexts can be successfully encrypted (if each guess is correct) up to one round before the last round. However, as we already mention in the 2nd question, we need to filter wrong guesses. Since there are at most $(2^{n_{in}} - \mathcal{M})^s$ expansions each plaintext by guessing table entries, we observe at most $(2^{n_{in}} - \mathcal{M})^s \times \mathcal{M}^\ell \times p$ wrong encryptions. To filter them, we use data, where $s+1$ out of ℓ accesses to table entries are known in the last round. The probability that such data appears is $f = \binom{\ell}{s+1} \times \left(1 - \frac{\mathcal{M}}{2^{n_{in}}}\right)^{\ell-s-1} \times \left(\frac{\mathcal{M}}{2^{n_{in}}}\right)^{s+1}$. For wrong guess, we have $((s+1) \times n_{in})$-bit filter. Therefore, when $p_w = (2^{n_{in}} - \mathcal{M})^s \times \mathcal{M}^\ell \times p \times f \times 2^{-(s+1)n_{in}} < 1$, we expect that all wrong guesses are discarded. On the other hand, when we correctly guess, $M^\ell \times p \times f$ plaintexts are available, and $\mathcal{M}' = M^\ell \times p \times f \times (\ell - 1)$ (overlapping) table entries are recovered.

Table 6 summarizes the time complexity and the size of the recovered (overlapping) table entries for each parameter. Note that these recovered table entries are not overlapped in the original $\mathcal{M}$ entries, but they could be overlapped inside of the recovered entries. In SPNbox-16, $2^{16} - 18116 = 47420$ and because of Corollary 1, $47420 - e^{\ln(47420) - \frac{2^{16.69}}{47420}} \approx 2^{15.37}$ new table entries are recovered. The remaining table entries are easily recovered because almost all table entries ($2^{14} + 2^{15.37} \approx 2^{15.88}$) are recovered already.

For 11-round SPNbox-16, $2^{4.97}$ (overlapped) table entries are recovered. However, since our goal is to recover the full table entries, it has not been clear whether 11-round SPNbox-16 can be attacked or not. As far as we analyze, we cannot recover any table entry for 12-, 16, and 19-round SPNbox-16, SPNbox-24, and SPNbox-32, respectively.

8 Conclusion

In this paper, we propose a new attack model called *hybrid code lifting* for space-hard block ciphers. We implicitly expected the leakage-resilient security by the space hardness, but it is not always equal. Our attack model reflects the leakage-resilient security, where λ-bit leakage by a whitebox attacker running in time 2^{τ_w} does not decrease the blackbox security. As an application, we showed practical program recovery attacks on Yoroi. Moreover, we also break the security claim about the *longevity*.

Hybrid code lifting and our attack against longevity present many future topics for both attacks and designs for space-hard block ciphers.

On attacks, the security against hybrid code lifting on existing space-hard block ciphers such as SPACE [BI15] or WhiteBlock [FKKM16] can be discussed. We evaluated SPNbox, but the technique is straightforward. More advanced and non-trivial attacks are open questions. In particular, we leave it as an open question whether we can attack 12-, 16-, and 19-round SPNbox-16, SPNbox-24, and SPNbox-32, respectively.

In designs, our attacks against the longevity on Yoroi are critical, and Yoroi is not easy to tweak to resist our attack. Although it would be possible to resist only known-space attacks by increasing the number of rounds, it is unlikely to resist not only arbitrary leakage but also adaptive chosen-space attacks. The problem of designing such ciphers goes back to being an open problem.

Our attack goal was a program recovery to demonstrate apparent vulnerability in the use case of space-hard ciphers. On the other hand, for future design, only defending program recovery is unlikely sufficient. For example, we should not allow blackbox attackers to recover the program, which can encrypt/decrypt many (but not all) plaintexts/ciphertexts. It is unavoidable that λ-bit leakage can leak λ-bit plaintext-ciphertext pairs. Therefore, one of the possible security goals is that blackbox attackers cannot output plaintext-ciphertext pairs beyond λ-bit data without an encryption oracle. In practice, as far as we analyze, 12-round SPNbox-16 can be a good candidate for ensuring such strong security.

Acknowledgments

The authors would like to thank the reviewers of ToSC. Their comments are very helpful to improve our paper. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031 and Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141).

References

[ABCW21] Shashank Agrawal, Estuardo Alpirez Bock, Yilei Chen, and Gaven J. Watson. White-box cryptography with device binding from token-based obfuscation and more. *IACR Cryptol. ePrint Arch.*, page 767, 2021.

[All14] Smart Card Alliance. A smart card alliance mobile & nfc council white paper, host card emulation (hce) 101, 2014.

[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.

[BBB$^+$19] Estuardo Alpirez Bock, Joppe W. Bos, Chris Brzuska, Charles Hubain, Wil Michiels, Cristofaro Mune, Eloi Sanfelix Gonzalez, Philippe Teuwen, and Alexander Treff. White-box cryptography: Don't forget about grey-box attacks. *J. Cryptol.*, 32(4):1095–1143, 2019.

[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, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, 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, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 63–84. Springer, 2014.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White Box Cryptography: Another Attempt. *IACR Cryptology ePrint Archive*, 2006:468, 2006.

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In *SAC 2004*, pages 227–240, 2004.

[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, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, 2016.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1058–1069. ACM, 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, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 126–158, 2016.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven D. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 373–402. Springer, 2018.

[BU21] Alex Biryukov and Aleksei Udovenko. Dummy shuffling against algebraic attacks in white-box implementations. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 219–248. Springer, 2021.

[CC19] CryptoExperts and Cybercrypt. The WhibOx Contest - edition 2. https://whibox.io/contests/2019/, 2019. CHES 2019 Capture the Flag Challenge.

[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, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 293–308. Springer, 2017.

[CEJvO02a] 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, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *ACM CCS-9, DRM 2002*, volume 2696 of *LNCS*, pages 1–15. Springer, 2002.

[con17] ECRYPT-CSA consortium. The WhibOx Contest: An ecrypt white-box cryptography competition. https://whibox.io/contests/2017/, 2017. CHES 2017 Capture the Flag Challenge.

[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, *SAC 2013*, volume 8282 of *LNCS*, pages 247–264. Springer, 2013.

[ER59] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[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, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 159–188, 2016.

[int18] intertrust. Intertrust white paper, taking steps to protect financial mobile applications, 2018.

[Kar10] Mohamed Karroumi. Protecting White-Box AES with Dual Ciphers. In *ICISC 2010*, pages 278–291, 2010.

[KI21] Yuji Koike and Takanori Isobe. Yoroi: Updatable whitebox cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):587–617, 2021.

[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, *CT-RSA 2020*, volume 12006 of *LNCS*, 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, *ACISP 2020*, volume 12248 of *LNCS*, pages 142–159. Springer, 2020.

[LN05] Hamilton E. Link and William D. Neumann. Clarifying Obfuscation: Improving the Security of White-Box DES. In *ITCC 2005*, pages 679–684, 2005.

[LRM$^+$13] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two Attacks on a White-Box AES Implementation. In *SAC 2013*, pages 265–285, 2013.

[MRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai White-Box AES Implementation. In *SAC 2012*, pages 34–49, 2012.

[MWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In *INDOCRYPT 2010*, pages 292–310, 2010.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In *SAC 2007*, pages 264–277, 2007.

A Trivial Chosen-Space Attack

The authors of [KI21] claimed the $(N/4)$-space hardness against the known/chosen-space attack. However, the analysis described in [KI21] is only for the known-space attack, and the authors overlooked a gap between the known-space attack and chosen-space attack when the space-hard block cipher uses multiple tables. Yoroi has S_1, S_2, and S_3, and S_2 is used more than S_1 and S_3. Therefore, extracting more table entries from S_2 is advantageous for attackers.

Let us consider the case of Yoroi-16. We extract $2^{n_{in}}/8$, $2^{n_{in}}/2$, and $2^{n_{in}}/8$ table entries from S_1, S_2, and S_3, respectively. The total leakage size is

$$2^{n_{in}} \times \left(\frac{1}{8} + \frac{1}{2} + \frac{1}{8}\right) = 3 \times 2^{n_{in}} \times \frac{1}{4}.$$

Then, the probability that any randomly drawn plaintext is encrypted is

$$2^{-3\times 8} \times 2^{-6\times 8} \times 2^{-3\times 8} = 2^{-96},$$

which is clearly higher than 2^{-128}.

B Understanding Canonical Representation

We exploit the canonical representation in our attacks against Yoroi. In this appendix, we provide a small example to understand the representation.

Let us consider a small-scaled Yoroi, where a 4-bit bijective function is used instead of the n_{in} bijective function. We use $t = 2$ and $m = 2$. Namely, θ and σ^r are applied to the last 2 bits, and the top 2 bits become a direct input of the next round. We suppose that θ and σ^r are revised adequately according to the change of bit length.

As an example, let us consider the following 4-bit bijective function S.

x	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$S(x)$	0001	1010	0100	1100	0110	1111	0011	1001	0010	1101	1011	0111	0101	0000	1000	1110

We divide 4 bits into top 2 bits and bottom 2 bits. Let $\mathbb{A}_i := \{x \in \mathbb{F}_2^4 | \mathsf{lsb}_2(S(x)) = i\}$. Then, with $\rho \in (\mathbb{F}_2^2)^2$,

$$\begin{aligned}
\mathbb{A}_{\rho_{00}} &:= \{\mathsf{0000}, \mathsf{0111}, \mathsf{1001}, \mathsf{1100}\} & \rho_{00} &= \mathsf{01}\\
\mathbb{A}_{\rho_{01}} &:= \{\mathsf{0001}, \mathsf{0100}, \mathsf{1000}, \mathsf{1111}\} & \rho_{01} &= \mathsf{10}\\
\mathbb{A}_{\rho_{10}} &:= \{\mathsf{0010}, \mathsf{0011}, \mathsf{1101}, \mathsf{1110}\} & \rho_{10} &= \mathsf{00}\\
\mathbb{A}_{\rho_{11}} &:= \{\mathsf{0101}, \mathsf{0110}, \mathsf{1010}, \mathsf{1011}\} & \rho_{11} &= \mathsf{11}
\end{aligned}$$

Similarly, let $\mathbb{B}_j := \{x \in \mathbb{F}_2^4 | \mathsf{msb}_2(S(x)) = j\}$. Then, with $\eta \in (\mathbb{F}_2^2)^2$,

$$\begin{aligned}
\mathbb{B}_{\eta_{00}} &:= \{\mathsf{0000}, \mathsf{0110}, \mathsf{1000}, \mathsf{1101}\} & \eta_{00} &= \mathsf{00}\\
\mathbb{B}_{\eta_{01}} &:= \{\mathsf{0001}, \mathsf{0111}, \mathsf{1010}, \mathsf{1110}\} & \eta_{01} &= \mathsf{10}\\
\mathbb{B}_{\eta_{10}} &:= \{\mathsf{0010}, \mathsf{0100}, \mathsf{1011}, \mathsf{1100}\} & \eta_{10} &= \mathsf{01}\\
\mathbb{B}_{\eta_{11}} &:= \{\mathsf{0011}, \mathsf{0101}, \mathsf{1001}, \mathsf{1111}\} & \eta_{11} &= \mathsf{11}
\end{aligned}$$

Note that we can compute $S(x)$ for any x from $\mathbb{A}_i$ and $\mathbb{B}_j$. For example, $S(\mathsf{1010}) = \eta_{01} \| \rho_{11} = 1011$ because $\mathsf{1010}$ belongs to $\mathbb{A}_{\rho_{11}}$ and $\mathbb{B}_{\eta_{01}}$.

We now change this S-box to the one in the canonical representation, where Property 1 holds in $(E\|I) \circ S$. Specifically, we focus on $\mathbb{A}_{\mathtt{00}}$, and

$$\begin{aligned}
\mathtt{0010} &\xrightarrow{S} \mathtt{0100} \xrightarrow{E\|I} \mathtt{0000},\\
\mathtt{0011} &\xrightarrow{S} \mathtt{1100} \xrightarrow{E\|I} \mathtt{0100},\\
\mathtt{1101} &\xrightarrow{S} \mathtt{0000} \xrightarrow{E\|I} \mathtt{1000},\\
\mathtt{1110} &\xrightarrow{S} \mathtt{1000} \xrightarrow{E\|I} \mathtt{1100}.
\end{aligned}$$

Then, E is defined as follows.

x	00	01	10	11
$E(x)$	10	00	11	01

Note that the canonical representation does not change $\mathbb{A}_i$, but it changes the index η_j of $\mathbb{B}_{\eta_j}$. Corresponding η_j can be computed by sorting all elements of $\mathbb{A}_{\mathtt{0}}$. In the case above, we focus on $\mathbb{A}_{\mathtt{00}} = \{\mathtt{0010}, \mathtt{0011}, \mathtt{1101}, \mathtt{1110}\}$. Then, 0010, 0011, 1101, and 1110 belong to $\mathbb{B}_{\eta_{\mathtt{10}}}$, $\mathbb{B}_{\eta_{\mathtt{11}}}$, $\mathbb{B}_{\eta_{\mathtt{00}}}$, and $\mathbb{B}_{\eta_{\mathtt{01}}}$, respectively. After applying $(E\|I)$,

$$\eta_{\mathtt{10}} < \eta_{\mathtt{11}} < \eta_{\mathtt{00}} < \eta_{\mathtt{01}}$$

holds, and it implies

$$\eta_{\mathtt{10}} = \mathtt{00}, \quad \eta_{\mathtt{11}} = \mathtt{01}, \quad \eta_{\mathtt{00}} = \mathtt{10}, \quad \eta_{\mathtt{01}} = \mathtt{11}$$

when the canonical representation is used.

After we change the S-box to the one in the canonical representation by applying $E\|I$ after the S-box, we next go to the next round. Note that the S-box in the next round is $S \circ (E^{-1}\|I)$ to maintain the functionality. Therefore, we next change $S \circ (E^{-1}\|I)$ to the one in the canonical representation.