IACR Transactions
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–32. DOI:XXXXXXXX

RubikStone: Strongly Space Hard White-Box Scheme Based on Lookup Table Pool and Key Guidance Implementation

Yipeng Shi

Shanghai Jiao Tong University, Shanghai, China, siponline@sjtu.edu.cn

Abstract. White-box cryptography is a software implementation technique based on lookup tables, with effective resistance against key extraction and code lifting attacks being a primary focus of its research. Space hardness is a widely used property for evaluating the resistance of white-box ciphers against code lifting attacks. However, none of the existing ciphers can provide strong space hardness under adaptively chosen-space attack model.

We propose a new scheme based on the lookup table pool and key guidance implementation as a more efficient approach to utilizing lookup tables to provide better security and practicality. Specifically, we introduce a new white-box cipher, RubikStone, which offers a range of variants from tens of kilobytes to infinite size. For the first time, we prove that all variants of RubikStone can provide strong space hardness under an adaptively chosen-space attack model. Additionally, we present a specific key guidance application for cloud-based DRM scenarios. Based on our proposed RubikStone variants, the key guidance applications can achieve at least overall $(0.950T, 128)$-space hardness.

Furthermore, we introduce a novel property, table consumption rate, for evaluating the durability of a specific white-box cryptographic implementation. In our evaluation, all the instantiations of RubikStone exhibit the lowest table consumption rate in algorithms with equally sized lookup tables. Besides, we conduct a comprehensive statistical analysis of the operations in all existing white-box ciphers. Our findings indicate that RubikStone remains highly competitive in terms of computational efficiency despite offering unprecedented levels of security.

Keywords: White-box cryptography · Space hardness · Lookup table pool · Key guidance implementation · Balanced Feistel network.

1 Introduction

1.1 White-box Cryptography

In untrusted environments, especially on devices lacking sufficient hardware support, how to securely execute cryptographic algorithms is a topic of widespread discussion. In 2002, Chow *et al.* [CEJvO02a, CEJvO02b] differentiated this scenario and traditional ones by white and black box contexts and pioneered a solution known as white-box cryptography. The main idea of white-box cryptography is pre-storing intermediate values that keys may participate in within lookup tables. These table entries are then utilized to substitute the keys during cryptographic operations, effectively ensuring that they do not directly appear in the implementation of cryptographic algorithms. To bolster the security of lookup table entries, additional internal and external encoding, as well as masking methods, are also widely employed. Compared to the computationally expensive fully homomorphic encryption [MOO+14] and the frequently vulnerable secure enclaves [BPS17, MIE17, BMW+18],

Licensed under Creative Commons License CC-BY 4.0.

white-box cryptography has been widely welcomed by industry and is even required for use in some standards [Pay, BBF+20].

Black-box and White-box Contexts. According to Chow *et al.*'s initial perspective, in the white-box context, the adversary is assumed to have "full access" to the implementation of cryptographic algorithms, enabling them to observe the dynamic execution process of the algorithm and modify details at will. While in the traditional black-box context, the adversary can only observe the input-output behavior of the algorithm as a whole and conduct known-plaintext, chosen-plaintext, chosen-ciphertext, or even adaptively chosen-ciphertext attacks based on that. In subsequent works, the notion of white-box context has been further refined(see Section 2.1).

Key Extraction and Code Lifting [DLPR13]. Undoubtedly, the white-box adversary possesses unprecedentedly powerful capabilities in the assumptions made by Chow *et al.*. With such abilities, he can execute cryptographic algorithms and possess the same privileges as a legitimate user. Sometimes, the white-box adversary is actually a legitimate user, referred to as a malicious user. However, the "privileges" are often tied to a specific device. The white-box adversary aims to transplant these privileges to other unauthorized devices and profit from them. To achieve the goal, the white-box adversary typically has two approaches:

- **Key Extraction.** The adversary analyses the information observed during the algorithm execution process and recovers the key, which is the core secret for legitimate users to obtain privileges. For an implementation that uses the key in plaintext form, the white-box adversary can even directly obtain this key by observing the intermediate processes of the algorithm execution without additional analysis.
- **Code Lifting.** In some algorithm implementations, such as the white-box implementation proposed by Chow *et al.*, recovering keys remains a very difficult task for the white-box adversary. In this case, the white-box adversary can simply isolate the cryptographic code in the implementation and lift it as a whole to other devices. In this process, the lifted cryptographic code can be considered as an inflated variant of the original key.

How to effectively defend the two types of attacks in a white-box context has been the focus of white-box cryptography research.

Related Works. In the early stages of white-box cryptography research, the primary focus was on improving the implementation of some existing block ciphers, especially AES [CEJvO02a, BCD06, Kar10, XL09, LLY14] and DES [CEJvO02b, LN05, WP05]. However, most of them have been explicitly broken [BGE04, GMQ07, JBF02, LRM+13, MGH08, MRP12, MWP10, WMGP07]. Due to the significant challenge of designing a white-box implementation for an existing cipher, these works and even some recent ones [RP20, RVP22] only aimed at preventing key extraction attacks.

In order to better prevent both key extraction and code lifting attacks, several dedicated white-box ciphers [BI15, BIT16, FKKM16, CCD+17, KSHI20, KLLM20, KI21, YZDZ23] were proposed later, which are more suitable for white-box application scenarios. By generating a lookup table based on a well-studied block cipher, these ciphers reduce the security against key extraction in a white-box context to that against key recovery in a black-box context. At the same time, by using lookup tables ranging from several hundred kilobytes to tens of gigabytes, they mitigate code lifting attacks to some extent. However, our work indicates that there is still significant room for improvement in the security of current white box ciphers. Moreover, all existing white-box ciphers can

only provide a few specific-sized variants and cannot provide more fine-grained variations to adapt to diverse application scenarios, which will be well addressed in our white-box scheme.

1.2 Application Scenarios

Digital Rights Management (DRM) is widely used to manage legal access to digital content, which is also the initial and most important application scenario of white-box cryptography. To reduce development and business costs for DRM developers and content service providers, while offering content consumers more flexible and diversified services, current DRM services have transitioned to cloud-based content distribution systems [LPSS16, Inc14].

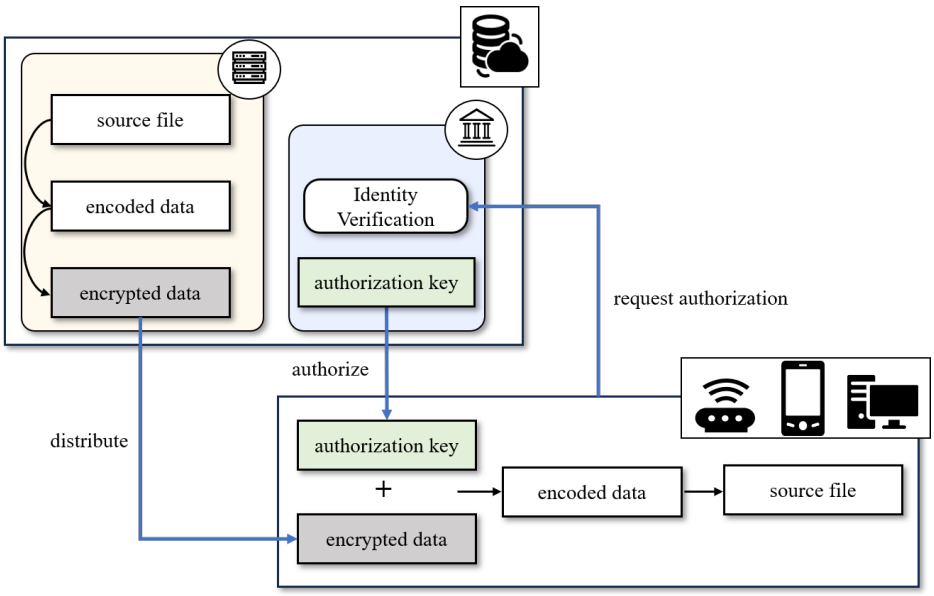


Figure 1: Overview of cloud-based DRM

Figure 1 provides an overview of cloud-based DRM service. The digital content stored on the cloud server is distributed to consumers' devices after being encoded and encrypted. When consumers want to decode the encrypted digital content, they need to interact with the cloud server for rights verification. During the process, the client initiates a verification request and sends identifying information about themselves, such as an ID and a signature. Once legitimate users pass the verification, the cloud server will transmit the corresponding authorization key to the client through a secure channel. Legitimate consumers can then decode the encrypted digital content with the authorization key. If the authorization keys are stored and utilized using conventional methods, the white-box adversary within client devices can easily extract the keys and subsequently distribute them to other unauthorized users. Hence, white-box cryptography holds significant applicability in cloud-based DRM systems.

In addition to cloud-based DRM, white-box cryptography is also considered an effective software security measure in various scenarios such as Host Card Emulation (HCE) in mobile payment services, and memory-leakage resilient software [BIT16]. Previous efforts have aimed to investigate these scenarios collectively, attempting to design a white-box cryptography algorithm capable of providing security assurance for all these scenarios simultaneously. However, we believe that the resources available in different white-box application scenarios vary, and the application methods of white-box cryptography can

also be vastly different at times. Therefore, white-box cryptography, as a software security solution that balances security assurance and existing resources, should be subject to specialized and meticulous research according to different application scenarios. Accordingly, the focus of our work is primarily on the cloud-based DRM scenario, aiming to design a secure and practical white-box scheme tailored to the specific context.

Compared to other white-box cryptography application scenarios, white-box cryptography in cloud-based DRM system exhibits two characteristics:

- **Broader Storage Adaptability.** The devices receiving cloud-based DRM services are diverse, with storage capacities available to the cryptographic modules ranging from a few hundred kilobytes to several tens of gigabytes. This necessitates white-box cryptography to possess broader storage adaptability, enabling it to fully leverage storage resources across different application scenarios to enhance the security against code lifting.
- **Interaction Based Computation.** In the context of cloud-based DRM, the cryptographic applications involve interactions between client devices and cloud servers. Effectively utilizing the interaction process can offer additional security protections for client devices. For instance, the key guidance application(see Section 4) exploits this process to provide enhanced security measures.

1.3 Our Contribution

The implementation of white-box ciphers can be divided into the program part that executes the cryptographic algorithm and the lookup table part that contains the key in a hidden form(Although the lookup tables are not generated based on a specific key in a dedicated white-box cipher, the lookup tables are still an equivalent key). Due to the fact that the white-box adversary has full access to the units that perform calculations, the program part can be easily obtained by the white-box adversary through reverse engineering, disassembly, and other means. Thus, the security of white-box implementation largely relies on the adversary's difficulty in accessing and retrieving lookup tables.

The focus of the paper is to explore a mechanism that better utilizes lookup tables to enhance the security and practicality of white-box cryptography. We emphasize that the security of white-box cryptography is closely related to its usage and the environment in which it is applied. Therefore, the design of white-box schemes should be integrated with specific application scenarios. In this paper, we focus on the primary application scenario of white-box cryptography - the cloud-based DRM system - for the design of concrete schemes. Specifically, the contributions of this paper are as follows:

Lookup Table Pool and Key Guidance Implementation. We propose the concept of a lookup table pool as a description of the overall lookup table resource. The number of lookup tables in the table pool can far exceed the quantity required for a single encryption or decryption operation. By uniformly and randomly accessing these lookup tables, better security assurances can be provided for white-box cryptography. Meanwhile, the utilization of a lookup table pool facilitates the diversification of algorithm implementations. Furthermore, we introduce the concept of guidance key implementation, wherein a guidance key is uniquely associated with the specific implementation of an algorithm. By combining the lookup table pool with guidance key implementation, we are able to encrypt the program part of white-box cryptographic implementations.

Design of RubikStone. We propose a novel white-box cipher based on a balanced Feistel structure, called RubikStone. By employing a lookup table pool, we can generate various variants ranging from a few kilobytes to infinite sizes. Notably, the size discrepancy

between these variants is minimal, allowing for nearly continuous scalability at the kilobyte level. This feature lays a solid foundation for RubikStone's application across diverse storage capacities. We have conducted a comprehensive security analysis on RubikStone, demonstrating that it serves as a white-box solution capable of resisting all current attack methodologies. This comprehensive analysis underscores the robustness and resilience of RubikStone against a wide array of adversarial techniques.

Key Guidance Application of Cloud-based DRM. Based on RubikStone, we further design a key guidance application specifically tailored for Cloud-based DRM scenarios. For each distinct digital content, there exists a uniquely bound guidance key that directs the implementation of encryption and decryption. As proof of feasibility, we also open-source a prototype implementation[1] for reference.

Provable Bounds on Space Hardness. We give bounds on strong and weak space hardness for RubikStone in known-, chosen- and adaptively chosen-space attack models. In comparison to existing white-box ciphers, our proposed instantiations of RubikStone achieve, for the first time, a strong $(T/4,128)$-space hardness under adaptively chosen-space attacks. Additionally, we evaluate the overall space hardness of key guidance applications employing RubikStone. The results indicate that the key guidance application implemented using the RubikStone instantiation provided in this paper has a minimum overall space hardness of $(0.891T, 128)$, while the scheme based on other instantiations can achieve at least an impressive overall space hardness of $(0.950T, 128)$.

Table Consumption Rate. We first propose a property for evaluating the durability of white-box cryptographic implementations, called table consumption rate. A lower table consumption rate indicates that the white-box cryptographic implementation has better durability. In our evaluation of all existing white-box ciphers, RubikStone-$(64,8,16,2^{26})$ has the lowest table consumption rate, implying that it is currently the cipher with the longest security guarantee before updating the lookup tables among all ciphers. Additionally, RubikStone-$(256,8,12,2^{11})$ achieves a great balance between lookup table size and table consumption rate - it has the lowest table consumption rate among all ciphers within 10 megabytes.

Evaluation of Efficiency We measure the efficiency of RubikStone on a real-world machine. The results show that the smallest instantiation RubikStone-$(128,8,16,2^5)$ exhibits better efficiency than the plain implementation of AES-128. The efficiency of other rubikStone instantiations is also comparable to the white-box implementation of AES-128. Furthermore, for the first time, we conduct comprehensive statistics on the operations of all existing dedicated white-box ciphers. These statistics reveal that RubikStone remains highly competitive in terms of implementation efficiency when compared to other dedicated white-box algorithms.

1.4 Organization

In Section 2, we introduce the models and security notions used in the paper. Then we present the concepts of lookup table pool and key guidance implementation and provide the specification for the white-box cipher RubikStone and several variants in Section 3. In Section 4, we propose a specific key guidance application tailored for cloud-based DRM systems. Subsequently, in Section 5, we conduct a detailed analysis of the security of RubikStone and the key guidance application, evaluating the bounds for their space hardness. In Section 6, we evaluate the table consumption rate and operations of all

[1] https://anonymous.4open.science/r/RubikStone

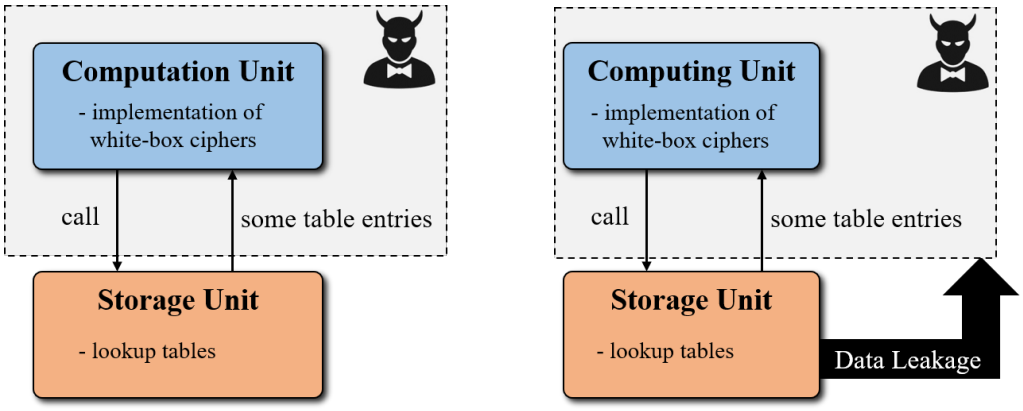
existing white-box ciphers and point out the distinctive features of RubikStone. At the same time, we measure the efficiency of multiple instantiations of RubikStone in real-world environments and compare it with AES-128. Section 7 is dedicated to further discussion of this work. Finally, we offer concluding remarks on our research in Section 8.

2 Preliminaries

2.1 Models

In current white-box cryptographic implementations, the system is divided into a program part responsible for executing computations and lookup tables that aid in these computations. To better characterize the capabilities of the white-box adversary within the framework, we partition it into a computation unit executing the cryptographic implementations and a storage unit containing lookup tables. As mentioned in [HITY22], the two intuitive types of models depicted in Figure 2 have been widely employed in previous works, namely the Only Computation Leakage(OCL) model and the Bounded Arbitrary Leakage(BAL) model:

- In the OCL model, the adversary possesses full control over the computation unit, but can only obtain leakage of the lookup tables by observing and intervening in certain computation processes that involve table lookups.
- In the BAL model, the adversary not only accesses leakage through the computation unit but can also obtain arbitrary leakage through an additional storage leakage channel. However, the leakage channel is limited, with an upper bound on the quantities of data leaked per unit of time.



(a) OCL model (b) BAL model

Figure 2: Models

Indeed, both models impose restrictions on the white-box adversary's ability to access lookup tables, which are also practical in real-world scenarios. On the one hand, even for the white-box adversary, accessing storage is not a trivial task. On the other hand, monitoring memory leaks allows for timely updates to lookup tables when a threshold is reached.

It seems that the only limitation for the white-box adversary lies in the ability to access lookup tables, and therefore all the security of white-box cryptography is predicated on

the white-box adversary's limited access to lookup tables. The determination of security bounds for white-box cryptography under conditions of partial lookup table leakage is crucial for quantifying the security capabilities of white-box cryptography. Three models were proposed in [BIT16] to characterize the white-box adversary's ability to access lookup tables in a more refined way. Each model constrains the adversary to access only a subset of input-output pairs of tables, differing in the selection methodology.

Definition 1. (Known-Space Attack(KSA) [BIT16].) The adversary obtains a certain number of input-output pairs of tables, where the inputs are randomly chosen.

Definition 2. (Chosen-Space Attack(CSA) [BIT16].) The adversary obtains a certain number of input-output pairs of tables, where the inputs are preselected according to the adversary's will.

Definition 3. (Adaptively Chosen-Space Attack(ACSA) [BIT16].) The adversary obtains a certain number of input-output pairs of tables, where each input is chosen according to the adversary's will, and he can choose the next input after obtaining the outputs corresponding to the previous inputs.

2.2 Security Notions

Space Hardness. Biryukov *et al.* proposed the notion of weak white-box security in [BBK14], which was also called incompressibility by De Mulder [Mul14]. The property uses the minimum size of code that the white-box adversary needs to extract from the white-box context for an equivalent key to evaluate the security of white-box ciphers. Based on the property of weak white-box security, Bogdanov and Isobe proposed space hardness to evaluate the difficulty of code lifting attacks in a more quantitative manner, which was widely used in subsequent works [BIT16, FKKM16, CCD+17, KSHI20, KLLM20, KI21, YZDZ23].

Definition 4. (Weak (M, Z)-space hardness [BI15].) An implementation of a block cipher E_K is weakly (M, Z)-space hard if it is computationally difficult to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability of more than 2^{-Z} given any code (table) of size less than M bits.

Definition 5. (Strong (M, Z)-space hardness [BI15].) An implementation of a block cipher E_K is strongly (M, Z)-space hard if it is computationally difficult to obtain a valid plaintext and ciphertext pair with probability of more than 2^{-Z} given any code (table) of size less than M bits.

Table Consumption Rate. White-box cryptography itself serves as a means to provide security over a period of time, and updates are something that white-box cryptography will inevitably need to address. After all, no cipher can provide everlasting security on an untrusted device. Excessive frequency of updates is evidently impractical, and each update may potentially introduce additional security risks to the system. In order to characterize the resilience of white-box ciphers, we define the notion of "table consumption rate" in Definition 6.

Definition 6. (Table consumption rate.) The table consumption rate for a white-box cipher is computationally equal to the ratio of the lookup table size used for encryption per byte to the total lookup table size.

In practical applications of white-box ciphers, more table lookups imply more potential leakages to the adversary. As the adversary gathers sufficient computational leakage information, the lookup tables require updating. Consequently, a lower table consumption rate of white-box ciphers signifies better durability of the cipher.

3 A New White-box Scheme Based on Lookup Table Pool and Key Guidance Implementation

In this section, we will explain the basic idea of the lookup table pool and propose a white-box block cipher called RubikStone based on it. In addition, we present the dynamic implementation concept of key guidance implementation that can uniquely bind an algorithm's implementation with a random number of a specific length. Finally, we provide some specific instantiations of RubikStone.

3.1 Lookup Table Pool

A lookup table stores a mapping relationship from input to output, achieving this by traversing every possible input and storing its corresponding output to ensure that each input can be mapped to the appropriate output. As mentioned in Section 2.1, the security of white-box cryptographic implementation largely comes from lookup tables. Our fundamental goal is to better utilize lookup tables as a resource to enhance the security and practicality of white-box cryptography applications. To this end, we propose the concept of the lookup table pool as a description of the overall lookup table resource, while providing more possibilities for the use of lookup tables, such as key guidance implementation(see Section 3.3).

The basic idea of the lookup table pool is to provide a significantly larger number of lookup tables than required for single encryption, treating them collectively as a resource pool. All the tables are uniformly and randomly accessed by the cryptographic implementation, ensuring that each lookup table contributes equally to the overall security of the white-box cryptographic implementation.

Algorithm 1: AES-128 Based Lookup Table Pool Generation

Input: Input length of tables n_{in}, output length of tables n_{out}, number of tables in the pool s.
Output: A lookup table pool $\mathcal{P}$.

$\mathcal{P} = [\,]$;
for $j \leftarrow 0$ ***to*** $s-1$ **do**
  $k \xleftarrow{\$} \{0,1\}^{128}$;
  for $i \leftarrow 0$ ***to*** $2^{n_{in}}-1$ **do**
    $T_j[i] = Truncate(E_k(i\|0^*), n_{out})$;
    // $Truncate(x,m)$ means truncating the highest m-bit of x
  $\mathcal{P}.append(T_j)$;
return $\mathcal{P}$

Lookup Table Pool Generation. Let $T : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$ be a lookup table that outputs n_{out}-bit $T(x)$ based on an n_{in}-bit input x. Lookup tables are the primary source of confidentiality in white-box cryptography, therefore the adversary should not be able to infer x from $T(x)$. As given in Algorithm 1, E_k is a well-studied cipher (AES-128 is used as an instantiation in Algorithm 1) with a randomly selected key. We can yield the desired lookup table with the specific input and output lengths by padding the input with an all-zero binary value to achieve length extension and truncating the output of E_k, which is also the method used in [BI15].

3.2 RubikStone

Balanced Feistel Network. The Feistel [Fei73] network is a structure widely used in block cipher designs [S+99, RRSY98, SKW+98]. According to whether the size of the

two parts split from each round's input is equal, it can be divided into two categories, i.e. balanced Feistel network and unbalanced Feistel network. As shown in Figure 3, the n-bit state in r-th round X^r is split into two equally sized parts X_a^r and X_b^r. Let K_r be a k-bit key of the r-th round, and $F : \{0,1\}^{n/2} \times \{0,1\}^k \to \{0,1\}^{n/2}$ be an F function. A K_r-based F function is represented as F_{K_r}. The round function of the balanced Feistel network can be formalized as the following equation:

$$X_a^{r+1} \| X_b^{r+1} = (F_{K_r}(X_b^r) \oplus X_a^r) \| X_a^r \tag{1}$$

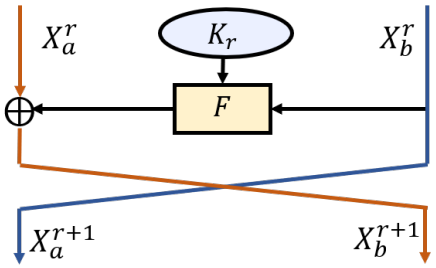


Figure 3: The r-th round of the balanced Feistel network

Design of RubikStone. RubikStone, which is a white-box block cipher using only table lookups and XOR calculations, employs a balanced Feistel network where the key-based F function is replaced by several table lookups. As shown in Figure 4, a plaintext X is encrypted to a ciphertext C by applying R-round transformations. Specifically, the most significant $n/2$ bits of X^r is expressed as $l(= (n/2)/n_{in})$ elements of n_{in} bits, i.e. $X_a^r = \{x_0^r, x_1^r, \ldots, x_{l-1}^r\}$,$x_i^r \in \{0,1\}^{n_{in}}$ for $1 \le r \le R$, where R is the number of total rounds of RubikStone. All the elements $x_i^r (1 \le r \le R, 0 \le i \le l-1)$ are then transformed into 128-bit strings through table lookups and ultimately XOR with X_a^r. The lookup table T_{k_j} corresponding to each element x_i^r is selected from the lookup table pool based on an index k_j, $0 \le k_j \le s-1$, where s is the number of total tables in the lookup table pool. Here j, r, l and i satisfy the following equation:

$$j = (r-1) \times l + i \tag{2}$$

Let RubikStone-(n,n_{in},R,s) denote the different variants of RubikStone with different parameters. For each different guidance key satisfying $K_{guidance} = \{k_0, k_1, \ldots, k_{lR-1}\}$, there is a different instantiation of RubikStone-(n,n_{in},R,s).

3.3 Key Guidance Implementation

In existing white-box ciphers, all the lookup tables are used during a single encryption process and called in a fixed order. However, with the help of the lookup table pool, the process can become more randomized to increase the difficulty of code lifting for the white-box adversary.

In our approach, we assign a unique index to each table in the lookup table pool, allowing the selection and invocation order of lookup tables to be uniquely determined by a sequence of the indexes. Then we can uniquely determine a white-box cryptographic implementation using a randomly generated sequence of specified length. We refer to the randomly generated sequence, which is used to determine the cryptographic implementation, as the "guidance key". Because the fact that only a small subset of tables are used and all the tables are uniformly and randomly called, the white-box adversary not

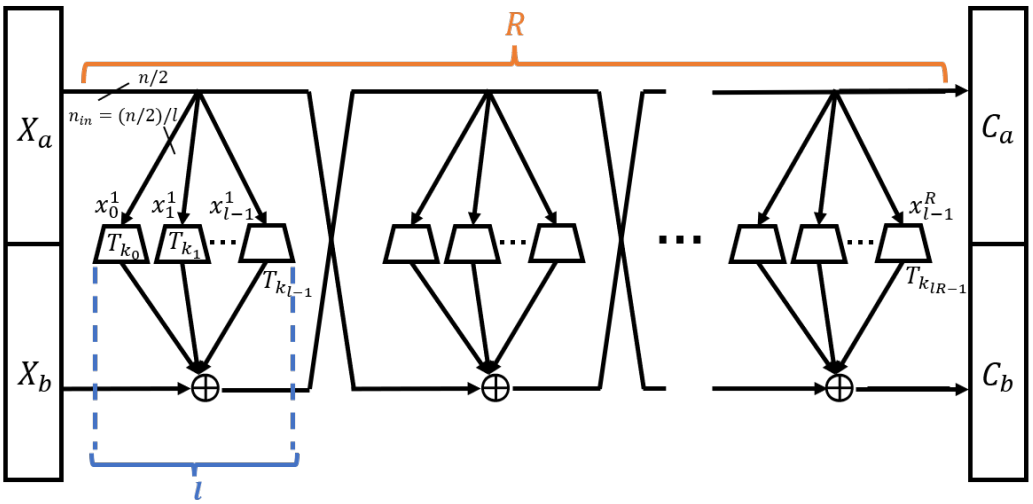


Figure 4: The RubikStone construction,where all the tables are called under the guidance of a guidance key $K_{guidance} = \{k_0, k_1, \ldots, k_{lR-1}\}$.

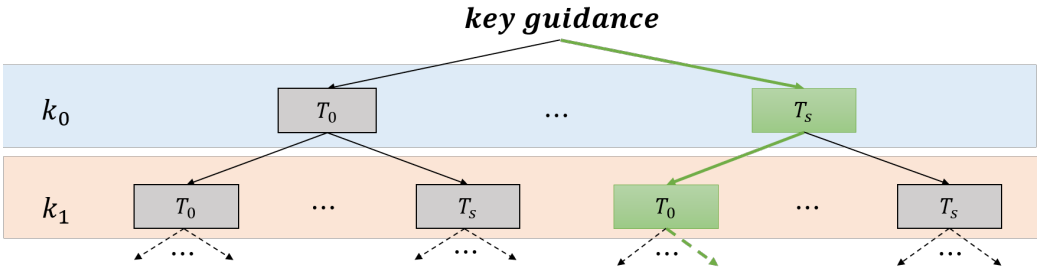


Figure 5: Key guidance tree

only needs to identify which tables from the lookup table pool are utilized during the encryption process but also requires knowledge of the specific order in which these lookup tables are called. The following theorem describes the complexity arising from the key guidance implementation process, which precisely corresponds to the size of the space for the guidance key.

Theorem 1. *For an algorithm RubikStone-(n,n_{in},R,s) with a fixed lookup table pool, the number of distinct algorithm instantiations obtainable using the key guidance implementation is denoted by* $s^{\frac{n}{2 \cdot n_{in}} \cdot R}$.

Proof. As shown in Figure 5, the process of key guidance implementation can be conceptualized as the growth of a multi-branch tree, where different implementations correspond to paths from the root node to the leaf nodes. Each selection of a lookup table corresponds to the random generation of an element in the guidance key. The algorithm RubikStone-(n,n_{in},R,s) possesses s lookup tables, thus there are s possible selections for each lookup table, indicating that each node in the multi-way tree has s child nodes. In total, $\frac{n}{2 \cdot n_{in}} \cdot R$ lookup tables lead to the generation of $s^{\frac{n}{2 \cdot n_{in}} \cdot R}$ leaf nodes, which correspond to $s^{\frac{n}{2 \cdot n_{in}} \cdot R}$ distinct algorithm instantiations. □

3.4 Instantiations

For concrete instantiations, by adjusting the specifications and quantities of lookup tables, we can obtain a series of variants ranging from a few kilobytes to infinite sizes. Meanwhile, the size gradient between different variants is very small, and it can almost continuously increase at the KB level. This facilitates RubikStone's adaptation across a wide range of devices with varying storage capacities. We list several variants of different magnitudes of lookup table sizes with the following specifications:

- **RubikStone-(128,8,16,2^5):** $n = 128, l = 8, R = 16, T : \{0,1\}^8 \to \{0,1\}^{64}, s = 2^5$, total table size$= 2^5 \times 2^8 \times 64\,bits = 64\,KB$, length of guidance key$=80\,Bytes$;
- **RubikStone-(256,8,12,2^{11}):** $n = 256, l = 16, R = 12, T : \{0,1\}^8 \to \{0,1\}^{128}, s = 2^{12}$, total table size$= 2^{11} \times 2^8 \times 128\,bits = 8\,MB$, length of guidance key$=264\,Bytes$;
- **RubikStone-(256,16,24,2^{11}):** $n = 256,\ l = 8,\ R = 24,\ T : \{0,1\}^{16} \to \{0,1\}^{128}$, $s = 2^{11}$, total table size$= 2^{11} \times 2^{16} \times 128\,bits = 2\,GB$, length of guidance key$=264\,Bytes$;
- **RubikStone-(64,8,16,2^{26}):** $n = 64, l = 4, R = 16, T : \{0,1\}^8 \to \{0,1\}^{32}, s = 2^{26}$, total table size$= 2^{26} \times 2^8 \times 32\,bits = 64\,GB$, length of guidance key$=208\,Bytes$.

4 Key Guidance Application of Cloud-based DRM

4.1 Design Rationale

In the context of key guidance implementation, it effectively binds the program part of a white-box cryptographic implementation uniquely to a key. In a cloud-based DRM scenario, the key, utilized for guiding algorithm implementation, may be transmitted alongside the encrypted digital content from cloud servers to client devices.

However, for the white-box adversary in the client devices, the guidance key remains easily accessible. He can extract the guidance key from the client devices and uniquely determine the actual calling order of the lookup tables in the cryptographic implementation. Consequently, the key transmitted to clients should not be the direct key guiding the final implementation. In our research models, the advantage of legitimate users, relative to the white-box adversary desiring to acquire an equivalent function of the cryptographic algorithm, lies in the complete access for the lookup table pool. Therefore, it is natural for us to consider protecting the guidance key using the lookup table pool.

4.2 Specification

Specifically, we propose the scheme shown in Figure 6. The meanings of all the notations are as follows:

- $\boldsymbol{X_p}$: The plaintext of the digital content requested by the client and needs to be encrypted by the cloud server before distribution;
- $\boldsymbol{X_c}$: The ciphertext of the digital content, obtained by encrypting X_p with E_{K_g};
- $\boldsymbol{K_0}$: A random number of equal length to the guidance key, generated by the cloud server;
- $\boldsymbol{K_g}$: The actual guidance key used for digital content encryption, obtained by encrypting K_0 with E_{K_0};
- $\boldsymbol{E_{K_0}}$: The encryption program utilizing K_0 as the guidance key, generated by $\mathcal{C}_E$ with the input K_0;

- $\boldsymbol{E_{K_g}}$: The encryption program utilizing K_g as the guidance key, generated by $\mathcal{C}_E$ with the input K_g;

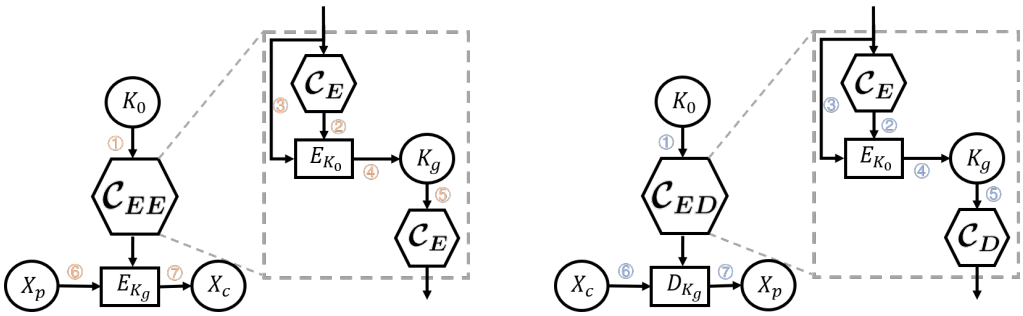
- $\boldsymbol{D_{K_g}}$: The decryption program utilizing K_g as the guidance key, generated by $\mathcal{C}_D$ with the input K_g;

- $\boldsymbol{\mathcal{C}_E}$: An compilation algorithm that takes a guidance key k as input and outputs the encryption program E_k that utilizes k as the guidance key;

- $\boldsymbol{\mathcal{C}_D}$: An compilation algorithm that takes a guidance key k as input and outputs the decryption program E_k that utilizes k as the guidance key;

- $\boldsymbol{\mathcal{C}_{EE}}$: A large compilation program containing two $\mathcal{C}_E$'s, taking a guidance key k as input, outputs an encryption program $E_{k'}$, where $k' = E_k(k)$. In practical applications, the cloud server implements a unified $\mathcal{C}_{EE}$ rather than two separate parts;

- $\boldsymbol{\mathcal{C}_{ED}}$: A large compilation program containing a $\mathcal{C}_E$ and a $\mathcal{C}_D$, taking a guidance key k as input, outputs a decryption program $D_{k'}$, where $k' = E_k(k)$. In practical applications, client devices implements a unified $\mathcal{C}_{ED}$ rather than two separate parts.



(a) Cloud server (b) Client devices

Figure 6: Key guidance application scheme

The calculations of cloud server and client devices are given in Algorithm 2 and Algorithm 3 respectively. As proof of feasibility, we have open-sourced a prototype implementation[2], which provides a comprehensive reference encompassing the generation algorithm for the lookup table pool, an instantiation of the Rubikstone algorithm, as well as a compilation algorithm tailored for generating a specific encryption implementation.

[2]https://anonymous.4open.science/r/RubikStone

Algorithm 2: Encryption Calculations of Key Guidance Application

Input: Plaintext X_p.
Output: Cihertext X_c and a random number K_0.

$K_0 \xleftarrow{\$} \{0,1\}^{len}$; // len denotes the length of the guidance key
$E_{K_0} \leftarrow \mathcal{C}_E(K_0)$;
$\{state_0, state_1, \ldots, state_t\} \leftarrow K_0$; // $t = \lfloor \frac{len}{n} \rfloor - 1$
for $i \leftarrow 0$ ***to*** t **do**
    $g_i = E_{K_0}(state_i)$;
$K_g \leftarrow \{g_0, g_1, \ldots, g_t\}$;
$E_{K_g} \leftarrow \mathcal{C}_E(K_g)$;
$X_c = E_{K_g}(X_p)$;
return X_c, K_0;

Algorithm 3: Decryption Calculations of Key Guidance Application

Input: Cihertext X_c and the random number K_0.
Output: Plaintext X_p.

$E_{K_0} \leftarrow \mathcal{C}_E(K_0)$;
$\{state_0, state_1, \ldots, state_t\} \leftarrow K_0$;
for $i \leftarrow 0$ ***to*** t **do**
    $g_i = E_{K_0}(state_i)$;
$K_g \leftarrow \{g_0, g_1, \ldots, g_t\}$;
$D_{K_g} \leftarrow \mathcal{C}_D(K_g)$;
$X_p = D_{K_g}(X_c)$;
return X_p;

4.3 Application

Practical application of the scheme can be divided into the following three stages:

Initialization:

1. The cloud server selects a specific instantiation of RubikStone and then utilizes Algorithm 1 to generate the lookup table pool;
2. The cloud server informs the client devices of the selected RubikStone instantiation and shares the lookup table pool with the client devices through a secure transmission channel;
3. The cloud server generates the encryption compilation program $\mathcal{C}_E$ based on the lookup table pool;
4. The client devices generate the encryption compilation program $\mathcal{C}_E$ and the decryption compilation program $\mathcal{C}_D$ based on the lookup table pool.

Encryption:

1. When encrypting digital content, the cloud server initially generates a random number K_0 of the same length as the guidance key;
2. Taking K_0 as the input for the encryption compilation program $\mathcal{C}_E$, the cloud server obtains an encryption program E_{K_0} with K_0 serving as the corresponding guidance key;

3. Taking K_0 as the input for the encryption program E_{K_0}, then the cloud server obtains the guidance key actually employed for encrypting the digital content-K_g;
4. Taking K_g as the input for the encryption compilation program $\mathcal{C}_E$, the cloud server obtains the final encryption program E_{K_g} that is truly used for digital content encryption;
5. Taking the digital content plaintext X_p as the input for the encryption program E_{K_g}, the cloud server finally obtains the specific ciphertext X_c;
6. The cloud server sends the ciphertext X_c and the corresponding random number K_0 together to client devices.

Decryption:

1. Client devices possessing the correct lookup table pool can utilize K_0 and the encryption compilation program $\mathcal{C}_E$ to obtain the same guidance key K_g used in the encryption process;
2. Taking K_g as the input for the decryption compilation program $\mathcal{C}_D$, the client devices obtain the final decryption program D_{K_g} which can perform a computation process that is the inverse of the encryption program E_{K_g};
3. Taking the ciphertext X_c as the input for the decryption program D_{K_g}, the client devices finally obtain the true plaintext X_p.

It is noteworthy that in the practical implementation, the encryption procedure of the cloud server is accomplished by a program as a whole, meaning that the ciphertext X_c can be obtained merely by inputting a random number K_0 and the corresponding plaintext X_p. Similarly, the decryption process for client devices operates in the same manner, where the plaintext X_p can be retrieved by inputting the random number K_0 and the ciphertext X_c into a program. The guidance key K_g merely serves as an intermediate parameter during the program's execution and is not output at any stage. Even if a white-box adversary possesses the capability to control and debug this program, significant effort would be required to obtain the guidance key. Furthermore, since the guidance key is randomly generated for each unique plaintext during encryption, the reward for the adversary stealing a single guidance key is low. For a white-box adversary with the ability to control the decryption program, it is far easier to directly invoke the program for decryption to obtain the plaintext than to debug the program to extract one of its intermediate parameters - the guidance key. Thus, for adversaries aiming to transplant the functionality of the decryption program, there exists no more straightforward approach than lifting the entire program itself.

Through the scheme, the cloud server and client devices achieve secret transmission of the decryption program using a randomly generated number K_0, which implies the guidance key K_g in a certain sense. Even if the white-box adversary intercepts K_0, he remains unable to reconstruct the decryption program without access to sufficient lookup tables. This presents an unprecedented approach to mitigate code lifting attacks by using lookup tables to protect the confidentiality of algorithm implementations.

5 Security

5.1 Security in Black-box Context

Here we analyze the security of RubikStone in the black-box context. Our results indicate that RubikStone has good resistance to general purpose cryptographic attacks.

5.1.1 Differential Cryptanalysis

For a function $f(x) : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$, the cardinality of a differential pair (a, b) is defined as the number of input pairs (x_1, x_2) that satisfy the input difference equation $x_1 \oplus x_2 = a$ and the output difference equation $f(x_1) \oplus f(x_2) = b$, denoted by $N(a, b)$. It has been proven in the Theorem 1 of [BI15] that for all non-trivial values of a and b, the probability q_B that $N(a, b)$ is at most B can be lower-bounded by the inequality:

$$q_B > (1 - 2 \cdot \frac{(2^{n_{in}-n_{out}-1})^{B+1}}{(B+1)!})^{2^{n_{in}+n_{out}}} \tag{3}$$

Table 1 shows the lower bound on q_1 for the functions used in all RubikStone instantiations, which implies that the probability of these functions having a differential pair with a maximum occurrence of 1 is very close to 1. Furthermore, we assume the maximum differential probability of the functions used in RubikStone-(128,8,16,2^5), -(256,8,12,2^{11}), -(256,16,24,2^{11}), -(64,8,16,2^{26}) to be 2^{-8}, 2^{-8}, 2^{-16}, 2^{-8}, respectively.

Table 1: Lower-bound on q_1 for the functions used in RubikStone instantiations

Instantiations	**Functions**	**Lower-bound on q_1**
RubikStone-(128,8,16,2^5)	$f : \{0,1\}^8 \to \{0,1\}^{64}$	$1 - 2^{-42}$
RubikStone-(256,8,12,2^{11})	$f : \{0,1\}^8 \to \{0,1\}^{128}$	$1 - 2^{-106}$
RubikStone-(256,16,24,2^{11})	$f : \{0,1\}^{16} \to \{0,1\}^{128}$	$1 - 2^{-82}$
RubikStone-(64,8,16,2^{26})	$f : \{0,1\}^8 \to \{0,1\}^{32}$	$1 - 2^{-10}$

Moreover, based on the lookup table pool generation algorithm in 1, each function is essentially a black-box instance of the AES, which further reinforces our confidence that the functions possess more uniform and randomized differential characteristics.

5.1.2 Linear Cryptanalysis

For a function $f(x) : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$, the correlation of a linear approximation (α, β) is defined by the equation 4, where $\alpha \in \{0,1\}^{n_{in}}$ is an input mask and $\beta \in \{0,1\}^{n_{out}}$ is an output mask.

$$Cor = 2^{-n_{in}} \cdot (|\{x \in \{0,1\}^{n_{in}} | \alpha \cdot x \oplus \beta \cdot f(x) = 0\}| - |\{x \in \{0,1\}^{n_{in}} | \alpha \cdot x \oplus \beta \cdot f(x) = 1\}|) \tag{4}$$

According to the Corollary 4.4 in [DR07], it can be assumed that the linear probability LP of a non-trivial linear approximation over n_{in}-bit to n_{out}-bit functions has mean $\mu(LP) = 2^{-n_{in}}$ and variance $\sigma^2(LP) \approx 2 \times 2^{-2n_{in}}$ when $n_{in} > 5$. Thereforethe linear probability LP of function $f(x)$ is lower than $2^{-n_{in}} + 10\sigma$ with probability $1 - 2^{-148}$. Then we assume the maximum linear probability of the functions used in RubikStone-(128,8,16,2^5), -(256,8,12,2^{11}), -(256,16,24,2^{11}), -(64,8,16,2^{26}) to be 2^{-4}, 2^{-4}, 2^{-12}, 2^{-4}, respectively.

5.2 Security against Key Extraction

In our white-box scheme, there are two specific keys involved: the underlying block cipher's key used to generate lookup tables and the guidance key used for secretly transmitting the dynamical decryption program. As mentioned earlier, all the security of white-box cryptography is based on the adversary's limited access to lookup tables. Since our algorithm for generating lookup table pools is public, if the adversary can extract the keys

from the lookup tables, then the adversary no longer needs to exert effort to obtain the lookup tables. They can simply use the original encryption algorithm to achieve the algorithm's functionality on any device. Similarly, we have employed an encrypted method to transmit the guidance key, thereby secretly transmitting the specific implementation of the algorithm. If the adversary can extract the guidance key, our encryption transmission scheme for the program part will no longer hold. Therefore, in this section, we will analyze the feasibility of the adversary successfully extracting these two types of keys.

5.2.1 Extracting the Keys from Tables

In the white-box context, the adversary can freely observe and intervene the execution process in the computation unit. With such ability, the adversary can easily obtain a large number of pairs of inputs and the corresponding outputs in lookup tables. This implies that adversaries can conduct any form of black-box attack on the lookup tables.

Therefore, the adversary can extract the secret keys from lookup tables in white-box context as long as he can recover the secret key from the underlying cipher for the tables in black-box context. As a corollary, we reduce the security of lookup tables against the key extraction attack in the white-box context to the key recovery problem for the underlying cipher in the black-box context, which is also the reduction method used in some existing dedicated white-box ciphers [BI15, BIT16, FKKM16, CCD+17, KLLM20, KI21].

We utilize AES-128 as an instantiation of the underlying cipher in our lookup table pool generation algorithm, for which no efficient key recovery attack has been proposed so far. Furthermore, in our design of the lookup table pool generation algorithm, a different random key is used for each lookup table generation. Even in the smallest instantiation of RubikStone proposed in Section 3.4, there are 32 lookup tables. This means that the adversary needs to crack at least $4096 (= 32 \times 128)$-bit AES keys in the black-box context if he attempts to gain an advantage in transplanting the functionality of the decryption program by extracting keys from the lookup tables, which is not easier than accumulating all lookup table entries through computation leakage in the white-box context.

5.2.2 Extracting the Guidance Key

In our key guidance application in cloud-based DRM systems described in Section 4, the real guidance key K_g is obtained by encrypting a random number K_0. The encryption algorithm is the same as the one used for encrypting digital content, i.e. the RubikStone algorithm utilizing a lookup table pool. According to our previous analysis, it is not feasible for the adversary to extract the inner keys from the lookup tables. As a corollary, we reduce the security of the guidance key against key extraction attacks to the security of RubikStone itself against code lifting attacks, which will be evaluated in the next section.

Additionally, in our scheme, each different digital content uses a unique guidance key. In the event that the adversary manages to acquire a guidance key, his access is restricted solely to the decryption program for a particular digital content item. Accessing the decryption program for any additional digital content necessitates an equivalent level of effort. This practical approach to digital rights management contributes to enhanced security in real-world scenarios.

5.3 Security against Advanced Side Channel Attacks

5.3.1 Differential Computation Analysis

Differential Computation Analysis (DCA) attack was proposed by Bos *et al.* at CHES 2016 [BHMT16], serving as the software counterpart to differential power analysis (DPA) [KJJ99] attacks employed by the cryptographic hardware community. The main idea of DCA involves utilizing Dynamic Binary Instrumentation (DBI) frameworks such as

Pin [LCM+05] and Valgrind [NS07] to acquire software traces. These traces encompass information like the physical addresses corresponding to memory read/write operations, and stack or register values during program execution, aiding attackers in determining the approximate location of the encryption algorithm within the software implementation and in conducting statistical analyses to extract the secret key. Employing this method, Bos *et al.* successfully extracted keys from various public white-box AES and DES implementations [CEJvO02a, CEJvO02b, LN05, Kar10, XL09] without knowledge of the encodings applied to intermediate results or which cipher operations are implemented by which lookup tables, and without resorting to reverse engineering of the binary files. This establishes DCA as a significant threat to white-box cryptographic implementations.

However, DCA is fundamentally an attack aimed at recovering a specific key. It may be highly effective against the white-box implementations of some existing block ciphers [CEJvO02a, BCD06, Kar10, XL09, LLY14, CEJvO02b, LN05, WP05], but is still ineffective against many dedicated white-box ciphers [BI15, BIT16, FKKM16, CCD+17, KSHI20, KLLM20, KI21, YZDZ23]. This is because these dedicated white-box ciphers are largely based on lookup tables generated from the overall inputs and outputs of a well-studied block cipher. Since the lookup tables are pre-generated, attackers cannot access any side-channel information produced during their creation, limiting them to black-box analysis of the tables. Therefore, when applying DCA to RubikStone, an attacker might recover the guidance key, which could assist in determining the encryption method corresponding to a particular ciphertext. However, without the ability to recover the underlying block cipher keys on which the lookup tables rely, the attacker gains no additional advantage over lifting the lookup tables for decrypting a specific ciphertext.

5.3.2 Algebraic Differential Computation Analysis

Linear decoding analysis (LDA) was first proposed by Goubin *et al.* in [GPRW20]. It was also called algebraic DCA by Biryukov and Udovenko in [BU18], which gradually evolved into an attack method that includes higher-order algebraic structures. This attack is designed to breach masking protection schemes by identifying algebraic combinations of some functions, thereby constructing a predictable sensitive function. With a sufficient number of computational traces, it can effectively pinpoint the location of shares after masking, thus circumventing the combinatorial explosion in complexity. In practical cases, algebraic DCA achieved remarkable success in the WhibOx contest 2017/2019 [PCY+17, GRW20]. On the basis of DCA, Algebraic DCA has improved its attack capability against some mask protection schemes. But algebraic DCA is still an attack method aimed at recovering a specific key, and therefore it cannot pose an effective threat to RubikStone.

5.3.3 Differential Fault Attack

Differential fault attack (DFA) targeting white-box ciphers was proposed by Sanfelix *et al.* in [SMdH15]. It modifies some specific bits by injecting faults into the white-box implementations and then conducts a differential analysis. This attack is also ineffective against RubikStone. Attackers cannot inject faults into the pre-generated lookup tables, and the internals of the underlying cipher are inaccessible. Therefore, all forms of attacks targeting the lookup tables ultimately reduce to black-box attacks against the underlying block cipher AES-128. If the adversary wishes to achieve the ultimate goaldecrypting a randomly drawn ciphertext on any device, the best strategy would be to lift all the lookup tables. The security of RubikStone against code lifting will be analyzed in Section 5.4.

As mentioned in [YZDZ23], side-channel analysis exploits the fact that each lookup table relies only on a small portion of the key, which allows it to exhaustively enumerate all possibilities in segments, compute the correlation of traces, and thus guess the key.

However, in RubikStone, all the lookup tables contain the full 128-bit of the key. Therefore, even if an attacker can fully monitor the memory access patterns of the target key-related lookup tables, the amount of information the attacker must guess is 2^{128}.

5.4 Security against Code Lifting

In our security model (see Section 2.1), the adversary has full observational and control abilities over the computation unit. The only limitation hindering the adversary from transplanting the decryption functionality is their lack of the whole lookup table pool. In fact, under the condition of applying a lookup table pool, the number of lookup tables significantly exceeds the quantity likely to be utilized in a single encryption (or decryption) operation. The adversary must precisely determine the index of a lookup table where an input-output pair resides to effectively apply a specific table entry for attack. This presents an additional difficulty for the adversary in gathering lookup table entries compared to existing white-box schemes. However, the adversary can still gather a significant number of lookup table entries through multiple analyses of the computation unit's execution process, which is just the process known as code lifting. To quantify the resilience of RubikStone against code lifting attacks, in this section, we will evaluate the space hardness of RubikStone under known-, chosen- and adaptively chosen-space attacks, to delineate the feasibility for the adversary to transplant decryption functionality under the condition of acquiring partial lookup table entries.

5.4.1 Weak Space Hardness

According to the discussion in [BIT16], we can obtain the following two theorems.

Theorem 2. *The probability that a randomly drawn plaintext (ciphertext) can be encrypted (decrypted) is upper bounded by* $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$ *given known or chosen space of size* M *from RubikStone-*(n,n_{in},R,s).

Proof. For a randomly drawn plaintext, it can be encrypted to the final ciphertext through $\frac{n}{2\cdot n_{in}}\cdot R$ table lookups. Because the inputs of tables are unpredictable in advance in both known- and chosen-space attacks, the probability for the adversary with a space of size M successfully locating the corresponding lookup table entry during each table lookup is given by $\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}}$, where $s\cdot 2^{n_{in}}\cdot\frac{n}{2}$ is the total size of all the lookup tables. To calculate the correct ciphertext, the adversary needs to possess exactly all the $\frac{n}{2\cdot n_{in}}\cdot R$ relevant table entries, so the probability is upper bounded by $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$. □

Theorem 3. *Given adaptively chosen space of size* M *from RubikStone-*(n,n_{in},R,s)*, the probability that a randomly drawn plaintext (ciphertext) can be encrypted (decrypted) is upper bounded by* $\frac{N}{2^n\cdot s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}+(1-\frac{N}{2^n\cdot s^{(\frac{n}{2\cdot n_{in}}\cdot R)}})\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$*, where* N *satisfies the equation* $N=\lceil(log_{(\frac{2^{n_{in}\cdot s-1}}{2^{n_{in}\cdot s}})}(1-\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}}))/(\frac{n}{2\cdot n_{in}}\cdot R)\rceil$.

Proof. In the process of obtaining input-output pairs, the ACSA adversary can choose an input after obtaining the outputs corresponding to the previous inputs. Exploiting the advantage, the adversary can ensure that a number of plaintexts' corresponding ciphertexts can be obtained with a probability of 100%. In other words, each lookup table entry required in the process of encrypting these plaintexts into the final ciphertexts is present in the part controlled by the adversary. Assuming the number of these plaintexts is N, the following equation can be obtained, where $2^{n_{in}}\cdot s$ is the number of all the entries in the lookup table pool and $\frac{n}{2}$ is the size of each entry.

$$(1-(\frac{2^{n_{in}}\cdot s-1}{2^{n_{in}}\cdot s})^{N\cdot\frac{n}{2\cdot n_{in}}\cdot R})\cdot 2^{n_{in}}\cdot\frac{n}{2}\cdot s=M \tag{5}$$

Namely, the adversary can confidently know the ciphertexts corresponding to $N(= \lceil (log_{(\frac{2^{n_{in}\cdot s-1}}{2^{n_{in}\cdot s}})}(1-\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}}))/(\frac{n}{2\cdot n_{in}}\cdot R)\rceil)$ plaintexts with adaptively chosen space of size M. For a randomly drawn plaintext and a randomly drawn guidance key, the probability that it is included in these N plaintexts is $\frac{N}{2^n\cdot s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}$, where 2^n is the number of all the plaintexts and $s^{(\frac{n}{2\cdot n_{in}}\cdot R)}$ is the number of all the guidance keys. Otherwise, the probability that the adversary can successfully calculate the corresponding ciphertexts given space of size M is upper bounded by $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$ from Theorem 2. The result in Theorem 3 can be obtained by adding the probabilities of the two parts. □

Thus, we obtain weak KSA/CSA-$(M, -log_2((\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}))$-space hardness and weak ACSA-$(M, -log_2(\frac{N}{2^n\cdot s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}+(1-\frac{N}{2^n\cdot s^{(\frac{n}{2\cdot n_{in}}\cdot R)}})\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}))$-space hardness for RubikStone-(n,n_{in},R,s) from Theorem 2 and Theorem 3 respectively.

5.4.2 Strong Space Hardness

The notion of strong space hardness requires that the adversary cannot obtain a valid plaintext-ciphertext pair, which is obviously more strict than weak space hardness where the adversary is not allowed to encrypt(decrypt) a randomly drawn plaintext(ciphertext). According to Theorem 2, a randomly-drawn plaintext or ciphertext can be computed with the probability $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$ or less given known or chosen space of size M. Then for 2^n plaintexts, the expected number of the computable pairs is upper bounded by $2^n\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$.

For strong space hardness in ACSA settings, we obtain the following theorem.

Theorem 4. *Given adaptively chosen space of size M, the probability that the ACSA adversary can obtain a valid plaintext-ciphertext pair is upper bounded by $\frac{N}{s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}+2^n\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$.*

Proof. As mentioned in the proof of Theorem 3, the ACSA adversary can ensure that a number of plaintexts can be computed with a 100% probability to obtain the final ciphertexts by adaptively accessing the lookup table entries. The number of these plaintexts is $N=\lceil (log_{(\frac{2^{n_{in}\cdot s-1}}{2^{n_{in}\cdot s}})}(1-\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}}))/(\frac{n}{2\cdot n_{in}}\cdot R)\rceil$, where M is the size of the given space. For a randomly drawn guidance key, the probability that the computation process for at least one plaintext exactly matches a randomly drawn guidance key is upper bounded by $\frac{N}{s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}$, where $s^{(\frac{n}{2\cdot n_{in}}\cdot R)}$ is the number of all the guidance keys. Otherwise, the probability for an adversary to compute a plaintext-ciphertext pair is the same as in the KSA/CSA settings, which is also $2^n\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$. The probability that the ACSA adversary successfully obtains a plaintext-ciphertext pair can be obtained by adding the two parts. □

As a corollary, we obtain strong KSA/CSA-$(M, -log_2(2^n\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}))$-space hardness and strong ACSA-$(M, -log_2(\frac{N}{s^{(\frac{n}{2\cdot n_{in}}\cdot R)}}+2^n\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}))$-space hardness for RubikStone-(n,n_{in},R,s).

Let T denote the total size of all the lookup tables, i.e. $T=s\cdot 2^{n_{in}}\cdot\frac{n}{2}$. The space hardness of $M=T/4$ has received considerable attention from previous works [BI15, BIT16, CCD+17, KLLM20, KI21]. Based on the evaluation of strong/weak space hardness for RubikStone-(n,n_{in},R,s), we have derived the results shown in Table 2.

Table 2: Space hardness of RubikStone instantiations

Instantiations	Weak Space Hardness		Strong Space Hardness	
	KSA/CSA	ACSA	KSA/CSA	ACSA
RubikStone-(128,8,16,2^5)	$(T/4, 256)$	$(T/4, 256)$	$(T/4, 128)$	$(T/4, 128)$
RubikStone-(256,8,12,2^{11})	$(T/4, 384)$	$(T/4, 384)$	$(T/4, 128)$	$(T/4, 128)$
RubikStone-(256,16,24,2^{11})	$(T/4, 384)$	$(T/4, 384)$	$(T/4, 128)$	$(T/4, 128)$
RubikStone-(64,8,16,2^{26})	$(T/4, 128)$	$(T/4, 128)$	$(T/8, 128)$	$(T/8, 128)$

We can see that the ACSA-space hardness of RubikStone is comparable to its KSA/CSA-space hardness. Compared to previous works [BI15, BIT16] where the ACSA-space hardness is significantly lower than the KSA/CSA-space hardness, this represents a substantial improvement in security. The improvement is made possible by the sufficiently large guidance key space. For any adversary based on past lookup table accumulation, the specific calculations required for encrypting a randomly drawn plaintext are unknown, which effectively reduces the advantages of ACSA adversaries in obtaining plaintext-ciphertext pairs through adaptively accessing lookup table entries.

5.4.3 Overall Space Hardness

In this section, we will evaluate the overall security against code lifting of the key guidance application proposed in Section 4. We have reduced the security of the guidance key against key extraction attacks to the space hardness of RubikStone in Section 5.2. Moreover, the guidance key is uniquely bound to a specific decryption program. To evaluate the security of the decryption program against code lifting attacks, we obtain the following theorem.

Theorem 5. *The probability that a randomly drawn guidance key can be computed is upper bounded by* $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor\cdot\frac{n}{2\cdot n_{in}}\cdot R}$ *given space of size* M *from RubikStone-*(n,n_{in},R,s)*.*

Proof. For a pool with s lookup tables, a $log_2 s$-bit number is needed to represent the index of each table. The number of table lookups required for each encryption or decryption process is $\frac{n}{2\cdot n_{in}}\cdot R$, hence the length of the guidance key of RubikStone-(n,n_{in},R,s) is $log_2 s\cdot\frac{n}{2\cdot n_{in}}\cdot R$. Furthermore, in order to compute this guidance key, $\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor\cdot\frac{n}{2\cdot n_{in}}\cdot R$ table lookups are required. Given space of size M, the probability that the adversary successfully computes each entry is upper bounded by $\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}}$. Thus, the probability to compute a randomly drawn guidance key is upper bounded by $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor\cdot\frac{n}{2\cdot n_{in}}\cdot R}$. □

Based on the evaluation in the previous subsection, the advantage of the ACSA adversary over the KSA/CSA adversary is significantly diminished to a negligible level due to the application of the key guidance implementation mechanism. Then we obtain KSA/CSA/ACSA-$(M, -log_2((\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor\cdot\frac{n}{2\cdot n_{in}}\cdot R})$-space hardness for the decryption program in the key guidance application.

Furthermore, due to the fact that the adversary can only decrypt a specific ciphertext with the correct decryption program, the probability that the adversary can successfully decrypt a randomly drawn ciphertext is upper bounded by $(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor\cdot\frac{n}{2\cdot n_{in}}\cdot R}\cdot(\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{\frac{n}{2\cdot n_{in}}\cdot R}$ given a randomly drawn guidance key. Thus, we obtain the overall $(M, -log_2((\frac{M}{s\cdot 2^{n_{in}}\cdot\frac{n}{2}})^{(\lfloor\frac{log_2 s}{2\cdot n_{in}}\cdot R\rfloor+1)\cdot\frac{n}{2\cdot n_{in}}\cdot R})$-space hardness for the key guidance application.

We further evaluate the overall space hardness of the key guidance application utilizing different instantiations of RubikStone. It can be seen from Table 3 that the key guidance application based on each variant of RubikStone exhibits a high degree of overall space hardness. More surprisingly, the key guidance applications based on RubikStone-(256,8,12,2^{11}), RubikStone-(256,16,24,2^{11}) and RubikStone-(64,8,16,2^{26}) all achieve at least overall $(0.950T, 128)$-space hardness, implying that even if the adversary obtains 95% of the lookup table entries, they still cannot decrypt a randomly drawn ciphertext with a probability exceeding 2^{-128}. This provides sufficiently reliable security for cloud-based DRM.

Table 3: Overall space hardness of key guidance application

Instantiations	**Overall Space Hardness**
RubikStone-(128,8,16,2^5)	$(0.891T, 128)$
RubikStone-(256,8,12,2^{11})	$(0.955T, 128)$
RubikStone-(256,16,24,2^{11})	$(0.955T, 128)$
RubikStone-(64,8,16,2^{26})	$(0.950T, 128)$

6 Performance

6.1 Table Consumption Rate

As mentioned in Section 2.2, the table consumption rate is an effective property for characterizing the resilience of white-box ciphers. A lower table consumption rate means that a table-based white-block cipher has better durability. Table 4 lists all existing white-box ciphers and the relevant parameters based on the table consumption rate ascending sequence. The four instantiations of rubikStone proposed in Section 3.4 all exhibit the lowest table consumption rate for the equal size.

Based on this observation, RubikStone-(64,8,16,2^{26}) exhibits the lowest table consumption rate among all existing white-box ciphers, implying it possesses the highest durability among them. Meanwhile, RubikStone-(64,8,16,2^{26}) has the largest size among all ciphers, which helps to improve the security, but also limits its application scope. In comparison, RubikStone-(256,8,12,2^{11}) achieves a balance here - it has the smallest table consumption rate among the ciphers within 10 megabytes, which can be applied to many memory-limited devices and provides the best durability among existing lightweight white-box ciphers.

6.2 Implementation

In order to measure the efficiency of RubikStone in a real-world environment, we conducted all the experiments with a machine that has Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz and 16GB DDR4 RAM. The processor on the machine has 384KB L1 cache, 1.5MB L2 cache, and 9MB L3 cache, respectively. All code was implemented using the C programming language. To more accurately measure the number of CPU clock cycles, we utilized GCC's inline assembly syntax to invoke the "rdpmc" instruction in the following, which directly reads the Performance Monitoring Counters (PMCs).

```
1  __asm__ volatile ("rdpmc; shlq $32,%%rdx; orq %%rdx,%%rax"
2  : "=a" (result) : "c" (ecx) : "rdx");
```

Table 4: Table consumption rate and relevant parameters of all white-box ciphers

Ciphers	**Parameters**				
	n	$T(Bytes)$[i]	R	T_r[ii]	p[iii]
RubikStone-(64,8,16,2^{26})	64	$2^{26} \times 2^8 \times 32 = 64G$	16	$4 \times 1 \times 32$	2^{-31}
WhiteBlock 32 [FKKM16]	128	$2 \times 2^{32} \times 64 = 64G$	34	$2 \times 1 \times 64$	$2^{-30.91}$
SPNbox-32 [BIT16]	128	$1 \times 2^{32} \times 32 = 17.2G$	10	$1 \times 4 \times 32$	$2^{-30.68}$
Yoroi-32 [KI21][iv]	128	$3 \times 2^{32} \times 32 = 48G$	16	$1 \times 4 \times 32$	$2^{-30.19}$
Galaxy-32 [KSHI20]	128	$1 \times 2^{32} \times 32 = 16G$	32	$1 \times 2 \times 32$	2^{-30}
SPACE-(32,128) [BI15]	128	$1 \times 2^{32} \times 96 = 48G$	128	$1 \times 1 \times 96$	2^{-29}
WhiteBlock 28 [FKKM16]	128	$2 \times 2^{28} \times 64 = 4G$	34	$2 \times 1 \times 64$	$2^{-26.91}$
RubikStone-(256,16,24,2^{11})	256	$2^{16} \times 2^{11} \times 128 = 2G$	24	$8 \times 1 \times 128$	$2^{-24.41}$
FPL-(128,20,12,17) [KLLM20]	128	$204 \times 2^{20} \times 64 = 1.59G$	17	$12 \times 1 \times 64$	2^{-24}
FPL-(128,20,12,33) [KLLM20]	128	$396 \times 2^{20} \times 64 = 3.09G$	33	$12 \times 1 \times 64$	2^{-24}
WhiteBlock 24 [FKKM16]	128	$2 \times 2^{24} \times 64 = 256M$	34	$2 \times 1 \times 64$	$2^{-22.91}$
SPNbox-24 [BIT16]	120	$1 \times 2^{24} \times 24 = 50.3M$	10	$1 \times 5 \times 24$	$2^{-22.26}$
SPACE-(24,128) [BI15]	128	$1 \times 2^{24} \times 104 = 208M$	128	$1 \times 1 \times 104$	2^{-21}
FPL-(128,16,16,17) [KLLM20]	128	$272 \times 2^{16} \times 64 = 136M$	17	$16 \times 1 \times 64$	2^{-20}
FPL-(128,16,16,33) [KLLM20]	128	$528 \times 2^{16} \times 64 = 264M$	33	$16 \times 1 \times 64$	2^{-20}
WEM-128 [CCD+17][v]	128	$104 \times 2^{16} \times 16 = 13M$	12	$8 \times 1 \times 16$	2^{-20}
WhiteBlock 20 [FKKM16]	128	$3 \times 2^{20} \times 64 = 24M$	23	$3 \times 1 \times 64$	$2^{-19.48}$
FPL-(64,16,8,17) [KLLM20]	64	$136 \times 2^{16} \times 32 = 34M$	17	$8 \times 1 \times 32$	2^{-19}
FPL-(64,16,8,33) [KLLM20]	64	$264 \times 2^{16} \times 32 = 66M$	33	$8 \times 1 \times 32$	2^{-19}
FPL-(64,16,16,17) [KLLM20]	64	$272 \times 2^{16} \times 32 = 68M$	17	$16 \times 1 \times 32$	2^{-19}
RubikStone-(256,8,12,2^{11})	256	$2^{11} \times 2^8 \times 128 = 8M$	12	$16 \times 1 \times 128$	$2^{-16.42}$
FPL-(128,12,20,17) [KLLM20]	128	$340 \times 2^{12} \times 64 = 10.63M$	17	$20 \times 1 \times 64$	2^{-16}
FPL-(128,12,20,33) [KLLM20]	128	$660 \times 2^{12} \times 64 = 20.63M$	33	$20 \times 1 \times 64$	2^{-16}
WhiteBlock 16 [FKKM16]	128	$4 \times 2^{16} \times 64 = 2M$	18	$4 \times 1 \times 64$	$2^{-15.83}$
Yoroi-16 [KI21]	128	$3 \times 2^{16} \times 16 = 384K$	8	$1 \times 8 \times 16$	$2^{-14.41}$
SPNbox-16 [BIT16]	128	$1 \times 2^{16} \times 16 = 132K$	10	$1 \times 8 \times 16$	$2^{-13.7}$
WAS [YZDZ23]	128	$1 \times 2^{16} \times 16 = 128K$	10	$1 \times 8 \times 16$	$2^{-13.68}$
SPACE-(16,128) [BI15]	128	$1 \times 2^{16} \times 112 = 896.5K$	128	$1 \times 1 \times 112$	2^{-13}
Galaxy-16 [KSHI20]	128	$1 \times 2^{16} \times 16 = 128K$	40	$1 \times 4 \times 16$	$2^{-12.68}$
FPL-(64,8,16,9) [KLLM20]	64	$144 \times 2^8 \times 32 = 144K$	9	$16 \times 1 \times 32$	2^{-11}
FPL-(64,8,16,17) [KLLM20]	64	$272 \times 2^8 \times 32 = 272K$	17	$16 \times 1 \times 32$	2^{-11}
FPL-(64,8,16,33) [KLLM20]	64	$528 \times 2^8 \times 32 = 528K$	33	$16 \times 1 \times 32$	2^{-11}
RubikStone-(128,8,16,2^5)	128	$2^5 \times 2^8 \times 64 = 64K$	16	$8 \times 1 \times 64$	2^{-10}
SPNbox-8 [BIT16]	128	$1 \times 2^8 \times 8 = 256$	10	$1 \times 16 \times 8$	$2^{-4.68}$
Galaxy-8 [KSHI20]	128	$1 \times 2^8 \times 8 = 256$	25	$1 \times 8 \times 8$	$2^{-4.36}$
SPACE-(8,300) [BI15]	128	$1 \times 2^8 \times 120 = 3.75K$	300	$1 \times 1 \times 120$	$2^{-3.77}$

[i] T denotes the total size of all the lookup tables, which was presented in the form of $n \times e \times b$, where n is the number of tables, e is the number of entries of a table, b is the length of an entry.

[ii] T_r denotes the size of the entries used in one round,which was presented in the form of $m \times e \times b$, where m is the number of tables involved in a round, e is the number of entries got from a table, b is the length of an entry.

[iii] p denotes the able consumption rate of each cipher.

[iv] Yoroi's tables are not uniformly used.

[v] WEM-128 has another table lookup after 12 rounds.

Table 5: Evaluation of encryption efficiency which is given in cycle per byte

Algorithm	Table Size	Efficiency (cycle/byte)
AES-128(Black-box)	-	539
AES-128(CEJO)[CEJvO02a]	752 KB	4027
RubikStone-(128,8,16,2^5)	64 KB	469
RubikStone-(256,8,12,2^{11})	8 MB	2786
RubikStone-(256,16,24,2^{11})	2 GB	5751
RubikStone-(64,8,16,2^{26})	64 GB	7312*

* To better focus on the performance of the algorithm itself, the value was obtained after subtracting the overhead of HDD reads.

The experiment results are shown in Table 5. For comparison, we measured the efficiency of AES-128 and the white-box implementation of AES-128 using the CEJO architecture [CEJvO02a] under the same conditions. RubikStone-(128,8,16,2^5) even exhibits better efficiency than the plain implementation of AES-128. The efficiency of other RubikStone instantiations is also comparable to the CEJO white-box implementation of AES-128, despite their lookup tables being orders of magnitude larger.

It is worth noting that our measurement results appear to be significantly larger than those reported in [KSHI20, KI21]. In fact, despite using the same efficiency evaluation metrics, they cannot be directly compared. This is not only due to performance differences between the devices used, but also because both articles aim for better implementation efficiency and have made numerous optimizations to achieve this, such as programming with instruction sets like AES-NI and SSE. However, this paper focuses on discussing the algorithm itself and the security guarantees during its use. The limits of implementation efficiency are not within our concern, so our implementations are measured without optimization. Nevertheless, by comparing implementation efficiency with AES black-box and white-box implementations under the same conditions, the potential of RubikStone for efficient implementation can still be demonstrated.

Since several other dedicated white-box ciphers do not have open-source code, we did not directly compare our implementation with them. However, we have conducted a detailed statistical analysis of the operations used in all existing dedicated white-box cryptographic schemes and discussed them in the following subsection.

6.3 Statistical Analysis of the Operations

The security of white-box cryptographic implementation based on lookup tables is mainly guaranteed from two aspects. On the one hand, the size of the lookup tables should be as large as possible to slow down the adversary's time to acquire sufficient table entries, providing security for a period of time. On the other hand, the program part should make enough calls to the lookup tables during execution to ensure that even if the adversary obtains part of the lookup tables, he cannot effectively complete the algorithm's function with a high probability. The two aspects inevitably lead to an increase in storage requirements and performance consumption. Therefore, making a reasonable trade-off between security and these two factors is also a focus of white-box cryptography research.

Table 6 lists the number of different operations required to encrypt each byte for all existing white-box ciphers. From it, we can see that RubikStone uses fewer operations compared to most other ciphers. It means that RubikStone is still competitive in terms of execution efficiency despite providing stronger security - strong space hardness under the ACSA model. While RubikStone employs the key guidance application scheme to ensure

this security, the initial computations used to calculate the guidance key and generate the decryption program are one-time operations. As the length of the message increases, the computational overhead of this portion becomes negligible. Furthermore, the simpler arithmetic operations involved also hold promise for RubikStone to potentially achieve better software and hardware implementation efficiency compared to other ciphers.

Meanwhile, table 6 also suggests an important issue to us: among different instantiations of the same white-box scheme, the computational operations required to encrypt one byte do not actually vary significantly. The reason why their implementation efficiencies can differ a lot is that larger lookup tables lead to a significant decrease in the cache hit rate, which is inevitable. This indicates that system cache management will be a significant bottleneck for algorithms such as dedicated white-box cryptographic schemes that rely on large-scale lookup tables. However, the lookup table pool strategy proposed in this paper does not further deteriorate the cache hit rate during specific implementation. This is because the so-called multiple different lookup tables are merely a logical organizational relationship. A well-designed algorithm should invoke all lookup table entries as evenly and randomly as possible. Therefore, whether it is a single lookup table or a pool of multiple lookup tables, the primary factors affecting the cache hit rate are their overall size and the total number of lookup table entries.

7 Further Discussion

On Black-box Implementation. In some previous works [BI15, BIT16, KI21], there was simultaneous discussion of the black-box implementations of the white-box ciphers they proposed. However, in this paper, our focus is on the better security of white-box implementations brought by an effective scheme of using lookup tables, and thus we do not delve deeper into black-box implementations. It can be anticipated that when white-box implementations employ a lookup table pool, corresponding black-box implementations would need to manage the keys contained within each lookup table in the pool simultaneously, resulting in a significant increase in the overall key length. This, while increasing computational overhead, also substantially enhances the security of the black-box implementation.

On Lookup Table Pool Updating. In the key guidance application of cloud-based DRM, due to its inherent high overall space hardness, there is no need to update all lookup tables in the pool at once. For instance, the key guidance application based on RubikStone-$(64,8,16,2^{26})$ can offer an overall $(0.950T, 128)$-space hardness, requiring only periodic updates about 5% of the lookup tables to maintain the security of the white-box cryptographic implementation within a reliable range. Such a lookup table updating mechanism can effectively enhance the practicality of white-box cryptographic implementations.

On Lookup Table Utilization Scheme. In this paper, the strong space hardness and the wide range of instantiation sizes we present are closely related to the scheme of using a lookup table pool and key guidance implementation. A possible consideration holds that the scheme could potentially be realized without resorting to lookup table pools. It seems that in scenarios where tables are employed for single encryption, by altering the tables in each round under the guidance of a key, the viability of this approach remains intact. However, as previously mentioned, a pivotal source of security in white-box cryptography designs stems from the utilization of large lookup tables to mitigate code lifting. Should the guidance key be employed to dynamically direct the update of lookup tables, an attacker would only need to acquire this guidance key to fully transplant the code, given that the methodology for updating lookup tables must inherently be public. Consequently, the integration of the key guidance implementation and lookup table pool

Table 6: Operations of all white-box ciphers

Ciphers	**Calculations(per byte)**				
	L[i]	XOR[ii]	M[iii]	$\mathcal{A}$[iv]	F[v]
SPACE-(8,300) [BI15]	18.75	2250	-	-	-
SPACE-(16,128) [BI15]	8	896	-	-	-
SPACE-(24,128) [BI15]	8	832	-	-	-
SPACE-(32,128) [BI15]	8	768	-	-	-
SPNbox-8 [BIT16]	10	80	0.63($M_{16\times16}$)	-	-
SPNbox-16 [BIT16]	5	80	0.63($M_{8\times8}$)	-	-
SPNbox-24 [BIT16]	3.33	80	0.67($M_{5\times5}$)	-	-
SPNbox-32 [BIT16]	2.50	80	0.63($M_{4\times4}$)	-	-
WhiteBlock 16[FKKM16]	4.50	288	-	1.13	-
WhiteBlock 20 [FKKM16]	4.31	276	-	1.44	-
WhiteBlock 24 [FKKM16]	4.25	272	-	2.13	-
WhiteBlock 28 [FKKM16]	4.25	272	-	2.13	-
WhiteBlock 32 [FKKM16]	4.25	272	-	2.13	-
WEM-128 [CCD+17]	7.5	-	-	0.38	-
Galaxy-8 [KSHI20]	12.50	100	-	-	-
Galaxy-16 [KSHI20]	5	80	-	-	-
Galaxy-32 [KSHI20]	4	128	-	-	-
FPL-(128,12,20,17) [KLLM20]	21.25	1360	-	-	1.06 ($F_{64\to240}$)
FPL-(128,12,20,33) [KLLM20]	41.25	2640	-	-	2.06 ($F_{64\to240}$)
FPL-(128,16,16,17) [KLLM20]	17	1088	-	-	1.06 ($F_{64\to256}$)
FPL-(128,16,16,33) [KLLM20]	33	2112	-	-	2.06 ($F_{64\to256}$)
FPL-(128,20,12,17) [KLLM20]	12.75	816	-	-	1.06 ($F_{64\to240}$)
FPL-(128,20,12,33) [KLLM20]	24.75	1584	-		2.06 ($F_{64\to240}$)
FPL-(64,8,16,9) [KLLM20]	18	576	-	-	1.13 ($F_{32\to128}$)
FPL-(64,8,16,17) [KLLM20]	34	1088	-	-	2.13 ($F_{32\to128}$)
FPL-(64,8,16,33) [KLLM20]	66	2112	-	-	4.13 ($F_{32\to128}$)
FPL-(64,16,8,17) [KLLM20]	34	544	-	-	2.13 ($F_{32\to128}$)
FPL-(64,16,8,33) [KLLM20]	66	1056	-	-	4.13 ($F_{32\to128}$)
FPL-(64,16,16,17) [KLLM20]	34	1088	-	-	2.13 ($F_{32\to256}$)
Yoroi-16 [KI21]	4	14	1($M_{8\times8}$)	0.06	-
Yoroi-32 [KI21]	4	30	1($M_{4\times4}$)	0.06	-
WAS [YZDZ23]	5	80	0.63($M_{8\times8}$)	-	-
RubikStone-(128,8,16,2^5)	8	512	-	-	-
RubikStone-(256,8,12,2^{11})	6	768	-	-	-
RubikStone-(256,16,24,2^{11})	6	768	-	-	-
RubikStone-(64,8,16,2^{26})	8	256	-	-	-

[i] L represents a table lookup.
[ii] XOR represents a bit XOR.
[iii] $M_{m\times m}$ represents a multiplication operation with a $m \times m$ MDS matrix.
[iv] $\mathcal{A}$ represents a 10-round AES.
[v] $F_{n_{in}\to n_{out}}$ represents a probe function.

strategies outlined in this paper is highly rational and indispensable. At the same time, the scheme can also be broadly applied in other table-based white-box structures. It provides a new direction for white-box cryptography research, namely, enhancing the security and practicality of white-box cryptography through the design of an efficient lookup table utilization scheme. Subsequent work could continue researching in this direction.

On the Instantiations of RubikStone. There are four instantiations of RubikStone presented in the paper and we provide a detailed demonstration of the processes involved in analyzing their security and efficiency. According to the specification of RubikStone, users can modify the parameters of RubikStone to create implementations with varying levels of security capabilities and different sizes of lookup table pools, to suit diverse application scenarios.

On the Distribution of the Lookup Table Pool and the Guidance Key. In our key guidance application, both the lookup table pool and the guidance key are indispensable for decrypting specific digital content. Therefore, how to securely and integrally transmit them is a crucial issue. This issue was not specifically discussed in the previous scheme, but the following analysis can illustrate the rationality of our scheme in practical application.

- The lookup table pool is transmitted once and reused. In practical applications, whether through hardware copying or a special secure channel, one-time secure transmission can always be guaranteed. The reason why this transmission channel is generally not used for data transmission is that its cost is relatively high. Essentially, the one-time transmission of the lookup table pool effectively endows legitimate users with the ability to distinguish themselves from others. Relying on such an ability, legitimate users can transmit data inexpensively without relying on other secure channels.
- Each guidance key for different digital content is unique, which is also one of the important security guarantees of the key guidance application. In our scheme, we reduce the security of the guidance key to the difficulty of adversaries obtaining a large number of lookup table entries, ensuring that obtaining the true guidance key used for plaintext encryption is quite difficult for adversaries. Besides, even if an adversary compromises the integrity of the guidance key during transmission, the worst outcome is that legitimate users cannot obtain valid messages in that particular transmission, but it will not lead to the leakage of the message itself. The cloud server can simply resend once to address this issue.

8 Conclusion

In this paper, we explore a novel white-box scheme leveraging a combination of the lookup table pool and key guidance implementation. This scheme forms the basis for RubikStone, a new white-box cipher capable of generating a diverse range of variants ranging from tens of kilobytes to infinite size. We establish that all RubikStone variants provide robust space hardness against adaptively chosen-space attacks. Furthermore, we present a specialized key guidance application tailored for cloud-based DRM scenarios, achieving at least overall $(0.950T, 128)$-space hardness based on our proposed RubikStone variants. Additionally, we introduce the concept of table consumption rate as a novel property for evaluating the durability of white-box cryptographic implementations. Our evaluation reveals that RubikStone-$(64,8,16,2^{26})$ exhibits the lowest table consumption rate among existing ciphers, while RubikStone-$(256,8,12,2^{11})$ achieves the lowest rate among ciphers within a 10-megabyte range. Finally, we conduct a comprehensive analysis of the computational

components of all existing white-box ciphers, demonstrating that despite offering unprecedented security levels, RubikStone remains highly competitive in terms of computational efficiency compared to other ciphers.

In fact, the paper primarily explores the feasibility of dynamically implementing white-box cryptography and employs lookup tables to ensure its security. Cloud-based DRM represents the initial and most prevalent application scenario for white-box cryptography, and in response, the paper presents a dedicated application scheme. Essentially, the presence of the guidance key heightens the dependency of encryption and decryption applications on the entire lookup table pool, resulting in more frequent and unpredictable calls to table entries. Consequently, attackers are compelled to lift the entire lookup table pool. Similarly, grounded in the utilization of lookup tables, our proposed scheme introduces new possibilities for more secure white-box cryptography applications. However, it is crucial to recognize that no cryptographic scheme can be deemed permanently secure. When conditions permit, it is imperative to implement as many protective measures as necessary to safeguard data security.

References

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

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2014.

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptol. ePrint Arch.*, page 468, 2006.

[BGE04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 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, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel,

editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, 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, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[BMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 991–1008. USENIX Association, 2018.

[BPS17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Sgx-step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*, pages 4:1–4:6. ACM, 2017.

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

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

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[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, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

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

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

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Math. Cryptol.*, 1(3):221–242, 2007.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 159–188, 2016.

[GMQ07] Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of white box DES implementations. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 278–295. Springer, 2007.

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

[GRW20] Louis Goubin, Matthieu Rivain, and Junwei Wang. Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 454–482, 2020.

[HITY22] Akinori Hosoyamada, Takanori Isobe, Yosuke Todo, and Kan Yasuda. A modular approach to the incompressibility of block-cipher-based aeads. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*, volume 13792 of *Lecture Notes in Computer Science*, pages 585–619. Springer, 2022.

[Inc14] Adobe Systems Incorporated. *Adobe Primetime Technical Primer for Operators*. Adobe Systems Incorporated, 2014.

[JBF02] Matthias Jacob, Dan Boneh, and Edward W. Felten. Attacking an obfuscated cipher by injecting faults. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2002.

[Kar10] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

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

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KLLM20] Jihoon Kwon, ByeongHak Lee, Jooyoung Lee, and Dukjae Moon. FPL: white-box secure block cipher using parallel table look-ups. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 106–128. Springer, 2020.

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

[LCM+05] Chi-Keung Luk, Robert S. Cohn, Robert Muth, Harish Patil, Artur Klauser, P. Geoffrey Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim M. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In Vivek Sarkar and Mary W. Hall, editors, *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, pages 190–200. ACM, 2005.

[LLY14] Rui Luo, Xuejia Lai, and Rong You. A new attempt of white-box AES implementation. In *Proceedings IEEE International Conference on Security, Pattern Analysis, and Cybernetics, SPAC 2014, Wuhan, China, October 18-19, 2014*, pages 423–429. IEEE, 2014.

[LN05] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: Improving the security of white-box DES. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA*, pages 679–684. IEEE Computer Society, 2005.

[LPSS16] Hyejoo Lee, Suwan Park, Changho Seo, and Sang-Uk Shin. DRM cloud framework to support heterogeneous digital rights management systems. *Multim. Tools Appl.*, 75(22):14089–14109, 2016.

[LRM+13] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 265–285. Springer, 2013.

[MGH08] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a generic class of white-box implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography,*

15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers, volume 5381 of *Lecture Notes in Computer Science*, pages 414–428. Springer, 2008.

[MIE17] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2017.

[MOO+14] Ciara Moore, Máire O'Neill, Elizabeth O'Sullivan, Yarkin Doröz, and Berk Sunar. Practical homomorphic encryption: A survey. In *IEEE International Symposium on Circuits and Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014*, pages 2792–2795. IEEE, 2014.

[MRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the xiao - lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012.

[Mul14] Yoni De Mulder. *White-Box Cryptography: Analysis of White-Box AES Implementations (White-Box Cryptografie: Analyse van White-Box AES implementaties)*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2014.

[MWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2010.

[NS07] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavy-weight dynamic binary instrumentation. In Jeanne Ferrante and Kathryn S. McKinley, editors, *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007*, pages 89–100. ACM, 2007.

[Pay] EMV-Mobile Payment. Software-based mobile payment security requirements.

[PCY+17] Emmanuel Prouff, Chen-Mou Cheng, Bo-Yin Yang, Thomas Baignères, Matthieu Finiasz, Pascal Paillier, and Matthieu Rivain. Ches 2017 capture the flag challenge-the whibox contest, an ecrypt white-box cryptography competition, 2017.

[RP20] Adrián Ranea and Bart Preneel. On self-equivalence encodings in white-box implementations. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 639–669. Springer, 2020.

[RRSY98] Ronald L Rivest, Matthew JB Robshaw, Ray Sidney, and Yiqun Lisa Yin. The rc6tm block cipher. In *First advanced encryption standard (AES) conference*, page 16, 1998.

[RVP22] Adrián Ranea, Joachim Vandersmissen, and Bart Preneel. Implicit white-box implementations: White-boxing ARX ciphers. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2022.

[S+99] Data Encryption Standard et al. Data encryption standard. *Federal Information Processing Standards Publication*, 112:3, 1999.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. *NIST AES Proposal*, 15(1):23–91, 1998.

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

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 2007.

[WP05] Brecht Wyseur and Bart Preneel. Condensed white-box implementations. In *Proceedings of the 26th Symposium on Information Theory in the Benelux*, pages 296–301. Werkgemeenschap voor Informatie-en Communicatietheorie, 2005.

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, 2009.

[YZDZ23] Yatao Yang, Yuying Zhai, Hui Dong, and Yanshuo Zhang. WAS: improved white-box cryptographic algorithm over AS iteration. *Cybersecur.*, 6(1):56, 2023.