White-Box AES Implementation Revisited

Chung Hun Baek, Jung Hee Cheon, and Hyunsook Hong

Department of Mathematical Sciences, Seoul National University,
1 Gwanak-ro, Gwanak-gu, Seoul 151-747, Korea
{love0116,jhcheon,hongsuk07}@snu.ac.kr

Abstract. White-box cryptography is an obfuscation technique for protecting secret keys in software implementations even if an adversary has full access to the implementation of the encryption algorithm and full control over its execution platforms. This concept was presented by Chow et al. with white-box implementations of DES and AES in 2002. The strategy used in the implementations has become a design principle for subsequent white-box implementations. However, despite its practical importance, progress has not been substantial. In fact, it is repeated that as a proposal for a white-box implementation is reported, an attack of lower complexity is soon announced. This is mainly because most cryptanalytic methods target specific implementations, and there is no general attack tool for white-box cryptography.

In this paper, we present an analytic toolbox on white-box implementations in this design framework and show how to reveal the secret information obfuscated in the implementation using this. For a substitution-linear transformation cipher on n bits with S-boxes on m bits, if m_Q-bit nonlinear encodings are used to obfuscate output values in the implementation, our attack tool can remove the nonlinear encodings with complexity $O(\frac{n}{m_Q}2^{3m_Q})$. We should increase m_Q to obtain higher security, but it yields exponential storage blowing up and so there are limits to increase the security using the nonlinear encoding. If the inverse of the encoded round function F on n bits is given, the affine encoding A can be recovered in $O(\frac{n}{m} \cdot {m_A}^3 2^{3m})$ time using our specialized affine equivalence algorithm, where m_A is the smallest integer p such that A (or its similar matrix obtained by permuting rows and columns) is a block-diagonal matrix with $p \times p$ matrix blocks. According to our toolbox, a white-box implementation in the Chow et al.'s framework has complexity at most $O\left(\min\left\{\frac{2^{2m}}{m} \cdot n^{m+4}, n \log n \cdot 2^{n/2}\right\}\right)$ within reasonable storage, which is much less than 2^n.

To overcome this, we introduce an idea that obfuscates two AES-128 ciphers at once with input/output encoding on 256 bits. To reduce storage, we use a sparse unsplit input encoding. As a result, our white-box AES implementation has up to 110-bit security against our toolbox, close to that of the original cipher. More generally, we may consider a white-box implementation on the concatenation of t ciphertexts to increase security.

Keywords: white-box cryptography, white-box implementation, specialized affine equivalence algorithm, AES, block cipher.

1 Introduction

Traditionally, the security of cryptographic algorithms is studied in the black-box model—the end points are trusted and the attacker only has access to the input/output of the algorithm. Under this model, cryptographic schemes are designed to prevent attackers from obtaining secret information using only the input/output values of algorithm without any knowledge of its internal information. In the real world, however, untrusted hosts may access unapproved contents illegally, malicious software in user devices may access the memory used to execute a cryptographic algorithm, or internal information may be leaked during the process of communication. Actually, many attacks have been proposed, such as side channel attacks [10, 13, 14, 18, 19], which extract secret information by access to the internal states in the implementation of algorithm. The concept of white-box cryptography has been proposed to enhance security of cryptosystems under such hostile environment.

The white-box cryptography is defined as an obfuscation technique which gives a secure software implementation, by Chow et al. in 2002. Its goal is to prevent attackers, who have full access to the implementation, from extracting secret key information. In the past, hardware such as smart cards and trusted platform modules were used to protect internal information. Such hardware is costly and difficult to be replaced by a new one when a flaw is discovered. White-box cryptography is a means of protecting the internal information of the software implementation. Many commercial products can use white-box cryptography. One of the main applications is in the digital rights management. Suppose that some content is in encrypted form, and should only be decrypted by permitted devices. If an adversary obtains the decryption key for the content, she can use it in other devices and distribute illegal copies of the content. White-box cryptography aims to prevent attackers from obtaining the decryption key.

The first proposals to implement cryptographic primitives in white-box cryptography were made by Chow et al., who presented a white-box AES implementation [5] and a white-box DES implementation [6] in 2002. They are based on the basic strategy: the whole cipher is decomposed into round functions and the round functions are represented by summation of lookup tables with small size. Although Chow et al.'s implementations have been broken with complexity 2^{14} for DES [24] and 2^{22} for AES [15], their strategy provided a framework, called "CEJO framework", for designing white-box implementation of using table lookups. Most white-box implementations after Chow et al.'s proposal follow the CEJO framework: Xiao and Lai [25] proposed white-box AES implementation using wider linear encodings than Chow et al.'s. Karroumi [12] modified the algebraic operations in each AES round function using dual representations of the AES cipher and presented a white-box AES implementation. However all of these have been broken in the sense that the secret key can be recovered in the lower complexity than their claimed security when the full lookup tables are given (complexity of 2^{32} and 2^{22}, respectively [17, 15]).

On the other hand, researches for white-box cryptography have been proceeded in various ways: Some security notions for white-box cryptography have been studied in [8, 20, 23]. Independently, Biryukov et al. [2] proposed a new symmetric ASASA-based block cipher with secret S-boxes satisfying white-box security notion, whereas previous works focused on proposing white-box implementation of the existing cipher which is well-known and secure.

As we can see from previous implementations, it is very difficult to design a white-box implementation with a security level similar to the black-box model. Hence, the practical objective of white-box implementations is to increase the complexity of cryptanalysis. All of the implementations mentioned above suffered unpredicted attacks soon after their designs were announced. This is mainly because there are no standard attack tools such as differential cryptanalysis and linear cryptanalysis for block ciphers.

Our Contributions: Throughout this paper, we focus on white-box implementations of SLT ciphers following CEJO framework. Let $E = M \circ S$ be the round function of an SLT cipher on n bits, where M is an invertible linear map and S is a concatenation of S-boxes on m bits with a fixed key. We define the input encoding as $f = A \circ P$, where A is an invertible linear map and P is a concatenation of small nonlinear permutations. If we let g be the input encoding of the next round, then the encoded round function F of E is of the form $F = g^{-1} \circ E \circ f = QBSAP$, where B is an invertible linear map and Q is a concatenation of small nonlinear permutations.

Our contributions are two-folded.

1. We present an **analytic toolbox** for white-box implementations of SLT ciphers in the CEJO framework. Our toolbox consists of several algorithms to recover nonlinear and affine encodings used in this model.
   First, by adopting the Biryukov–Shamir technique [4], we show that the nonlinear part Q can be removed up to an affine transformation in $O\left(\frac{n}{m_Q} 2^{3m_Q}\right)$ when $Q = (Q_1, \cdots, Q_{n/m_Q})$ and each Q_i is a nonlinear bijection on m_Q bits. For example, the nonlinear encoding in the Chow et al.'s implementation can be removed in 2^{18} bit operations, whereas it takes 2^{29} bit operations using Billet et al.'s attack [1]. While Billet et al.'s method is only available when the input size of the S-boxes is the same as the input size of the encodings, ours can be efficiently applied when $m \neq m_Q$.

Second, when $F = B \circ S \circ A$ for affine mappings A, B, it is affine equivalent to S. Hence we can apply the affine equivalence algorithm in [3], which has a complexity of $O(n^3 2^{2n})$. We improve this algorithm for the case where S consists of small S-boxes of size m. According to our specialized affine equivalence algorithm (SAEA), if the F^{-1} oracle is given, we can find A and B in $O\left(\frac{n}{m} \cdot {m_A}^3 2^{3m}\right)$, where m_A is the smallest integer p such that A (or its similar matrix obtained by permuting rows and columns) is a block diagonal matrix with $p \times p$ matrix blocks. In fact, m_A is the minimal block size when considering A as a block diagonal mapping. When F^{-1} oracle is not given, SAEA requires $O\left(\min\left\{\frac{n}{m} \cdot {m_A}^{m+3} \cdot 2^{2m}, n \cdot \log m_A \cdot 2^{m_A/2}\right\}\right)$, including the complexity of inverting F, to recover the affine encodings.

Our attack is universal in the sense that all known implementations based on the CEJO framework are susceptible to them. Furthermore, they could play a role of estimating the security of possible white-box implementation designs.

2. We propose a **new design for a white-box implementation** whose security level is close to that of the original cipher. Most variants of Chow et al.'s implementation [12, 25] attempted to increase the security by introducing new affine encodings. According to our toolbox, however, for any affine encoding the complexity for finding the secret key is upper bounded by the minimum of $O(\frac{2^{2m}}{m} \cdot n^{m+4})$ and $O(n \log n \cdot 2^{n/2})$, which is much lower than 2^n. This provides a negative perspective on secure white-box implementations of SLT ciphers using table lookups.

   Our new approach is to use the encryption of multiple plaintexts: For AES-128, we consider the concatenation of two AES-128 ciphers. Let E be a round function of AES-128 and $F = g^{-1} \circ (E, E) \circ f$ be the encoded round function on 256 bits. Then we can take $m_A = 2n > n$ and hence accomplish higher security, up to 2^{110} for $m_A = 256$ and $m = 8$. This approach can be applied to any SLT cipher with $m_A = tn$ for suitable $t \in \mathbb{Z}$ and then the security level is large up to $\left(\frac{2 \cdot 4^n}{m} n^{m+4}\right) \cdot t^{m+4}$. Therefore, this provides a new approach for the design of a secure white-box implementation, regardless of the block length of the original cipher. One shortcoming of this approach is its large storage requirement. However, this is compensated by the use of special sparse encodings. We give an instance with storage requirements of about 16 MB and 64 MB for a single round when $m_A = 128$ and 256, respectively, in Section 5. Our design does not have a security reduction to well known problems and needs to be scrutinized to get a confidence. However, it is still worthy in that it explains why the previous design trials have been failed and how to overcome this barrier in the current state. We expect our work inspires further research to design a secure white-box implementation.

Outline of the Paper: In Section 2, we introduce the basic strategy of Chow et al.'s white-box implementation, and discuss its extension in previous works. We propose attack tools that can be applied to a white-box implementation in Section 3. An approach to the design of a white-box implementation based on the result of our toolbox is given in Section 4. In Section 5, we present an instance of such a the white-box AES implementation. We conclude the paper in Section 6.

2 Revisiting the Chow et al.'s implementation for the SLT Cipher

In the black-box model, it is assumed that the encryption algorithm is executed in trusted platforms. Hence, an adversary cannot observe the internal behavior of the encryption process, but can only the external values, such as the plaintext/ciphertext of the encryption algorithm. However, these models are theoretical, and the leakage of secret information can occur in practical implementations. In gray-box models, adversaries can access more information about the internal details of the encryption algorithm. This information includes side channel information related to runtime, power consumption, and fault analysis, which can be leaked by partial access.

$$\underbrace{M_{out}\circ E^{(r)}\circ f^{(r)}}_{\text{table}}\circ\underbrace{(f^{(r)})^{-1}\circ E^{(r-1)}\circ f^{(r-1)}}_{\text{table}}\circ\cdots\circ\underbrace{(f^{(2)})^{-1}\circ E^{(1)}\circ f^{(1)}}_{\text{table}}\circ\underbrace{(f^{(1)})^{-1}\circ M_{in}}_{\text{table}}$$

$$= \quad M_{out}\circ E^{(r)}\circ\cdots\circ E^{(2)}\circ E^{(1)}\circ M_{in}$$

Fig. 1: The basic strategy of in the CEJO framework

In the white-box model, however, it is assumed that the adversary has full access to the implementation of the encryption algorithm and full control over its execution platforms. In this context, the main objective of the adversary is to extract the secret key. That is, the purpose of secure white-box implementations is to prevent the encryption key from being revealed even when internal algorithm details are completely visible in the untrusted platform, and the adversary has full access to the execution of the encryption algorithm.

One approach for secure white-box implementation of a block cipher is to give a table of all input/output values of the encryption. In this case, the security of the implementation of an algorithm is equivalent to the security of the encryption in the black-box model, and hence depends on the security of the encryption scheme itself, regardless of implementations. Unfortunately, such an implementation is not practical, because the storage requirements of the table are prohibitive. For example, the size of the input/output table of AES-128 is $2^{128} \times 128 = 2^{102}$ GB. Chow et al. suggested a white-box implementations with an implementable table size for AES [5].

Chow et al.'s implementation In the Chow et al.'s implementation, the basic approach for reducing the table size is to decompose the table into small tables with a composition that composition is equivalent to the original input/output table. The most important factor in the table size is the size of the input affecting each S-box, because the S-boxes cannot be decomposed into smaller parts. In AES, one S-box in a single round is influenced by only 8 input bits, but in more than two rounds, each S-box is influenced by all input bits. Hence, Chow et al. decomposed the whole AES cipher into round functions, and represented these as the composition of small tables whose inputs are those corresponding to each S-box.

Because the round key can be exposed if the input/output values of a single round are provided, the input/output tables of each round must be obfuscated by input/output encoding functions. For equivalence with the original AES, the input encoding of the i-th round is offset by the output encoding of the previous round, as in Fig 1 (where $E^{(i)}$ is the i-th round function, $f^{(i)}$ is an input encoding function of the i-th round, and M_{in}/M_{out} are external input/output encodings for security supplement).

The strategy used in the Chow et al.'s implementation can be summarized as follows:

1. The cipher is decomposed into round functions and the round functions are obfuscated by input/output encodings.
2. Each round function is decomposed into a network of lookup tables whose inputs are those corresponding to each S-box.

This strategy provided a framework for designing white-box implementation of block cipher using table lookups. We call it "CEJO framework".

In Chow et al.'s implementation, the encodings composed of nonlinear mappings and linear mappings are used. To prevent an increase in the size of the input that affects each S-box, Chow et al. used 8-bit encodings whose size are the same as that of the size of the S-boxes. A more precise description of the encoded round function is as follows. Each encoded round function on 128 bits is composed of four parallel subround functions on 32 bits. In the Chow et al.'s implementation, the subround function F on 32 bits has the form $F = QBMSAP$, where P, Q are concatenations of 4-bit nonlinear permutations, A, B are block diagonal linear mappings with block size 8, S is the bytewise operation of S-boxes, and M is the Mixcolumns operation on 32 bits. Note that an AddRoundKey operation can be merged to the nonlinear encoding P. Because the block size of the encodings is 8 which is the same as the input size of the S-boxes, the ShiftRows operation can be omitted in the round function. (Thus, we consider the encoded round function as a concatenation of four parallel subround functions.) Hence, F can be represented by the summation of four 8-bit to 32-bit lookup tables. For the "summation" of these tables, twenty-four 8-bit to 4-bit XOR tables are required additionally.[1]

BGE Attack BGE attack [1] exploited that the input encoding size is the same as that of the S-box in the Chow et al.'s implementation. It consists of three steps. First, they recover the nonlinear parts of the encodings. As the Chow et al.'s implementation only uses input encoding on 8 bits (composition of 8-bit mixing bijection and two 4-bit nonlinear encodings), it is easy to obtain the bijective subfunction of F on 8 bits by fixing three bytes of the input. Using this property, the BGE attack can recover nonlinear parts of the encodings (up to affine) in 2^{24} time. In the second step of the attack, the relations between input/output of the table are found using a property of the Mixcolumns operation. Finally, the round key can be found using the result of the second step. The dominant part of this attack's complexity is in the first step, and the total complexity of recovering a 128-bit AES key is 2^{30}.

Michiels et al.'s Cryptanalysis for SLT cipher The CEJO framework can be applied for designing white-box implementation of any other ciphers, such as a generic class of substitution-linear transformation (SLT) ciphers. Michiels et al. [16] considered the white-box implementation of SLT ciphers based on the CEJO framework and presented the associated cryptanalysis. The SLT cipher defined in [16] is a type of iterated cipher with substitution layers and linear transformation layers. A more precise definition of the SLT cipher is as follows.

Definition 1. *The **SLT cipher** $\mathcal{E}$ is defined as follows. It consists of R rounds for some $R \geq 1$. For each $r = 1, \cdots, R$, the r-th round function $E^{(r)}(x_1, \cdots, x_k)$ is a bijective function on n bits, where $n = k \cdot m$ and x_j is an m-bit value for each j and consists of the following three operations:*

1. ***XOR-ing round key*** *XOR the r-th round key $K^{(r)} = (K_1^{(r)}, \cdots, K_k^{(r)})$ of n bits to the input $(x_1, \cdots, x_k)$. This outputs $y_i = x_i \oplus K_i^{(r)}$ for all $i = 1, \cdots, k$.*
2. ***Substitution*** *Compute $z_i = S_i^{(r)}(y_i)$ for all $i = 1, \cdots, k$, where each $S_i^{(r)}$ is an invertible S-box on m bits in the i-th round.*
3. ***Linear transformation*** *For $z = (z_1, \cdots, z_k)$, compute $M^{(r)}z$ where $M^{(r)}$ is an $n \times n$ invertible matrix over $GF(2)$. This n-bit value is the output of the r-th round function.*

Michiels et al.'s use input encodings whose input size is the same as the input size of the S-boxes, as for the original Chow et al.'s implementation. This means the first step of the BGE attack is available to recover the nonlinear parts of the encodings. However, because Michiels et al.'s setting is not only defined on AES, but on any SLT cipher, the other steps of the BGE attack that use the property of AES are not available. Instead, Michiels et al. transformed the encoded round function into a block diagonal mapping whose block size is the same as that of the S-boxes, and recovered the affine encoding of each block using an affine equivalence algorithm [3]. The reason for transforming the encoded round function into a block diagonal mapping is that the input encodings still have an input size that is the same as that of the S-boxes.

[1] As each output value is transformed by a nonlinear encoding, the output values cannot be added directly. Therefore, we need an "XOR table" to perform decoding-XOR-reencoding.

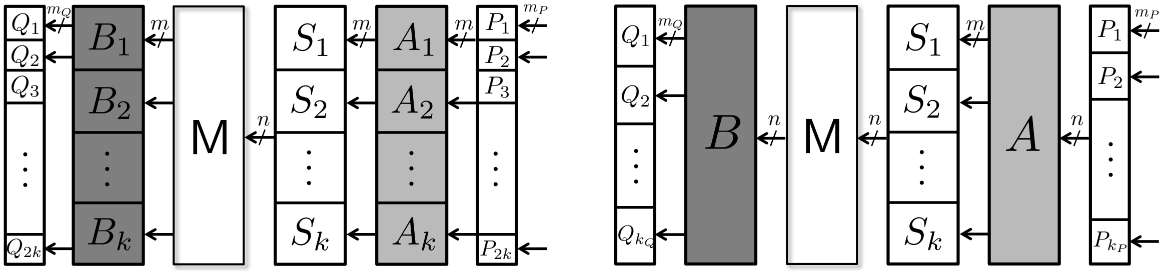

Fig. 2: Original encodings and extended encodings for the CEJO framework

CEJO framework with extended encoding In the Chow et al.'s implementation and Michiels et al.'s modification, the reason for using the input encodings whose input size is the same as that of the S-boxes is to maintain the number of input bits affecting the S-boxes. However, this leads to weakness against BGE attack and Michiels et al.'s cryptanalysis. Therefore, the next step is to extend the form of the encodings for CEJO framework to satisfy both practical and security aspects.

Consider a white-box implementation of an SLT cipher, which follows the CEJO framework. Let $E = M \circ S$ be a round function of the SLT cipher on $n = km$ bits, where M is a linear layer of the SLT cipher and S is a concatenation of S-boxes $S_1, \cdots, S_k$ on m bits.[2] Let f and g be input and output encoding functions, respectively, which are bijections on n bits. Clearly, g is the inverse function of the input encoding function of the next round. Thus, the encoded round function F is defined as $F = g \circ E \circ f$. We first consider an extended form of the encoding at $f = A \circ P$, where A is an invertible linear map on n bits and P is a nonlinear permutation. In the CEJO framework, the table size is mainly determined by the size of input affecting each S-box. Therefore, if A and P are arbitrary bijective linear and nonlinear mappings, respectively, the table size would be huge. Hence, we consider a special mappings that ensure the white-box implementation with reasonable size.

Let $A = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix}$, where A_j is the j-th horizontal strip of size $m \times n$, and let P be a concatenation of nonlinear bijective encodings $P_1, \cdots, P_{k_P}$ on m_P bits, where $n = k_P \cdot m_P$. The output of $f_j = A_j \circ P$ is the input of S_j, and hence the net input[3] size of f_j determines the table size related to S_j. The net input size of f_j is related to the net input size of A_j, and the net input size of A_j is the number of nonzero columns in A_j. Therefore, if we can ensure a small number of nonzero columns in A_j, the table size will be small. Furthermore, since these net input bits are affected by the corresponding P_{j_t}, we should aim for few P_{j_t}, each with a small number of input bits. Therefore, P should be a concatenation of small nonlinear permutations, and A should be an invertible linear map, where each A_j has a small number of nonzero columns.

Since the output encoding is the inverse of the input encoding of the next round, we can write the output encoding g as $g = Q \circ B$, where Q is a concatenation of small nonlinear permutations and B is an invertible linear map. Thus, the encoded round function is of the form $F = QBMSAP$. For simplicity, we write BM as B, because B, M are invertible linear maps, and M is known, *i.e.,* we let $F = QBSAP$.

We may consider the case $f = P \circ A$, where the encoded round function is of the form $F = BQMSPA$. In this case, since the $n \times n$ linear map B follows the Q layer, the XOR tables should decode the encoding BQ, rather than Q. This will make the size of XOR tables huge, and hence we must decompose F into two parts after the Q layer. That is, we let $F = G \circ H$ and make input/output

[2] For a fixed key, the adding key operation can be merged with the nonlinear permutation or the S-box.

[3] The net input of a function is the part that really affects the output of the function.

tables of G and H, where $G = B \circ Q_1$ and $H = Q_2MSPA$ with $Q = Q_1 \circ Q_2$. However, if we combine H with G from the previous round, the function is of the form $Q'MSP'$, because the linear mappings A and B (from the previous round) will be canceled out. Since this is covered by the case $f = A \circ P$, we do not consider it further in this paper. Similarly, for a composition of more than two encodings, we just consider the case $f = A \circ P$ as a generalized form of the encoding.

Remark 1. In practice, m_P cannot be much larger than m, because the use of a nonlinear encoding of size m_P induces the use of a $2m_P$-bit to m_P-bit XOR table. Hence, the choice of m_P is limited. However, we need not be restricted to $m_P \mid m$ or $m \mid m_P$. The nonlinear encodings of the P layer do not need to be aligned with the S-boxes because two different f_j can share some input bits and input encodings. For example, let $n = 192$, $m = 8$, $m_P = 6$, and A be a block diagonal linear mapping with block size 8. For all $j = 1, \cdots, 24$, the number of net input bits of f_j is 12. f_1, f_2 share the same input bits corresponding to P_2 and f_2, f_3 share the same input bits corresponding to P_3. The tables related to each S-box are 12-bit to 192-bit tables and the total table size (including XOR tables) for a round is about 4.4 MB.

Notation In the remainder of this paper, we define $E = M \circ S$ to be a round function of the SLT cipher with block size n, where M is a linear mapping on n bits and S is a layer of k S-boxes on m bits. We let $F = QBSAP$ be the encoded round function of E, where P, Q are layers of small nonlinear permutations, A, B are layers of linear mappings on n bits, and each A_j has a small number of nonzero columns (A_j is the j-th $m \times n$ horizontal strip of A). Note that B contains M.

We also define variables for the input size of the mappings. For the encoded round function $F = QBSAP$ on n bits, we let P, Q be layers of k_P nonlinear bijective encodings on m_P bits and k_Q nonlinear bijective encodings on m_Q bits, respectively. Furthermore, if A is a block diagonal map consisting of mixing bijections on each block, then we write m_A to denote the size of the blocks and k_A for the number of blocks (*i.e.*, $n = k \cdot m = k_P \cdot m_P = k_Q \cdot m_Q = k_A \cdot m_A$).

3 General Attack Toolbox for White-Box Implementation

In white-box cryptography, the attacker's objective is to extract the secret key information. Most block ciphers have key schedules, and so cryptanalysis focuses on recovering one round key. In order to extract the secret key, we find the encodings of consecutive two round functions. Using the relation between the output encodings and the input encodings of the consecutive two rounds, the secret key can be extracted efficiently. Therefore, the goal of this section is the extraction of the secret encodings used to obfuscate in the implementation.

We introduce general tools to recover encodings in $F = QBSAP$ as defined in the previous section. We first recover nonlinear parts of encodings up to affine transforms and then we can let $F = B \circ S \circ A$, where A, B are invertible affine maps. Next, we propose attack tools to find A and B in general cases.

3.1 Recovering Nonlinear Encodings

Usually, recovering nonlinear parts of encoding is very difficult, but in white-box implementations it is easier because only small nonlinear encodings are used. Billet et al. [1] presented a method to recover nonlinear parts of the encoding in Chow et al.'s implementation [5] in 2^{3m} steps. Billet et al. applied this method to only the case that the size of encoding blocks is the same as the size of S-boxes, more precisely, $\mathrm{lcm}(m_P, m_A, m_Q) = m$, where lcm means the least common multiple. Actually, in Chow et al.'s implementation [5] the size of the S-boxes and the mixing bijections is 8 and the size of the nonlinear encodings is 4. The BGE attack can be easily extended to the case that $\mathrm{lcm}(m_P, m_A, m_Q)$

divides m by regarding $\frac{m}{m_P}$ encodings in layer P, $\frac{m}{m_A}$ mixing blocks in layer A and $\frac{m}{m_Q}$ encodings in layer Q as a single encoding in the P, A, Q layers, respectively.

How about the case that $\mathrm{lcm}(m_P, m_A, m_Q)$ does not divide m? In this case, also the BGE attack can be applied to the implementation if $\mathrm{lcm}(m_P, m_A, m_Q, m) < n$, by considering $\mathrm{lcm}(m_P, m_A, m_Q, m)$ as the size of encodings in the P, A, Q and S layers. The complexity of this attack is $2^{3\mathrm{lcm}(m_P, m_A, m_Q, m)}$, and no longer depend only m. For example, consider the case that $n = 192, m_P = m_Q = 6$ and $m_A = m = 8$, the BGE attack has complexity 2^{75}. This gives the following theorem which is extended version of the BGE attack.

Theorem 1. *Let $F = QBSAP$ be an encoded round function of white-box implementation as defined in Section 2. If $l = lcm(m_P, m_A, m_Q, m) < n$, then one can recover a nonlinear part Q (up to affine transformation) in time $\frac{n}{l} \cdot 2^{3l}$.*

In this subsection, we introduce a more efficient tool to recover nonlinear parts of encodings for the latter case, which is based on the multiset attack of Biryukov and Shamir [4]. Using this tool, we can recover nonlinear parts of encodings efficiently even if the size of linear mixing bijections is larger than the size of the S-boxes or the layer of the nonlinear encodings is not aligned with the layer of the S-boxes. This is first approach which provides a link between the technique in [4] and cryptanalysis of white-box implementation.

In order to explain this tool, we will use the multiset properties as in [4]. For more general attack, we add a subscript to each property symbol to denote the size of input. For a multiset M of m-bit values ($m > 1$), the multiset properties are defined as follows:

- M has property C_m (constant) if it contains only numbers of a single m-bit value.
- M has property P_m (permutation) if it contains all numbers of the 2^m possible values exactly once.
- M has property E_m (even) if each value occurs an even number of times or does not occur.
- M has property B_m (balanced) if the XOR of all the values is 0^m.

We extend this notation to denote combined properties. First, we define a projection map $\pi_I : \{0,1\}^n \to \{0,1\}^\tau$ by $\pi_I(x_1, \cdots, x_n) = (x_{i_1}, \cdots, x_{i_\tau})$, for index set $I = \{i_1, \cdots, i_\tau\} \subseteq \{1, \cdots, n\}$. We say a multiset M of n-bit values has property $\mathsf{P}_{2m}^k \mathsf{C}_{n-2km}$, if $\pi_{\{2im+1, \cdots, 2im+2m\}}(M)$ has property P_{2m} for each $i = 0, \cdots, k-1$ and $\pi_{\{2km+1, \cdots, n\}}(M)$ has property C_{n-2km}.

Now let us consider how the multiset properties are transformed by an affine mapping, in the following two lemmas. See Appendix A for the proofs.

Lemma 1. *Let $A : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ be an affine mapping and $I = \{i_1, \cdots, i_\tau\} \subseteq \{1, \cdots, n\}$ with $\tau \geq m > 1$. For a multiset M of n-bit values, a multiset $A(M)$ has property P_m or E_m if $\pi_I(M)$ has property P_τ, $\pi_{\{1,\cdots,n\} \setminus I}(M)$ has property $\mathsf{C}_{n-\tau}$.*

Lemma 2. *Let $A : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ be an affine mapping. For a multiset M of n-bit values, the multiset $A(M)$ of m-bit values has property B_m if M has property B_n and the size of M is even.*

Using these lemmas, we obtain the following theorem, a generalized version of the result in [4]. This attack tool which can remove the nonlinearity of encodings is more efficient than Billet et al.'s attack.

For description of the theorem, we provide some definitions. We say a function $f : \mathbb{Z}_2^u \to \mathbb{Z}_2^v$ with $u \geq v$ is balanced if every output occurs 2^{u-v} times. We define $F^{i,\alpha} : \mathbb{Z}_2^{i \cdot m_P} \to \mathbb{Z}_2^n$ as $F^{i,\alpha}(x) := F(\alpha_1, x, \alpha_2)$ where $\alpha = (\alpha_1, \alpha_2)$ and $\alpha_1 \in \mathbb{Z}_2^{t \cdot m_P}, \alpha_2 \in \mathbb{Z}_2^{n-(i+t) \cdot m_P}$ for some $0 \leq t \leq k_P - i$ and $F_j^{i,\alpha} := \pi_j \circ F^{i,\alpha}$, where π_j is a projection onto the j-th block of layer Q. Lastly, we define a set of functions $\Lambda_{i,j} = \{F_j^{i,\alpha} \mid \alpha \in \mathbb{Z}_2^{t \cdot m_P} \times \mathbb{Z}_2^{n-(i+t) \cdot m_P} \text{for some } 0 \leq t \leq k_P - i\}$.

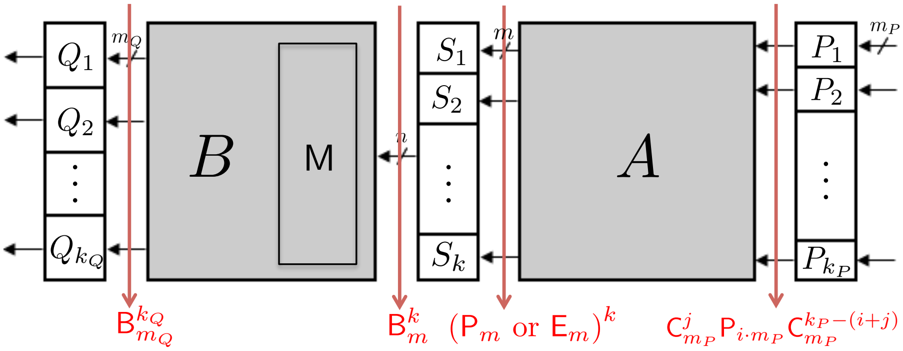

Fig. 3: The relations between multiset properties on QBSAP

Theorem 2. *Let $F = QBSAP$ be a round function of white-box implementations and $\Lambda_{i,j}$ be a set of functions defined above where $i = \lceil \frac{m}{m_P} \rceil$. Assume the probability that a function in $\Lambda_{i,j}$ is not balanced is at least $\mathfrak{p} > 0$ for each j. If $lcm(m_P, m_A, m_Q)$ does not divide m, then one can recover nonlinear part Q (up to affine transformation) using $2^{i\cdot m_P + m_Q} \cdot O(1/\mathfrak{p})$ chosen plaintexts in about $O(k_Q \cdot 2^{3m_Q})$ bit operations.*

Proof. Let α be an $(n - i\cdot m_P)$-bit value. For some t, take M_α to be a set with property $\mathsf{C}^t_{m_P}\mathsf{P}_{(i\cdot m_P)}\mathsf{C}^{k_P-(i+t)}_{m_P}$ such that $\pi_{\{1,\cdots,t\cdot m_P,(i+t)m_P+1,\cdots,n\}}(x) = \alpha$ for each $x \in M_\alpha$.

The property $\mathsf{C}^t_{m_P}\mathsf{P}_{(i\cdot m_P)}\mathsf{C}^{k_P-(i+t)}_{m_P}$ is preserved by the layer P, and thus output multiset has also property $\mathsf{C}^t_{m_P}\mathsf{P}_{(i\cdot m_P)}\mathsf{C}^{k_P-(i+t)}_{m_P}$. Since A can be divided into k affine mappings from n-bit to m-bit and $i \cdot m_P \geq m$, this property is transformed by the layer A into the multiset with property $(\mathsf{P}_m \text{ or } \mathsf{E}_m)^k$ by Lemma 1. Since the property $(\mathsf{P}_m \text{ or } \mathsf{E}_m)^k$ is preserved after layer S, the multiset after layer S has the property B^k_m and this property is equivalent to property B_n. By Lemma 2, the property B_n is transformed by the layer B into the multiset with property $\mathsf{B}^{k_Q}_{m_Q}$ by dividing B into k_Q affine mappings from n-bit to m_Q-bit.

Now, consider the j-th nonlinear bijective encoding Q_j in the layer Q and define $F_j = \pi_j \circ F$, where π_j is a projection onto the j-th block. Then we get a homogeneous equation

$$\sum_{x \in M_\alpha} Q_j^{-1}(F_j(x)) = 0^{m_Q}$$

and since we know the values of $F_j(x)$ for all $x \in M_\alpha$, this equation is a homogeneous equation of the unknowns $Q_j^{-1}(y)$'s for all y through m_Q-bit values, *i.e.*,

$$\sum_{y} c_{\alpha,y} \cdot Q_j^{-1}(y) = 0^{m_Q}$$

where $c_{\alpha,y}$ is the number of $x \in M_\alpha$ satisfying $F_j(x) = y$.

Since the number of unknowns is 2^{m_Q}, we need more than 2^{m_Q} equations. If we use different constant α at the part correspond to property C from $\mathsf{C}^t_{m_P}\mathsf{P}_{(i\cdot m_P)}\mathsf{C}^{k_P-(i+t)}_{m_P}$, we are likely to get a different homogeneous equation of $Q_j^{-1}(y)$'s. By the assumption, we can obtain 2^{m_Q} equations from $2^{m_Q} \cdot O(1/\mathfrak{p})$ multisets, then we can solve the system of equations by Gaussian elimination. We can do this process for all j's and hence we need $O(k_Q 2^{3m_Q})$ bit operations with $2^{i\cdot m_P + m_Q} \cdot O(1/\mathfrak{p})$ chosen plaintexts to recover the layer Q up to affine transformation. □

The BGE attack take $2^{3\mathrm{lcm}(m_P,m_A,m_Q,m)}$ bit operations, but our attack tool only takes 2^{3m_Q} bit operations. Reconsider example for $n = 192, m_P = m_Q = 6$ and $m_A = m = 8$. In this case, our attack tool reduces the complexity from 2^{75} to 2^{23} to remove the layer Q up to affine transformation. Therefore, our attack tool is useful for removing the non-linear encodings in white-box implementation, whether the nonlinear encodings and the S-boxes are aligned or unaligned.

Remark 2. To apply the method in Theorem 2, we require sufficiently many homogeneous equations of the form $\sum_{x \in M_\alpha} Q_j^{-1}(F_j(x)) = 0^{m_Q}$. It is related to the probability $\mathfrak{p}$ because if $F_j^{i,\alpha}$ is balanced, the equation is a trivial equation. By the nonlinearity of S-boxes, if $F_j^{i,\alpha}$ is related to more than 2 S-boxes, $F_j^{i,\alpha}$ is likely to be not balanced. So, we have to take care of choosing multiset of plaintexts, so that $F_j^{i,\alpha}$ is related to more than 2 S-boxes and we note that if $\mathrm{lcm}(m_P, m_A, m_Q)$ does not divide m, $F_j^{i,\alpha}$ is related to more than 2 S-boxes. However, in the case that m_A, m_P and m_Q are equal to m, we can acquire only trivial equation, and hence we cannot use this method. Nevertheless, we can also recover the nonlinear parts of the encodings because the BGE attack can be applied to this case (the BGE attack has same complexity as the method in Theorem 2). Therefore, the toolbox to recover the nonlinearity of the encodings should include both methods with same complexities, considering all the cases.

Actually, we cannot recover Q exactly because we cannot get a system of equation with full rank of 2^{m_Q}, but we can recover Q up to affine transform. Furthermore, we can recover P by attack for the previous round. Therefore, if we assume $k_P = k_Q$, we can recover all nonlinear part of encoding of a round function in $2k_Q \cdot 2^{3m_Q}$ steps.

Applications In Chow et al.'s implementation [5], the input bit size of the linear encodings and the S-boxes is 8 and the size of input/output nonlinear encodings is 4: In our notations, $m = m_A = 8$ and $m_P = m_Q = 4$. Thus, applying the result of Theorem 2, we can recover the nonlinear encodings in $2k_Q \cdot 2^{3m_Q} = 2 \cdot 32 \cdot 2^{3 \cdot 4} = 2^{18}$ time and the complexity is much less than Billet et al.'s [1], 2^{29}. The only thing to be careful about is to take multiset of plaintexts. We have to take multiset of plaintexts, so that the function is related to two S-boxes, for example, the multiset of plaintexts of size 128-bit values that has the property $\mathsf{C}_4\mathsf{P}_8\mathsf{C}_4^{29}$.

3.2 Affine Equivalence Algorithm with Multiple S-boxes

We say that two bijections F and S are *linear/affine equivalent* if there exist linear/affine mappings A, B such that $F = B \circ S \circ A$. *The linear/affine equivalence problem* is to find invertible linear/affine mappings A and B such that $F = B \circ S \circ A$ for given nonlinear bijections F and S.

Biryukov et al. [3] proposed algorithm for solving the linear equivalence problem for arbitrary permutations over $\mathbb{Z}_2^n$ with complexity $O(n^3 2^n)$. For the affine equivalence algorithm, they proposed the concept of the representatives for the linear equivalence classes of permutations and solved the affine equivalence problem in $O(n^3 2^{2n})$ time.

In this subsection, we consider the case that the nonlinear mapping S consists of k invertible S-boxes S_i's which map from $\mathbb{Z}_2^m$ to $\mathbb{Z}_2^m$, where $n = km$, as shown in Fig. 4. The problem may be considered to be a specific case of [3] and so called *the specialized affine equivalence problem.* The following theorem says the problem can be solved more efficiently when compared with the affine equivalence problem.

Theorem 3. *Let F and S be two permutations on n bits where $S = (S_1, \cdots, S_k)$ with nonlinear permutations S_i on m bits for $i = 1, \cdots, k$. Assume that we can easily access the inversion of F. Then, we can find all affine mappings A and B such that $F = B \circ S \circ A$ in time $O(kn^3 2^{3m})$ if they exist.*

Proof. First, we assume that F and S are linear equivalent. Suppose that A and B are invertible linear mappings over $\mathbb{Z}_2^n$ with $F = B \circ S \circ A$. Let us consider A and B^{-1} to be partitioned into k horizontal

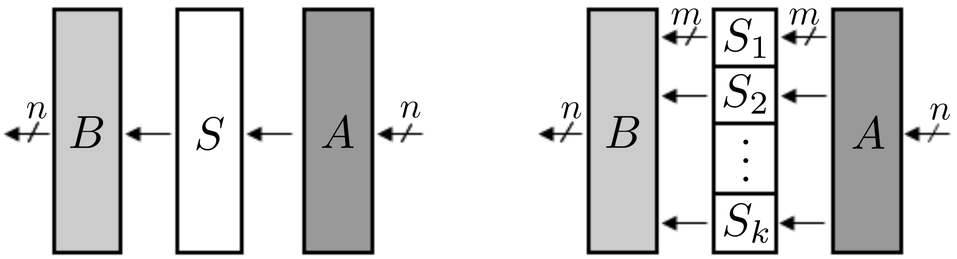

Fig. 4: Affine equivalence problem and specialized affine equivalence problem

strips of size $m \times n$. Denote the i-th strip of A and B^{-1} by A_i and B_i respectively. That is,

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix} \quad \text{and} \quad B^{-1} = \begin{bmatrix} B_1 \\ \vdots \\ B_k \end{bmatrix}. \tag{1}$$

If one can obtain two sets $\{x_1, x_2, \cdots, x_n\}$ and $\{B_i \circ F(x_1), B_i \circ F(x_2), \cdots, B_i \circ F(x_n)\}$ such that $\{F(x_1), F(x_2), \cdots, F(x_n)\}$ is linearly independent, then one can find B_i from

$$B_i = \Big[B_i \circ F(x_1) \; B_i \circ F(x_2) \; \cdots \; B_i \circ F(x_n) \Big] \Big[F(x_1) \; F(x_2) \; \cdots \; F(x_n) \Big]^{-1}, \tag{2}$$

where we consider $B_i \circ F(x_j)$ and $F(x_j)$ as column vectors for $1 \leq j \leq n$. Hence, the main strategy is to find two sets $\{x_1, \cdots, x_n\}$ and $\{B_i \circ F(x_1), \cdots, B_i \circ F(x_n)\}$ such that $\{F(x_1), \cdots, F(x_n)\}$ is linearly independent in order to recover B_i.

Suppose that we have two sets $\{x_1, \cdots, x_\ell\}$ and $\{y_1 = B_i \circ F(x_1), \cdots, y_\ell = B_i \circ F(x_\ell)\}$ such that $\{x_1, \cdots, x_\ell\}$ is linearly independent. For any $x = \sum_{j=1}^{\ell} b_j x_j$ ($b_j \in \{0,1\}$), we can compute $y = B_i \circ F(x)$ from $y_1, \cdots, y_\ell$ by

$$y = S_i \circ A_i(x) = S_i \left(\sum_{j=1}^{\ell} b_j A_i(x_j) \right) = S_i \left(\sum_{j=1}^{\ell} b_j S_i^{-1}(y_j) \right). \tag{3}$$

Since F is a nonlinear bijection, we can obtain another vector x such that $F(x) \notin \mathbb{Z}_2 F(x_1) + \cdots + \mathbb{Z}_\ell F(x_\ell)$ with high probability. (Assuming F is random bijection, at least one of $\{F(x) \mid x \in \mathbb{Z}_2 x_1 + \cdots + \mathbb{Z}_2 x_\ell\}$ does not belong to $\mathbb{Z}_2 F(x_1) + \cdots + \mathbb{Z}_2 F(x_\ell)$ with probability $1 - \left(\frac{2^{d_\ell}}{2^n}\right)^{2^\ell - \ell}$ where $d_\ell = \dim\langle\{F(x_1), \cdots, F(x_\ell)\}\rangle$.)

On the other hand, suppose that we have two sets $\{F(x_1), \cdots, F(x_\ell)\}$ and $\{y_1 = B_i \circ F(x_1), \cdots, y_\ell = B_i \circ F(x_\ell)\}$ such that $\{F(x_1), \cdots, F(x_\ell)\}$ is linearly independent. For any $x' = F^{-1}(\sum_{j=1}^{\ell} b'_j F(x_j))$ ($b'_j \in \{0,1\}$), we can compute $y' = B_i \circ F(x')$ from $y_1, \cdots, y_\ell$ by

$$y' = B_i \circ F \left(F^{-1} \left(\sum_{j=1}^{\ell} b'_j F(x_j) \right) \right) = \sum_{j=1}^{\ell} b'_j B_i \circ F(x_j) = \sum_{j=1}^{\ell} b'_j y_j. \tag{4}$$

Since F^{-1} is a nonlinear bijection then we can obtain a new vector x' such that $x' \notin \mathbb{Z}_2 x_1 + \cdots + \mathbb{Z}_2 x_\ell$ with high probability by assuming F^{-1} is random bijection.

Set $x_0 = 0$, $y_0 = B_i \circ F(x_0)$, $x_1 = F^{-1}(0)$ with $F(x_1) = 0$. Then we have $y_0 = S_i \circ A(x_0) = S_i(0), y_1 := B_i \circ F(x_1) = 0$. We need to make an initial guess $y_2 := B_i \circ F(x_2)$ for some $x_2 \in \{0,1\}^n \backslash \{x_0, x_1\}$ to generate another vectors. Note that x_1, x_2 are linearly independent. If we set $x_3 =$

x_2+x_1, then $F(x_3)$ does not belong to $\mathbb{Z}_2F(0)+\mathbb{Z}_2F(x_2)$ because F is nonlinear and $x_3 \notin \{x_0, x_1, x_2\}$. By repeating above process in the equation (3) and (4) several times, we can successfully obtain n vectors whose F values are linearly independent. For each successful guessing, we get an $m \times n$ linear mapping B_i. We check whether the mapping $S_i^{-1} \circ B_i \circ F$ is linear and reject the incorrect guesses. This process requires n^3 operations for each guessing, and thus the complexity becomes $kn^3 2^m$ to find full matrix B.

Now, let us consider the affine equivalence problem. An affine case is very similar to the linear case. Since an affine mapping is the composition of a linear map and a translation, we can write

$$B_i \circ F(x) + b_i = S_i \left(A_i(x) + a_i\right),$$

for $m \times n$ linear mappings A_i, B_i and the m-bit constant vectors a_i, b_i for $i = 1, \cdots, k$.

For each pair $(a_i, b_i) \in \mathbb{Z}_2^m \times \mathbb{Z}_2^m$, we follow the above process with inputs $F(x)$ and $S_i(x + a_i) + b_i$ and then we can solve the affine equivalence problem. Therefore, the total complexity is $O(kn^3 2^{3m})$ by additionally choosing two m-bit constant vectors. □

We call the algorithm in the Theorem 3 the *specialized affine equivalence algorithm* (SAEA). While the affine equivalence algorithm has the complexity $O(n^3 2^{2n})$ to find the affine mappings A, B, the SAEA has only complexity $O(kn^3 2^{3m})$. This algorithm gives that the dominant parts of the complexities depend on m, not on n even though A and B are random affine mapping over $\mathbb{Z}_2^n$. Therefore, the SAEA is more efficient whenever S is a concatenation of several S-boxes as in the white-box implementation.

Without the oracle of the invese of F The SAEA requires several evaluations of F^{-1} in equation (4) and so we can not apply the SAEA directly when the oracle of inversion of F is not given. In that case, we can use only the property in the equation (3). We have to guess about $\log m_A$ vectors, instead of one vector, to obtain m_A linearly independent vectors, which results in complexity

$$O\left(kn^3 2^{m(\log m_A + 2)}\right) = O\left(kn^{m+3} 2^{2m}\right)$$

for finding the affine encodings. On the other hand, we can use the relation (4) if we evaluate the required inverse value of F in the equation (4). We will discuss about it in Section 4.2.

When A is split We may consider $A \in (\mathbb{Z}_2)^{n\times n}$ as a $\tilde{A} \in (\mathbb{Z}_2^{m\times m})^{k\times k}$ with $n = km$. If $\tilde{A}$ is of form $\left[\begin{array}{c|c|c} * & 0 & * \\ \hline 0 & A^* & 0 \\ \hline * & 0 & * \end{array}\right]$ for some $A^* \in (\mathbb{Z}_2^{m\times m})^{k_0 \times k_0}$ and $k_0 \geq 1$, we say that A is *split*, and *unsplit* otherwise. If A is split, we can recover the encoding that corresponds to A^* with complexity $k_0(k_0 m)^3 2^{3m}$. For more detail, refer to the appendix B.

Applications In Xiao and Lai's implementation [25], they use only the linear mappings for input/output encoding. The input bit size of the input encodings is twice of the input bit size of the S-boxes. By fixing input value on all but 2 bytes as a constant, one can obtain the bijection map F on 16 bits of the following form:

$$F = B \circ (S, S) \circ (\oplus_{K'}, \oplus_{K''}) \circ A$$

where A, B are linear invertible maps on 16 bits. Then F is affine equivalent to (S, S) with linear map B and affine map $(\oplus_{k'}, \oplus_{k''}) \circ A$.

By applying the extended affine equivalence algorithm, we can recover one part of the secret encoding in $\frac{n}{m} n^3 2^{2m} = 2^{29}$ steps for $m = 8$ and $n = 16$. This result is coincident with the result of

Mulder et al. [17], $2n^3 2^n = 2^{29}$ steps. However, our attack tool has some potential advantages over Mulder et al.'s: (1) First, as n is larger than twice of m, *i.e.,* $n = km$ with $k > 2$, our attack has less complexity than Mulder et al.'s. For the case of $m = 8$ and $n = 4m = 32$, the complexity to recover one part of the secret encoding is $\frac{n}{m} n^3 2^{2m} = 2^{33}$ using our attack tool, while $2n^3 2^n = 2^{48}$ using Mulder et al.'s method. (2) One additional advantage of our attack is that if we set A and B to be affine mappings instead of linear mappings to increase security, our tool can be applicable to the scheme while Mulder et al.'s method cannot. For the affine case with same n and m, one can recover a secret encoding in $\frac{n}{m} n^3 2^{3m} = 2^{37}$ using our tool.

4 Approaches for Resisting Our Attack Tools

There have been many proposals for a new white-box implementation, but none appear to require more than 2^{32} complexity to recover the whole secret key. Hence, the urgent subject of white-box cryptography is to design a white-box implementation of higher security with reasonable storage. In this section, we explore why previous white-box implementations can be attacked with low complexity, and investigate several approaches that may overcome this barrier. Note that we consider an SLT-type block cipher of n-bit inputs with m-bit S-boxes.

Recall that m_A is the size of the minimized blocks of block diagonal affine encodings. More precisely, consider the affine encoding of the form $\oplus_a \circ A$, where A is an invertible matrix in $R^{k \times k}$ with $R = \mathbb{Z}_2^{m \times m}$ and a is an n-bit value. Let k_0 be the smallest integer such that there exist two permutation matrices P_1 and $P_2 \in \mathbb{Z}_2^{km \times km}$ satisfying $P_1 A P_2 = \left[\begin{array}{c|c} A_1 & \mathbf{0} \\ \hline \mathbf{0} & A_2 \end{array}\right]$ for some $A_1 \in R^{k_0 \times k_0}$. We define $m_A = k_0 \cdot m$.

4.1 Limitation of White-Box Implementation

Putting the above theorems together, we can summarize our attacks in the following theorem:

Theorem 4. (Main Theorem) *For $i = 1, 2, 3$, $F^{(i)} = Q^{(i)} \circ B^{(i)} \circ S^{(i)} \circ \oplus_{K^{(i)}} \circ A^{(i)} \circ P^{(i)}$, bijections on n bits and $S^{(i)}$, a concatenation of $\frac{n}{m}$ nonlinear bijections on m bits are given where $K^{(i)}$ are secret keys of n bits, $P^{(i)}$ and $Q^{(i)}$ are concatenations of $\frac{n}{m_Q}$ nonlinear bijection on m_Q bits, and $A^{(i)}$ and $B^{(i)}$ are invertible linear mappings on n bits, satisfying $Q^{(i)} \circ P^{(i+1)} = id = B^{(i)} \circ A^{(i+1)}$.*

Then, one can find $K^{(2)}$ in time

$$O\left(3\frac{n}{max(m_Q, m)} \cdot 2^{3max(m_Q,m)} + 2\frac{n}{m} \cdot lcm(m_A, m_Q)^3 2^{3m}\right)$$

with $O\left(\frac{2n\log(lcm(m_A,m_Q))}{lcm(m_A,m_Q)}\right)$ calls of $(F^{(i)})^{-1}$ oracle, or in time

$$O\left(3\frac{n}{max(m_Q, m)} \cdot 2^{3max(m_Q,m)} + 2\frac{n}{m} \cdot lcm(m_A, m_Q)^{m+3} 2^{2m}\right)$$

without using $(F^{(i)})^{-1}$ oracle, where m_A is the size of minimized blocks of $A^{(i)}$'s as difined above.

Proof. Note that $m | m_A$ by definition of m_A. Since $\mathrm{lcm}(m_A, m_Q) | m$ implies $l = \mathrm{lcm}(m_A, m_Q, m) = m$, one can recover $Q^{(i)}$ (up to affine transformation) in time $O(\frac{n}{max(m_Q,m)} \cdot 2^{3max(m_Q,m)})$ by Theorem 1 and Theorem 2 and also can recover $P^{(1)}$ and $P^{(2)}$ from $P^{(1)} = (Q^{(0)})^{-1}$ and $P^{(2)} = (Q^{(1)})^{-1}$.

Now, for $i = 1, 2$, the nonlinear effects of $P^{(i)}$ and $Q^{(i)}$ can be removed in $F^{(i)}$ and hence $F^{(i)}$ can be considered $F^{(i)} = \tilde{B}^{(i)} \circ S^{(i)} \circ \oplus_{K^{(i)}} \circ \tilde{A}^{(i)}$ for some affine mappings $\tilde{A}^{(i)}$ and $\tilde{B}^{(i)}$ on n bits. Note that $\tilde{A}^{(i)}$ can be considered block diagonal affine mappings with block size $l = \mathrm{lcm}(m_A, m_Q)$. Therefore, one can apply SAEA to each block of size l. When SAEA is applied to block of size l, it needs $\log l$ calls of $(F^{(i)})^{-1}$ oracle or to guess about $\log m_A$ vectors, instead of one vector, without using $(F^{(i)})^{-1}$ oracle. It follows that one can recover $\hat{A}^{(i)} = \oplus_{K^{(i)}} \circ \tilde{A}^{(i)}$ and $\tilde{B}^{(i)}$ in time $O(\frac{n}{m} \cdot l^3 2^{3m})$ with $\frac{n}{l} \log l$ calls of $(F^{(i)})^{-1}$ oracle or $O(\frac{n}{m} \cdot l^3 2^{m(\log l+2)}) = O\left(\frac{n}{m} \cdot l^{m+3} 2^{2m}\right)$ without using $(F^{(i)})^{-1}$ oracle.

From the relation between $P^{(2)}$ and $Q^{(1)}$, one can find $K^{(2)}$ by computing $\hat{A}^{(2)} \circ \tilde{B}^{(1)}(\mathbf{0}) = \oplus_{K^{(2)}} \circ A^{(2)} \circ B^{(1)}(0) = K^{(2)}$. □

All of the previous white-box implementations have common features: For $n = 128, m = 8$, (1) they use affine/linear encodings with $m_A \leq 16$, and (2) they do not use nonlinear encodings, or use nonlinear encodings with only $m_Q = 4$. In these case, $\mathrm{lcm}(m_A, m_Q) \leq 16$, and so one can easily compute the inverse. By the result of Theorem 4, all previous implementations can be broken in less than 2^{41} time without using a specific attack.

To increase the complexity, we need to increase $\mathrm{lcm}(m_A, m_Q)$. Increasing m_Q results in large storage requirements for the XOR table; e.g., one XOR table requires 8 for $m_Q = 16$. Another approach is to increase m_A. We may try to increase m_A up to n. When $m_A = n \leq 128$ and $m = 8$, the complexity is at most 2^{50} if the F^{-1} oracle is given, where F is an encoded round function.

4.2 Perspective of White-Box Implementation

In reality, however, the oracle of F^{-1} is not provided, and so we focus on this case. By Theorem 4, when $m_A = n$, the complexity of the SAEA is $O\left(\frac{n}{m} \cdot {m_A}^{m+3} 2^{2m}\right)$, which is a polynomial of m_A with degree $m+3$. On the other hand, we could also use the SAEA by evaluating the required inverse values of F. Using a meet-in-the-middle attack (MITM), one inverse evaluation of F has a time complexity of $O(m_A 2^{m_A/2})$ and memory requirement of $O(m_A 2^{m_A/2})$, which can be reduced to $O(m_A 2^{m_A/4})$ using a dissection-type technique [9] (See Appendix C). Because the SAEA requires about $\log m_A$ evaluations of F^{-1}, its complexity is

$$O\left(\tfrac{n}{m} \cdot m_A^3 2^{3m} + \tfrac{n}{m_A} \cdot \log m_A \cdot m_A \cdot 2^{m_A/2}\right)$$

which is dominated by the complexity of inversion when $m_A > 6m$. Therefore, when $m_A = n = 128$ and $m = 8$, the SAEA complexity is

$$O\left(\min\left\{\tfrac{n}{m} \cdot {m_A}^{m+3} \cdot 2^{2m}, n \cdot \log m_A \cdot 2^{m_A/2}\right\}\right) = 2^{74}.$$

A security level of 74 bits is higher than that of previous implementations [5, 12, 25], but is not sufficient considering the security level of the original cipher. This limitation arises because the complexity is heavily dependent on m_A, but m_A cannot exceed n. We must therefore examine another approach to further increase the security level of the white-box implementation.

Let us consider the case whereby we encrypt messages that are longer than the cipher's block length or multiple messages. We investigate an approach in which multiple plaintexts are simultaneously encrypted by one white-box implementation. Then, we can take m_A larger than n, such as $m_A = 2n, 3n, \cdots$, and hence the security level can be improved over that stated above. For example, the complexity of the SAEA can be large up to 2^{109} when $m_A = 2n, n = 128$, and $m = 8$. However, this approach may lead to large storage requirement. To overcome this problem, we use special "*sparse unsplit*" encodings, as shown in Fig 5. In the next section, we present an instance of this white-box implementation for concatenations of AES-128 using sparse unsplit encodings that are bijections on $m_A = 256$.

5 A Proposal for a White-Box Implementation of the AES Cipher

We propose a white-box implementation for concatenation of two AES using table lookups. The AES consists of 10 rounds and the round function of AES is made of the four steps: SubBytes(SB),

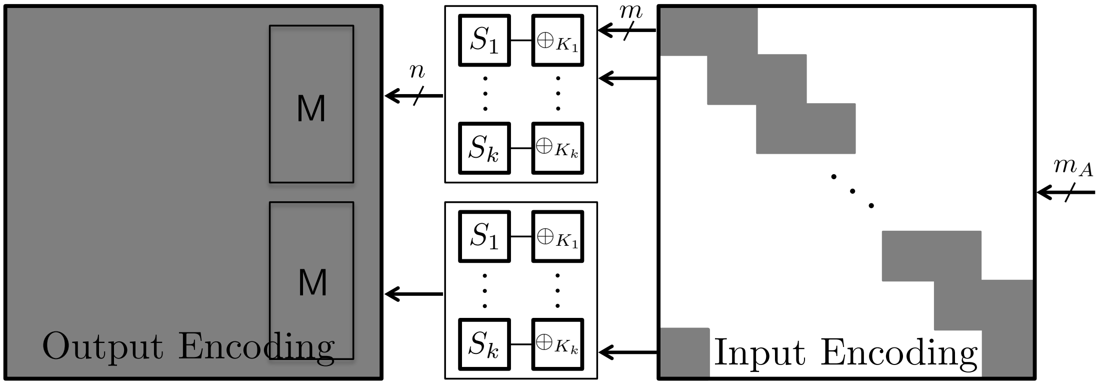

Fig. 5: The hard-to-invert encoded round function with $m_A = 2n$

ShiftRows(SR), MixColumns(MC), and AddRoundKey(ARK). Each step is a bytewise operation on the 16 bytes. For efficiency, we set the round function $\mathrm{AES}^{(r)}$ of AES as follows:

$$\mathrm{AES}^{(r)} = \begin{cases} \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SB} \circ \mathsf{ARK}, & \text{if } r = 1, \cdots, 9 \\ \mathsf{SR} \circ \mathsf{AK} \circ \mathsf{SB} \circ \mathsf{ARK}, & \text{if } r = 10 \end{cases}.$$

Input and Output Encodings We use *sparse unsplit* encodings as input encodings like in equation (5) to reduce the storage: Let $A^r \in \mathbb{Z}_2^{256 \times 256}$ be an invertible linear map such that $A^r_{i,j}$ is the zero matrix for all $(i, j) \neq (i, i), (i, i + 1)$ and $(32, 1)$, where $A^r_{i,j}$ is (i, j)-th block of A^r when A^r is partitioned into 1024 blocks $A^r_{i,j}$ of size 8×8 for $i, j = 1, \cdots, 32$. We define an input encoding $A^{(r)}$ of r-th round of the form $\oplus_{a^r} \circ A^r$ for a 256-bit value $a^r = (a^r_1, \cdots, a^r_{32})$ where a^r_i's are 8-bit values. That is,

$$A^{(r)}(x) = \begin{bmatrix} A^r_{1,1} & A^r_{1,2} & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} \\ \mathbf{0} & A^r_{2,2} & A^r_{2,3} & \mathbf{0} \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \ \ddots & \vdots \\ A^r_{32,1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \cdots & A^r_{32,32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{32} \end{bmatrix} \oplus \begin{bmatrix} a^r_1 \\ a^r_2 \\ \vdots \\ a^r_{32} \end{bmatrix} \tag{5}$$

for $x = (x_1, \cdots, x_{32}) \in \{0, 1\}^{256}$ with 8-bit values x_i's.

For the input encoding $A^{(r+1)}$ of the $(r + 1)$-th round, we define the output encoding $B^{(r)}$ of r-th round by $B^{(r)} = (A^{(r+1)})^{-1} \circ (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR})$, for $r = 1, \cdots, 9$, where $(\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR})$ is a concatenation of $\mathsf{MC} \circ \mathsf{SR}$ on 128 bits. In general, the inversion $(A^r)^{-1}$ has no sparse characteristics despite the sparse structure of A^r. (One can easily check it using Gaussian elimination.) Therefore, the output encoding $B^{(r)}$ has no specific structure. Note that nonlinear encodings of small size are used in the white-box implementation in general and the complexity of removing the nonlinear encodings is not higher than the complexity of finding the affine encodings. We have no consideration for the nonlinear encodings in this section.

Then, the encoded round function $F^{(r)}$ of $\mathrm{AES}^{(r)}$ is defined by $F^{(r)} = (A^{(r+1)})^{-1} \circ (\mathrm{AES}^{(r)}, \mathrm{AES}^{(r)}) \circ A^{(r)} = B^{(r)} \circ (S, \cdots, S) \circ \oplus_{(K^r, K^r)} \circ A^{(r)}$ for $r = 1, \cdots, 9$, where S is the S-box function on 8 bits in the SubBytes step and K^r is the secret key of r-th round on 128 bits in the AddRoundKey step. Since the final round of AES is slightly different from the other rounds, the encoded round function of final round will be discussed later.

Construction of Lookup Tables Let B^r_i be a linear mapping from 8 bits to 256 bits for $i = 1, \cdots, 32$ and b^r be a 256-bit value vector such that $B^{(r)}(x) = [B^r_1 | \cdots | B^r_{32}](x) \oplus b^r$ for any 256-bit value x. Choose 256-bit random value b^r_i for each $i = 1, \cdots, 31$, and let $b^r_{32} := b^r \oplus b^r_1 \oplus \cdots \oplus b^r_{31}$.

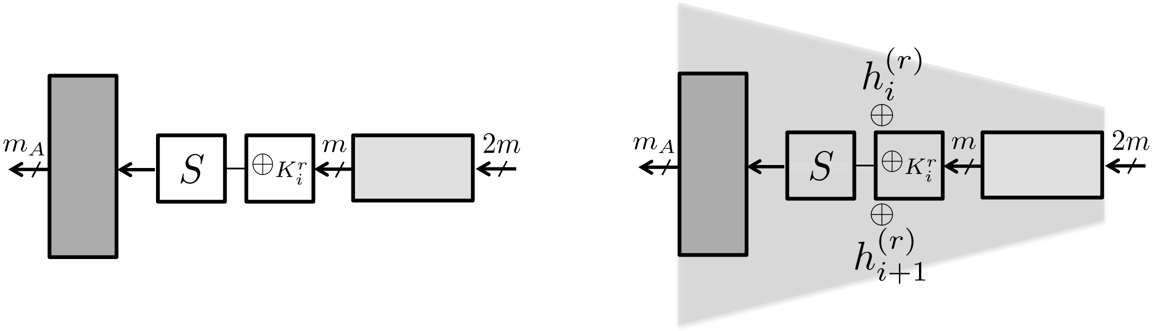

Fig. 6: The lookup tables of $F_i^{(r)}$ and $T_i^{(r)}$

Now, each 16-bit to 256-bit table $F_i^{(r)}$, depicted in Fig 6, is defined as follows:

$$F_i^{(r)} = \begin{cases} \oplus_{b_i^r} \circ B_i^r \circ S \circ \oplus_{K_i^r \oplus a_i^r} \circ (A_{i,i}^r, A_{i,i+1}^r), & \text{if} \quad 1 \leq i \leq 16 \\ \oplus_{b_i^r} \circ B_i^r \circ S \circ \oplus_{K_{i-16}^r \oplus a_i^r} \circ (A_{i,i}^r, A_{i,i+1}^r), & \text{if} \quad 17 \leq i < 32 \\ \oplus_{b_{32}^r} \circ B_{32}^r \circ S \circ \oplus_{K_{16}^r \oplus a_{32}^r} \circ (A_{32,32}^r, A_{32,1}^r), & \text{if} \quad i = 32 \end{cases}$$

where $K^r = (K_1^r, \cdots, K_{16}^r)$ is the r-th round key for $K_i^r \in \{0,1\}^8$. The affine mapping $\oplus_{a_i^r} \circ (A_{i,i}^r, A_{i,i+1}^r)$ from $\mathbb{Z}_2^{16}$to $\mathbb{Z}_2^8$ is inserted before $S \circ \oplus_{K_i^r}$ and the affine mapping $\oplus_{b_i^r} \circ B_i^r$ from $\mathbb{Z}_2^8$ to $\mathbb{Z}_2^{256}$ is inserted after the S-box part. Then the encoded round function $F^{(r)}$ of AES$^{(r)}$ can be expressed as a sum of $F_i^{(r)}$'s:

$$F^{(r)}(x_1, x_2, \cdots, x_{32}) = F_1^{(r)}(x_1, x_2) \oplus F_2^{(r)}(x_2, x_3) \oplus \cdots \oplus F_{32}^{(r)}(x_{32}, x_1)$$

for $r = 1, \cdots, 9$ and 8-bit values x_i's. Therefore, the encoded round function $F^{(r)}$ can be implemented using thirty-two 16-bit to 256-bit lookup tables, instead of implementing a 256-bit to 256-bit table of huge size. However, since $F_i^{(r)}(x, 0) = \oplus_{b_i^r} \circ B_i^r \circ S \circ \oplus_{K_i^r \oplus a_i^r} \circ A_{i,i}^r(x)$ is a 8 bit-to-128 bit function for 8-bit value x, it can be transformed to a bijection on 8 bits by some projection. Then, it is affine equivalent to S, Hence, the affine equivalent algorithm of [3] can be applied to each $F_i^{(r)}(x, 0)$ and it has only 2^{25} complexity to recover affine mappings. To prevent the individual attack, we modify the functions $F_i^{(r)}$'s. We replace $F_i^{(r)}$ by $T_i^{(r)} : \mathbb{Z}_2^{16} \rightarrow \mathbb{Z}_2^{256}$ such that

$$T_i^{(r)}(x, y) = \begin{cases} F_i^{(r)}(x, y) \oplus h_i^{(r)}(x) \oplus h_{i+1}^{(r)}(y), & \text{if } i \neq 32 \\ F_{32}^{(r)}(x, y) \oplus h_{32}^{(r)}(x) \oplus h_1^{(r)}(y), & \text{if } i = 32 \end{cases}$$

for any random function $h_i^{(r)}$ from $\mathbb{Z}_2^8$ to $\mathbb{Z}_2^{256}$ and 8-bit values x, y. The resulting lookup table $T_i^{(r)}$ is depicted in Fig 6. Then, if we set $x_{33} := x_1$,

$$\sum_{i=1}^{32} T_i^{(r)}(x_i, x_{i+1}) = \sum_{i=1}^{32} F_i^{(r)}(x_i, x_{i+1}) = F^{(r)}(x_1, \cdots, x_{32})$$

for 8-bit valued x_i's. Furthermore, since the functions $h_i^{(r)}$ have no certain structure unlike $F_i^{(r)}$, we cannot extract encodings from $T_i^{(r)}$'s using the affine equivalence algorithm.

Therefore, we can express the encoded round function $F^{(r)}$ as a sum of 16-bit to 256-bit lookup tables $T_i^{(r)}$'s without revealing $F_i^{(r)}$'s.

External Encoding Tables Let M_{in} and M_{out} be random affine functions on 256 bits. Then, external input encoding function $F^{(0)}$ is defined by $F^{(0)} = (A^{(1)})^{-1} \circ M_{\text{in}}$ and the encoded round function of the final round is defined by $F^{(10)} = M_{\text{out}} \circ (\text{AES}^{(10)}, \text{AES}^{(10)}) \circ A^{(10)}$, where $\text{AES}^{(10)} = \oplus_{K^{11}} \circ \mathsf{SR} \circ (S, \cdots, S) \circ \oplus_{K^{10}}$.

Since the external input encoding function $F^{(0)}$ is an affine function on 256 bits, it is implemented using a matrix of size 256 × 256 and a vector of length 256 bits. The external output encoding function $F^{(10)}$ is split into thirty-two 16-bit to 256-bit lookup tables $T_i^{(10)}$ by the above design technique.

Security Analysis Since the encoded round function $F^{(r)}$ is a bijection on 256 bits, computing the inverse value of $F^{(r)}$ is not an easy task for $r = 1, \cdots, 10$. Therefore, we count the complexity of computing the inverse of $F^{(r)}$ in the cryptanalysis of our proposed white-box implementation using SAEA. The hardness of inverting $F^{(r)}$ for $r = 1, \cdots, 10$, which has sparse unsplit encodings can be considered as a special version of sparse subset sum problem (SSSP) used in [7, 11, 22] to design fully homomorphic encryptions.

If A^r is a block diagonal matrix (*i.e.,* all off-diagonal blocks A^r_{ij} $(i \neq j)$ are zero matrices), then $F^{(r)}$ is expressed by the summation of the $F_i^{(r)}$'s, where $F_i^{(r)}$ is a function from $\mathbb{Z}_2^8$ to $\mathbb{Z}_2^{256}$. For a given 256-bit value y and sets $\{(x, F_i^{(r)}(x)) \in \mathbb{Z}_2^8 \times \mathbb{Z}_2^{256} \mid x \in \{0,1\}^8\}_{i=1}^{32}$, computing $(F^{(r)})^{-1}$ of y is to find the 8-bit values x_i such that $y = \sum_{i=1}^{32} F_i^{(r)}(x_i) = F^{(r)}(x_1, \cdots, x_{32})$. It is equivalent to a variant of the SSSP problem to finding the coefficients $\delta_{i,x}$, such that

$$y = \sum_{i=1}^{32} \sum_{x \in \{0,1\}^8} \delta_{i,x} \cdot F_i^{(r)}(x),$$

where

$$\delta_{i,x} \in \{0,1\}, \#\{x \in \{0,1\}^8 \mid \delta_{1,x} = 1\} = \cdots = \#\{x \in \{0,1\}^8 \mid \delta_{32,x} = 1\} = 1.$$

In this case, if we regard $F^{(r)}$ as a bijection on m_A bits, this SSSP can be solved in $\widetilde{O}(2^{m_A/2})$ time with $\widetilde{O}(2^{m_A/4})$ memory using a variant of Schroeppel–Shamir algorithm [21]. This is the same complexity as for the proposed MITM attack. However, the presented implementation uses an unsplit encoding that is not a block diagonal mapping. As a result, the 8-bit value x_i is used as the input value of several $T_j^{(r)}$'s, and the computation of $(F^{(r)})^{-1}$ is slightly different from that in the SSSP. Using the unsplit encoding instead of a block diagonal encoding makes the computation of $(F^{(r)})^{-1}$ from the subfunctions $T_j^{(r)}$ more difficult. Therefore, computing $(F^{(r)})^{-1}$ from the subfunctions $T_j^{(r)}$ seems as difficult as in the SSSP, and the subfunctions $T_j^{(r)}$ do not help determine the inverse of $F^{(r)}$.

More generally, if we use sparse unsplit input encodings that are affine mappings on m_A bits, then the complexity of extracting the secret key in the proposed implementation is

$$O\left(\min\left\{{m_A}^{12} \cdot 2^{14}, 2m_A \cdot \log m_A \cdot 2^{m_A/2}\right\}\right)$$

for $m_A = 128, 256, 384, \cdots$. Table 1 presents the security level and storage requirements of the proposed implementation for $m_A = 128, 256, 384$. The attack complexity can be up to 2^{110} and 2^{117} when $m_A = 256$ and 384, respectively, which is quite close to the original 128-bit security level. This shows that sparse unsplit input encodings that have a multiple input size of the cipher's block length may be a useful way of designing a secure white-box implementation.

m_A	Security	Storage
	$\min\left\{{m_A}^{12} \cdot 2^{14}, 2m_A \cdot \log m_A \cdot 2^{m_A/2}\right\}$	$\frac{m_A}{8} \cdot m_A \cdot 2^{16}$ bits
128	$2m_A \cdot \log m_A \cdot 2^{m_A/2} = 2^{75}$	16 MB × (# of rounds)
256	${m_A}^{12} \cdot 2^{14} = 2^{110}$	64 MB × (# of rounds)
384	${m_A}^{12} \cdot 2^{14} = 2^{117}$	144 MB × (# of rounds)

Table 1: The security and storage of the proposed white-box AES implementation for $m_A = 128, 256, 384$

6 Conclusion

In this paper, we proposed a general analytic toolbox for white-box implementation, which can efficiently extract the secret encodings used to obfuscate in the implementation when its design follows CEJO framework. With our toolbox, it is very easy to evaluate the asymptotic complexity of any white-box implementation of SLT ciphers, and all previous designs belong to this model. Hence, our toolbox could be used to measure the security of white-box implementations.

Another advantage of our toolbox is that we can remove insecure designs at an early stage, and concentrate on more plausible approaches. We showed that the input size of the encodings is the most important factor in the security of a white-box implementation. In this sense, we presented a white-box implementation that uses sparse unsplit input encodings with an input size that is a multiple of the block length. This not only produces high level of security, but also has reasonable storage requirements. Thus, we have developed a good candidate design for a white-box implementation with practical security. We expect that this will lead to the design of a practically secure white-box implementation.

References

1. O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In H. Handschuh and M. A. Hasan, editors, *Selected Areas in Cryptography - SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, 2005.
2. A. Biryukov, C. Bouillaguet, and D. Khovratovich. Cryptographic Schemes Based on the ASASA Structure: Black-box, White-box, and Public-key. *IACR Cryptology ePrint Archive*, 2014:474, 2014. To appear in ASIACRYPT 2014.
3. A. Biryukov, C. D. Canniére, A. Braeken, and B. Preneel. A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 33–50. Springer, 2003.
4. A. Biryukov and A. Shamir. Structural Cryptanalysis of SASAS. In B. Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 395–405. Springer, 2001.
5. S. Chow, P. Eisen, H. Johnson, and P. C. V. Oorschot. White-Box Cryptography and an AES Implementation. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography - SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, 2003.
6. S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot. A White-Box DES Implementation for DRM Applications. In J. Feigenbaum, editor, *Digital Rights Management - DRM 2002*, volume 2696 of *LNCS*, pages 1–15. Springer, 2003.
7. J.-S. Coron, D. Naccache, and M. Tibouchi. Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 446–464. Springer, 2012.
8. C. Delerablée, T. Lepoint, P. Paillier, and M. Rivain. White-Box Security Notions for Symmetric Encryption Schemes. In T. Lange, K. Lauter, and P. Lisoněk, editors, *Selected Areas in Cryptography - SAC 2013*, LNCS, pages 247–264. Springer, 2014.
9. I. Dinur, O. Dunkelman, N. Keller, and A. Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial search Problems. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*, pages 719–740. Springer, 2012.
10. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In C. Koc, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.

11. C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 129–148. Springer, 2011.
12. M. Karroumi. Protecting White-Box AES with Dual Ciphers. In K.-H. Rhee and D. Nyang, editors, *Information Security and Cryptology - ICISC 2010*, volume 6829 of *LNCS*, pages 278–291. Springer, 2011.
13. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
14. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
15. T. Lepoint, M. Rivain, Y. D. Mulder, P. Roelse, and B. Preneel. Two Attacks on a White-Box AES Implementation. In T. Lange, K. Lauter, and P. Lisoněk, editors, *Selected Areas in Cryptography - SAC 2013*, LNCS, pages 265–285. Springer, 2014.
16. W. Michiels, P. Gorissen, and H. D. L. Hollmann. Cryptanalysis of a Generic Class of White-Box Implementations. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *LNCS*, pages 414–428. Springer, 2009.
17. Y. D. Mulder, P. Roelse, and B. Preneel. Cryptanalysis of the Xiao - Lai White-Box AES Implementation. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography - SAC 2012*, volume 7707 of *LNCS*, pages 34–49. Springer, 2013.
18. R. Novak. Spa-based adaptive chosen-ciphertext attack on rsa implementation. In D. Naccache and P. Paillier, editors, *Public Key Cryptography - PKC 2002*, volume 2274 of *LNCS*, pages 252–262. Springer, 2002.
19. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In I. Attali and T. Jensen, editors, *Smart Card Programming and Security*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.
20. A. Saxena, B. Wyseur, and B. Preneel. Towards Security Notions for White-Box Cryptography. In P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, editors, *Information Security*, volume 5735 of *LNCS*, pages 49–58. Springer, 2009.
21. R. Schroeppel and A. Shamir. A TcS2 = 0 (2n) time/space tradeoff for certain NP-complete problems. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 328–336, Oct 1979.
22. N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.
23. B. Wyseur. *White-Box Cruptography*. PhD thesis, Katholieke Universiteit Leuven, 2009.
24. B. Wyseur, W. Michiels, P. Gorissen, and B. Preneel. Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography - SAC 2007*, volume 4876 of *LNCS*, pages 264–277. Springer, 2007.
25. Y. Xiao and X. Lai. A Secure Implementation of White-box AES. In *Computer Science and its Applications - CSA 2009*, pages 1–6. IEEE, 2009.

A Proofs of the Lemmas

A.1 Proof of Lemma 1

We may assume that A is linear, because an addition by a constant preserves property P_m or E_m when M has an even number of elements.

Let $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}$ with column vectors a_i's and $A^* = \begin{bmatrix} a_{i_1} & \cdots & a_{i_\tau} \end{bmatrix}$. Then we have $A(M) = \{A^*(x') + b \mid x' \in \pi_I(M)\}$ for some constant vector $b \in \mathbb{Z}_2^m$. It is enough to show that the multiset $A^*(\pi_I(M))$ has property P_m or E_m.

If $\tau = m$ and A^* has rank m, then the multiset $A^*(\pi_I(M))$ of m-bit values has property P_m. Otherwise, the size of the kernel of A^* is $2^{\tau - \mathrm{rank}(A^*)}$ and hence the number of preimage of $y \in A^*(\pi_I(M))$ is $2^{\tau - \mathrm{rank}(A^*)}$. Since $2^{\tau - \mathrm{rank}(A^*)}$ is even, the multiset $A^*(\pi_I(M))$ has property E_m. It follows that $A^*(\pi_I(M))$ has property P_m or E_m. □

A.2 Proof of Lemma 2

We can write $A(x) = L(x) + b$ for some linear mapping L from n bits to m bits and a m bits value b. Then

$$\sum_{y \in A(M)} y = \sum_{x \in M} (Lx + b) = L\left(\sum_{x \in M} x\right) + \sum_{x \in M} b = 0$$

since the multiset M has property B_n and the size of M is even. □

B The SAEA with split A

When we use the SAEA for the white-box implementation, we can reduce the complexity for the case where input encoding A is of some special form.

For convenience, let A and B be linear. Let's consider $A \in (\mathbb{Z}_2)^{n\times n}$ as a $\tilde{A} \in (\mathbb{Z}_2^{m\times m})^{k\times k}$, where $n = km$. If $\tilde{A}$ is block-diagonal map, then we can perform separately the above attack on each block. If the size of the each block of block-diagonal map $\tilde{A}$ is k_i with $\sum_i k_i = k$, $A_i \in \mathbb{Z}_2^{k_im\times k_im}$ is i-th block of A and $B_i \in \mathbb{Z}_2^{n\times k_im}$ is i-th vertical strip of B correspond to A_i, we can find maps of the form $F_i = B_i \circ (S, \cdots, S) \circ A_i$, where $(S, \cdots, S)$ is concatenation of k_i S-boxes. Since image of F_i has rank k_i, we can find a $k_i \times n$ matrix C_i satisfying that $C_i \circ F_i$ is bijective. Then we obtain bijective maps of the following form:

$$\tilde{F_i} = \tilde{B_i} \circ (S, \cdots, S) \circ A_i$$

where $\tilde{B_i} = C_i \circ B_i$. Thus we can recover the encodings in complexity $\sum_i k_i(k_im)^3 2^{3m}$, less than $kn^3 2^{3m}$.

More generally, if $\tilde{A}$ can be split into two or more bijective map, that is, A is *split* as defined in section 3.2, we can apply the above argument to $\tilde{A}$. In detail, in the case that $(\tilde{A})_{i,j} = 0^{m\times m}$ for $(i, j) \in [k_1 + 1, k_1 + k_0] \times ([1, k] \setminus [k_2 + 1, k_2 + k_0])$ or $(i, j) \in ([1, k] \setminus [k_1 + 1, k_1 + k_0]) \times [k_2 + 1, k_2 + k_0]$ fore some k_0, k_1, k_2 with $k_1 + k_0, k_2 + k_0 \leq k$,
i.e., $\tilde{A}$ is the form of $\left[\begin{array}{c|c|c} * & 0 & * \\ \hline 0 & A^* & 0 \\ \hline * & 0 & * \end{array}\right]$, one can obtain a bijective map on $\mathbb{Z}_2^{k_0m}$ using small submatrix, A^* in the above.

C Computing Inverse of F

In order to use the SAEA, several number of evaluation of F^{-1} are required, so we need to check the complexity to compute the inverse of F. As in Section 3.2, we let F be a bijection on n bits. For simplicity, we assume $F(x) = \sum_{j=1}^{k} F_j(x_j)$ for $F_j : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$, where $x = (x_1, \cdots, x_k)$ and x_j's are m-bit values with $n = km$. A trivial approach to invert F is the exhaustive search, which takes 2^n time complexity. One can improve it using the meet-in-the-middle (MITM) attack: By combining functions, we let $F(x) = G_1(x_1, \cdots x_{\lfloor \frac{k}{2} \rfloor}) + G_2(x_{\lfloor \frac{k}{2} \rfloor+1}, \cdots, x_k)$. For $y \in \mathbb{Z}_2^n$, one can make a table of $\oplus_y \circ G_1$, sort it by the output values, and compare it with the value of G_2. Then one can evaluate $F^{-1}(y)$ in $O(n2^{n/2})$ time complexity with $O(n2^{n/2})$ memory. The size of required memory for the MITM attack is quite large to implement - for example, 2^{38} GB are required for $n = 128$. We provide another method requiring smaller memory while maintaining asymptotic time complexity.

For convenience of notations, we let $F(x) = \sum_{j=1}^{4} F_j(x_j)$ where $x = (x_1, x_2, x_3, x_4)$ and each x_j is an m_0-bit value with $m_0 = \frac{n}{4}$. For a function f whose value is on n bits, $\tilde{f}$ denotes the projection of f on the first m_0 bits. To evaluate $F^{-1}(y)$ for any n-bit value y, we perform the following steps:

1. Guess m_0-bit value $\tilde{z}$ for $\tilde{F_1}(x_1) + \tilde{F_2}(x_2)$.
2. Perform the MITM attack using $\tilde{F_1}(x_1) + \tilde{F_2}(x_2) = \tilde{z}$ and store the list

   $$L = \left\{(x_1, x_2, F_1(x_1) + F_2(x_2) + y) \mid \tilde{F_1}(x_1) + \tilde{F_2}(x_2) = \tilde{z}\right\}$$

   for the result of the MITM attack.
3. Perform the MITM attack using $\tilde{F_3}(x_3) + \tilde{F_4}(x_4) = \tilde{y} + \tilde{z}$, where $\tilde{y}$ is the first m_0-bit of y.
4. For each (x_3, x_4) satisfying $\tilde{F_3}(x_3) + \tilde{F_4}(x_4) = \tilde{y} + \tilde{z}$, compare $F_3(x_3) + F_4(x_4)$ with the values in L.

Both the average number of elements in L and the number of (x_3, x_4) satisfying $\tilde{F}_3(x_3) + \tilde{F}_4(x_4) = \tilde{y} + \tilde{z}$ are $2^{n/4}$. For one guessing of $\frac{n}{4}$-bit value, we perform 3 times of MITM attacks on the sets with $2^{n/4}$ cardinality. Therefore, we can evaluate $F^{-1}(y)$ in $O(n2^{n/2})$ time complexity with $O(n2^{n/4})$ memory. For the case of $n = 128$, the required memory is 64 GB, which is practical to implement.