WHITE-BOX BLOCK CIPHER IMPLEMENTATION BASED ON LS-DESIGN

HATICE KÜBRA GÜNER, CEYDA MANGIR, AND OĞUZ YAYLA

Abstract. Protecting secret keys from malicious observers in untrusted environments is a critical security issue. White-box cryptography suggests software protection by hiding the key in the white-box setting. One method for hiding the key in the cipher code is through encoding methods. Unfortunately, encoding methods may be vulnerable to algebraic attacks and side-channel analysis. Another technique to hide the key is (M,Z)-space hardness approach that conceals the key into a large lookup table generated with a reliable small block cipher. In (M,Z)-space-hard algorithms, the key extraction problem in the white-box setting turns into a key recovery problem in the black-box setting. One of the problems for (M,Z)-space-hard algorithms is improving run-time performance. In this study, we aim to improve the run-time performance of the existing white-box implementations. We propose an LS-design based white-box algorithm with better run-rime performance than space-hard SPNbox algorithm. Moreover, an LS-design based table creation method is designed. When we compare the run-time performance of our method with the SPNbox algorithm, we obtain 28% improvement for white-box implementation and 27% for black-box implementation for 128-bit block size. The LS-design based method is also used for 256-bit block size in the white-box setting.

1. Introduction

Protecting the secret key of a cryptographic algorithm is a major problem in white-box attack conditions. In the white-box attack conditions, the attacker can observe the internal details of the encryption process. White-box cryptography is a software protection method for devices in untrusted environments that preserves the secret key in the algorithm layers.

The white-box DES [11] for DRM applications in 2002 and the white-box AES [12] in 2003 were proposed by Chow et al. In the white-box AES, the key is embedded in the Sbox, and encoding methods are used to hide the algorithm layers. Unfortunately, white-box AES was broken by algebraic attacks [2, 23]. Some other variants have been proposed that were secure against the applied algebraic attacks [25, 34] but were also broken [15, 30].

ASASA structure based memory-hard white-box algorithm against code-lifting attacks has been proposed by Biryukov et al. [3]. The ASASA structure consists of nonlinear layer S and affine layer A. The code-lifting attacks enable the attacker to use the algorithm as a large key outside the device. Although the proposed white-box method is not secure against key recovery attacks, the memory-hard algorithms inspired the idea of space-hard white-box algorithms.

Key words and phrases. White-box cryptography, Space-hard ciphers, Software protection, LS-design, Efficiency.

1.1. **(M,Z)-Space Hard Ciphers.** The (M,Z)-space hard white-box ciphers aim to provide security using a reliable small block cipher instead of internal and external coding throughout the algorithm phases. A large lookup table is created with this small block cipher. Thus, the secret key is embedded in this table, which is used as a substitution box in the nonlinear layer. In this case, the key extraction issue in the white-box setting becomes a key recovery issue in the black-box setting. The first (M,Z)-space hard white-box algorithm proposed by Bogdanov and Isobe is Space [6], based on the Feistel structure, which hides the key in the F function.

The incompressibility of the table, where the key is embedded, is a critical security notion against code-lifting attacks [16]. Weak (M,Z)-space hardness is defined to resist these attacks. The generated key-based tables must be updated regularly according to this definition to ensure security in the white-box setting.

Definition 1.1 (Weak (M,Z)-space hardness [6])**.** A white-box block cipher is called weak (M,Z)-space hard if it is not possible to encrypt a randomly selected text with a probability greater than 2^{-Z} until the leakage size from the table is reached to M bits.

The SPN-based (M,Z)-space hard cipher SPNbox was suggested by Bogdanov et al. [7] to improve Space's run-time performance. To achieve efficiency, they used a lightweight MDS matrix in the linear layer of the white-box algorithm. SIMD instructions were used to improve performance in the implementation. Moreover, an efficient key-based table generation method was proposed using AES components.

Another SPN-based cipher WARX was proposed by Liu et al. [31] to improve the performance of white-box and black-box implementations using addition, rotation, xor (ARX) operations. The table creation method in WARX was inspired by the lightweight SPARX [17] algorithm. They also proposed using a random MDS matrix in the linear layer of the white-box algorithm to reduce the round number.

Other than these (M,Z)-space hard algorithms, Feistel-based WhiteBlock [19], FPL [28] and Galaxy [27], Even-Mansur structure based WEM [10] and SPN-based Yoroi [26] were proposed to accelerate run-time performance while keeping the security strength of the white-box algorithms.

1.2. **Our Contribution.** Creating reliable key-based tables and updating these tables are fundamental security notions for the space-hard white-box algorithms. At the same time, accelerating the run-time performance of white-box/black-box algorithms becomes important, as efficiency is as essential as security in real-world applications. Lightweight MDS matrices were preferred in SPN-based white-box algorithms since they provide higher security strength with less computational cost.

This study aims to propose a new space-hard white-box algorithm that is handling lightweight components other than MDS matrices utilized in the current algorithms. With this motivation, the linear components of the NIST Lightweight competition [22] candidate Spook [1] were found suitable for an efficient space-hard white-box algorithm without reducing the security level. A new (M,Z)-space hard white-box algorithm based on LS-design [20] is implemented in this paper to accelerate the run-time performance of the existing algorithms. Also, a new table creation method based on LS-design is proposed to take advantage of the bitslice implementation against side-channel analysis.

The details of the LS-design based white-box algorithm and the new table creation method are explained in Section 2. The security issues for the table creation

method and the white-box algorithm are detailed in Section 3. The run-time performance results are stated in Section 4.

2. An LS-Design Based White-Box Block Cipher

LS-design based algorithms [20, 24] aim to prevent differential side-channel analysis with bitslice implementations. With this scope, we propose an LS-design based white-box algorithm and table construction method to take advantage of security considerations along with the run-time performance improvement. We give the specifications of our algorithm below.

2.1. **Table Creation Method.** A new LS design-based small-block cipher is derived as a method of creating tables for use in the white-box context. The block size of the table construction algorithm is 32-bit, and the state is taken as an (8×4)-bit grid as shown in Figure 1. The round transformation of the algorithm has a key addition layer, a nonlinear layer as a substitution box, a bitslice implemented linear layer, and a round constant layer. The nonlinear layer tSbox is applied to two concatenated 4-bit columns, while the linear layer tLbox is applied to 8-bit rows. The round keys are obtained with an XOF instead of key scheduling algorithm, and the round constants are generated with an 8-bit LFSR. The algorithm consists of 12/16 rounds to provide 128/256-bit security.

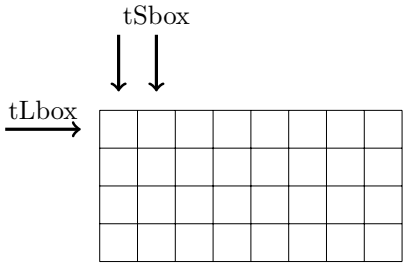

Figure 1. State of the Input

In the table creation method, each element of the field $\mathbb{F}_2^{32}$ is encrypted with the LS-design based small block cipher. At the beginning of the algorithm, the input is reordered with the *bitslice* function as the concatenating 4-bit columns, as shown in the grid structure. After round transformations, it is sorted in initial order with the *unbitslice* function. We applied the bitslice and unbitslice functions only once to improve run-time performance, rather than each *tsbox* and *tlbox* layers. The pseudo-code of the table construction method is given in the Algorithm 1.

2.1.1. *tSbox.* The 8-bit tSbox is taken from Scream-v3 algorithm [21], the second round candidate of CAESAR competition [13]. The tSbox consists of three steps based on the Feistel structure. The first and third steps are almost perfect nonlinear (APN) functions, and the second step is a permutation with differential uniformity 4. Details of the Feistel structures are stated in Table 1. In the implementation, the nonlinear layer *tsbox* is pre-computed and used as a substitution box.

Algorithm 1 Table Construction

INPUT: $x \in (\mathbb{F}_2^{32})$
OUTPUT: x
$x \leftarrow bitslice(x)$
for $i = 0$ to tr **do**
  $x \leftarrow addroundkey(x, key[i])$
  $x \leftarrow tsbox(x)$
  $x \leftarrow tlbox(x)$
  $x \leftarrow addroundconstant(x)$
end for
$x \leftarrow unbitslice(x)$

TABLE 1. tSbox.

Step 1
$x_0 = (s_1 \ \& \ s_2) \oplus s_0$
$x_1 = s_1 \oplus s_3$
$x_2 = s_2 \oplus x_0$
$s_4 = s_4 \oplus ((s_3 \oplus x_2) \ \& \ (s_2 \oplus x_1))$
$s_5 = s_5 \oplus x_2$
$s_6 = s_6 \oplus (s_3 \ \& \ x_0)$
$s_7 = s_7 \oplus (x_1 \ \& \ x_2)$
Step 2
$x_0 = (s_4 \ \& \ s_5) \oplus s_6$
$x_1 = (s_5 \mid s_6) \oplus s_7$
$x_2 = (s_7 \ \& \ x_0) \oplus s_4$
$x_3 = (s_4 \ \& \ x_1) \oplus s_5$
$s_0 = s_0 \oplus x_0$
$s_2 = s_2 \oplus x_1$
$s_1 = s_1 \oplus x_2$
$s_3 = s_3 \oplus x_3$
Step 3
$x_0 = \neg((s_1 \ \& \ s_2) \oplus s_0)$
$x_1 = s_1 \oplus s_3$
$x_2 = s_2 \oplus x_0$
$s_4 = s_4 \oplus ((s_3 \oplus x_2) \ \& \ (s_2 \oplus x_1))$
$s_5 = s_5 \oplus x_2$
$s_6 = s_6 \oplus (s_3 \ \& \ x_0)$
$s_7 = s_7 \oplus (x_1 \ \& \ x_2)$

2.1.2. *tLbox.* The linear layer tLbox is taken from Mysterion algorithm [24]. The permutation is a recursive MDS matrix obtained from an $[16, 8, 9]_{\mathbb{F}_{2^4}}$ MDS code with the branch number 9. The recursive MDS matrix is constructed by calculating the k-power of the complementary matrix in the field $\mathbb{F}_{2^9}$. In the Mystreion

algorithm, the companion matrix M is taken as

$$M=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 8 & 3 & f & 5 & f & 3 & 8 \end{bmatrix}$$

The MDS matrix and its inverse are computed as 8-power of the companion matrix M for $q = 4$ with the reduction polynomial $p(x) = x^4 + x + 1$.

$$M^8=\begin{bmatrix} 1 & 8 & 3 & f & 5 & f & 3 & 8 \\ 8 & d & 3 & 2 & 1 & 4 & 4 & f \\ f & 9 & f & 9 & 4 & b & 6 & 5 \\ 5 & 1 & 6 & 9 & b & 2 & 4 & 8 \\ 8 & 9 & a & 7 & 7 & a & 9 & 8 \\ 8 & 4 & 2 & b & 9 & 6 & 1 & 5 \\ 5 & 6 & b & 4 & 9 & f & 9 & f \\ f & 4 & 4 & 1 & 2 & 3 & d & 8 \end{bmatrix} \qquad (M^8)^{-1}=\begin{bmatrix} 8 & d & 3 & 2 & 1 & 4 & 4 & f \\ f & 9 & f & 9 & 4 & b & 6 & 5 \\ 5 & 1 & 6 & 9 & b & 2 & 4 & 8 \\ 8 & 9 & a & 7 & 7 & a & 9 & 8 \\ 8 & 4 & 2 & b & 9 & 6 & 1 & 5 \\ 5 & 6 & b & 4 & 9 & f & 9 & f \\ f & 4 & 4 & 1 & 2 & 3 & d & 8 \\ 8 & 3 & f & 5 & f & 3 & 8 & 1 \end{bmatrix}$$

In the implementation, the *xtime* function for the finite field multiplication is defined as:

$$(((x) << (1)) \oplus ((((x) >> (3))\&1) \cdot (0x13)))$$

Depending on the matrix values, the *xtime* function is applied to 4-bit inputs up to 3 times in succession. The size of these pre-computed *xtime* values is 192 bits. Therefore, the *xtime* values are pre-computed to achieve efficiency in the implementation.

2.1.3. *Round Keys.* The round keys are generated using extendable output function SHAKE [18] with the master key. For the 128-bit security case, the 128-bit master key is expanded to 384-bit for twelve rounds. Similarly, the 256-bit master key is expanded to a 512-bit key for 16 rounds of 256-bit security level. The extended outputs are divided into 32 bits, and the state is xored with the corresponding round key in the *addroundkey* layer.

2.1.4. *Round Constants.* Round constants are generated with an 8-bit LFSR with the feedback polynomial $p(x) = x^8 + x^6 + x^5 + x^4 + 1$. The output of the LFSR is divided into 32 subwords of length 8, and the subwords are taken as round constants. The state is xored with the corresponding round constant in the *addroundconstant* layer.

2.2. **Specifications of The Algorithm.** The LS-design based white-box algorithm is implemented with 32 bits word and 128/256 bits key and block sizes. The round transformation of the algorithm is implemented to 128-bit subblocks. One round of the algorithm consists of a table-based nonlinear layer, a bitslice implemented linear layer, and a round constant layer. The generated table T is used as a substitution box in the nonlinear layer. The pseudo-code of the white-box

implementation is given in Algorithm 2. The numbers of rounds for the white-box implementations' 128-bit and 256-bit block sizes are calculated as 12 and 14, respectively.

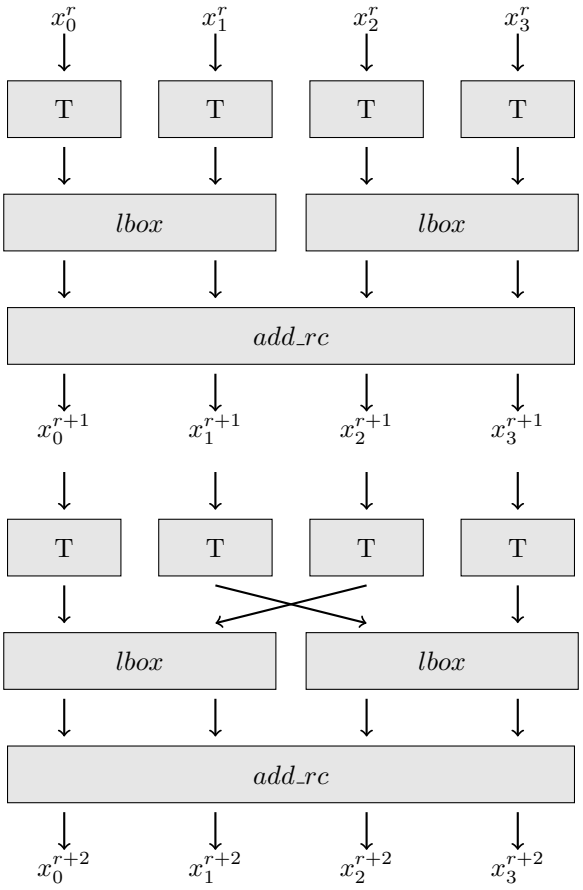

FIGURE 2. Two rounds of the white-box algorithm of 128-bit block size

2.2.1. *Nonlinear layer.* The nonlinear layer is implemented as a key-dependent substitution box generated by a small-scale block cipher. This small block cipher is also based on the LS-design approach with a 32-bit block size. The details of the small block cipher are given in Table Construction part 2.1.

2.2.2. *Linear Layer.* The linear layer Lbox is taken from the lightweight design Spook [1]. Bitslice implemented linear layer *lbox* takes two 32-bit inputs and is applied twice in one round. The internal operations are specified as in Table 2. The order of the *lbox* entries is chosen according to the round number to increase the diffusion effect. In the even rounds, the first Lbox takes the upper half of the state words and the second Lbox takes the other side of the words. However, in the odd rounds, the first Lbox gets the first and third words of the state, and the second Lbox gets the remaining words.

Algorithm 2 The new LS-design based white-box algorithm

INPUT: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 4 \cdot n - 1\}$, T-Table
OUTPUT: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 4 \cdot n - 1\}$
for $i = 0$ to R-1 **do**
  for $j = 0$ to 4·n-1 **do**
    $x_j \leftarrow T(x_j)$
  end for
  for $j = 0$ to n-1 **do**
    $(x_{4\cdot j}, x_{4\cdot j+1+(j\&1)}) \leftarrow lbox(x_{4\cdot j}, x_{4\cdot j+1+(j\&1)})$
    $(x_{4\cdot j+2-(j\&1)}, x_{4\cdot j+3}) \leftarrow lbox(x_{4\cdot j+2-(j\&1)}, x_{4\cdot j+3})$
    $(x_{4\cdot j}, \cdots, x_{4\cdot j+3}) \leftarrow add\_rc((x_{4\cdot j}, \cdots, x_{4\cdot j+3}), i)$
  end for
  if n==2 **then**
    $(x_0 \cdots x_{4\cdot n}) \leftarrow dbox(x_0 \cdots x_{4\cdot n})$
    for $j = 0$ to n-1 **do**
      $(x_{4\cdot j}, \cdots, x_{4\cdot j+3}) \leftarrow add\_drc((x_{4\cdot j}, \cdots, x_{4\cdot j+3}), i)$
    end for
  end if
end for

TABLE 2. Lbox

$u = x \oplus (x \lll 12);$	$v = y \oplus (y \lll 12);$
$u = u \oplus (u \lll 3);$	$v = v \oplus (v \lll 3);$
$u = u \oplus (x \lll 17);$	$v = v \oplus (y \lll 17);$
$t = u \oplus (u \lll 31);$	$z = v \oplus (v \lll 31);$
$u = u \oplus (z \lll 26);$	$v = v \oplus (t \lll 25);$
$u = u \oplus (t \lll 15);$	$v = v \oplus (z \lll 15)$

Since the round transformation of the implementation is applied to 128-bit blocks, an additional linear component, Dbox, is used to diffuse 128-bit subblocks to each other for 256-bit block size. The Dbox has left-rotate and xor operations shown in Table 3. Each word of one subblock is rotated to the left and xored to the corresponding state of the other subblock.

TABLE 3. Dbox

$x = x \oplus (y \lll 15);$
$y = y \oplus (x \lll 19)$

2.2.3. *Round Constants.* Round constants are generated from a 4-bit LFSR with the feedback polynomial $f(x) = x^4 + x^3 + 1$. For 256-bit block size, an additional round constant layer is implemented after the Dbox layer. These round constants are generated from 8-bit LFSR with the feedback polynomials $f(x) = x^8 + x^5 + x^3 + x + 1$.

2.2.4. *Computational Cost.* One round of 128-bit white-box implementation has $24 \cdot xor + 24 \cdot or$ from $lbox$, and $4 \cdot xor$ operations from add_rc. Hence, the total computational cost of the white-box implementation is $336 \cdot xor + 288 \cdot or$ operations. Similarly, one round of the 256-bit block has $48 \cdot xor + 48 \cdot or$ from lbox, $8 \cdot xor$ and $8 \cdot or$ operations from dbox, and $16 \cdot xor$ operations from add_rc. Therefore, the total computational cost of the 256-bit white-box implementation is $1008 \cdot xor + 784 \cdot or$ operations.

2.3. **The Black-Box Algorithm.** The only difference in the black-box implementation is that the table construction algorithm, which is used in the nonlinear layer, is implemented instead of used as a pre-computed table. The pseudo-code of the black-box algorithm is given in Algorithm 3.

Algorithm 3 Black-box algorithm

INPUT: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 4 \cdot n - 1\}$, T-Table
OUTPUT: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 4 \cdot n - 1\}$
for $i = 0$ to R-1 **do**
  for $j = 0$ to 4·n-1 **do**
    $x_j \leftarrow bitslice(x_j)$
    for $i = 0$ to tr **do**
      $x_j \leftarrow addroundkey(x_j, key[i])$
      $x_j \leftarrow tsbox(x_j)$
      $x_j \leftarrow tlbox(x_j)$
      $x_j \leftarrow addroundconstant(x_j)$
    end for
    $x_j \leftarrow unbitslice(x_j)$
  end for
  for $j = 0$ to n-1 **do**
    $(x_{4 \cdot j}, x_{4 \cdot j+1+(j\&1)}) \leftarrow lbox(x_{4 \cdot j}, x_{4 \cdot j+1+(j\&1)})$
    $(x_{4 \cdot j+2-(j\&1)}, x_{4 \cdot j+3}) \leftarrow lbox(x_{4 \cdot j+2-(j\&1)}, x_{4 \cdot j+3})$
    $(x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}) \leftarrow add\_rc((x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}), i)$
  end for
  if n==2 **then**
    $(x_0 \cdots x_{4 \cdot n}) \leftarrow dbox(x_0 \cdots x_{4 \cdot n})$
    for $j = 0$ to n-1 **do**
      $(x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}) \leftarrow add\_drc((x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}), i)$
    end for
  end if
end for

3. Security

The security strength of the table creation method and the LS-design based white-box algorithm are evaluated against black-box and white-box attacks.

3.1. **The Security of Table Construction Method.** The table construction method is analyzed against the known black-box attacks. Also, the structural attacks based on the faulty construction of LS-design are discussed for our design method.

3.1.1. *Differential and Linear Cryptanalysis.* The LS-design is based on WTS approach [14]. The linear and differential probability are formalized in [20] as,

$$Pr_{\text{diff}}(2r) \leq Pr_{\text{diff}}^{\max}(S)^{r \cdot B(L)} \tag{3.1}$$

and

$$Pr_{\text{lin}}(2r) \leq Pr_{\text{lin}}^{\max}(S)^{r \cdot B(L)} \tag{3.2}$$

The differential and linear probability of the tSbox is 2^{-5} and 2^{-2}, and the branch number of the tLbox is 9. From Equation 3.1 and 3.2, the table generation method provides 128-bit security after eight rounds and 256-bit security after 14 rounds. Also, according to the MILP method in [32], there are 54 active tSboxes for twelve rounds and 72 active tSboxes for 16 rounds. Hence, the method resists differential and linear attacks with the determined round numbers.

3.1.2. *Slide Attacks.* Slide attacks [4, 5] exploit the algorithm's high degree of self-similarity vulnerability regardless of the round numbers. A different round constant is used to prevent slide attacks in every round of the table construction algorithm.

3.1.3. *Algebraic Attacks.* Algebraic attacks aim to recover the encryption key by solving the multivariate algebraic equations of the encryption system. The upper bound for the maximum algebraic degree for a block cipher is given in [9, 8]. Therefore, at least three rounds are required to reach the algebraic degree 31 against the attacks.

3.1.4. *Structural Attacks.* The invariant subspace attacks [29] and the nonlinear invariant attacks [33] are applied against LS-design algorithms. Both attacks rely on the vulnerability of using the weak key and sparse constants in the algorithms. Our table generation method is resistant to such attacks, as a different round key, generated by a reliable key derivation function, is used in each round. Also, round constants are generated with an LFSR instead of random sparse numbers.

3.2. **The White-box Security.** Key extraction and code-lifting security are the most fundamental security considerations for white-box algorithms. The key extraction security is related to the reliability of the table construction algorithm. The leak limit in (M,Z)-space hardness against code lifting attacks determines the white-box algorithm's round number. Hence, key extraction and code-lifting security are detailed for our white-box design.

3.2.1. *Key Extraction Security.* In space-hard ciphers, key extraction security is vital since the secret key is hidden in the lookup table used in the nonlinear layer. Therefore, key recovery attacks are needed to extract the key from the table created with a small block cipher in the black-box setting. The white-box algorithm is as resistant to key extraction attacks as is the reliability of the small-block cipher against key recovery attacks [6]. The key extraction security of our white-box implementation depends on the table creation method. The table creation method is designed to be secure against known attacks. Also, the purpose of bitslice implementation is to prevent side-channel attacks. Therefore, our LS-design based white-box implementation is secured against the key extraction attacks.

3.2.2. *Code Lifting Security.* In space-hard ciphers, the secret key is embedded in the lookup table, like a large device key. If an attacker retrieves the table from the device, he gets the encryption key. Therefore, incompressible tables must be used to prevent code lifting attacks [16]. Using a reliable small-block cipher to construct the lookup table provides incompressible tables for white-box implementation. On the other hand, since malicious sides observe the device in an untrusted environment, the table can be leaked piece by piece. According to the weak (M,Z)-space hardness definition, the table must be renewed when the leakage limit is reached.

The round number of the white-box algorithm is calculated according to the MAS level determined by the leakage limit in the weak (M, Z)-space hardness definition. The MAS size is defined as $keysize - log_2(tablesize)$ in [19, **?**, 31]. We have taken the MAS size by the leak size for white-box algorithms as $keysize - log_2(leaksize)$ to calculate the round number more precisely. The round numbers are computed with the Equation (3.3)

$$r = \frac{\log_2(L) - keysize}{n \cdot \log_2(\frac{L+1}{T})}, \tag{3.3}$$

where L is size of leakage, T is the table size, and n is the number of lookup table in a round. According to Equation 3.3, the white-box implementation requires at least 12/14 rounds to provide 93-bit/221-bit MAS strength for 128-bit/256-bit block size.

3.3. **The Black-Box Security.** The encryption algorithm is analyzed against differential and linear cryptanalysis, slide and structural attacks in the black-box environment.

3.3.1. *Differential and Linear Cryptanalysis.* The branch number of the linear layer is 16, and the algorithms are applied in 12/14 rounds. According to Equation 3.1 and 3.2, the desired security levels against differential and linear attacks are provided.

3.3.2. *Other Attacks.* The round constants are generated with an LFSR against slide attacks and LS-design based structural attacks. The reliability of the table creation method provides security against algebraic and related attacks.

4. Performance Results

The run-time performance of the white-box and black-box implementations was compared with the SPNbox-32 algorithm. The algorithms [1], run on randomly generated 3072 bytes messages with 100000 cycles and -O3 optimization on a laptop equipped with x86-64 architecture, a 2.80 GHz Intel Core i7-1165G7 CPU and 8 GB DDR4-3200 RAM. The performance results are given in Table 4 for white-box implementation and in Table 5 for black-box implementation.

According to the performance results in Table 4, wClyde is 28% faster than SPNbox-32, and wShadow is 15% faster than SPNbox-32.

The run-time performance of the black-box implementation was compared with the black-box SPNbox-32. SPNbox's method is implemented in 16 rounds for 128-bit security in the table creation, while our LS-based design is in 12 rounds. For the 256-bit security level, our table creation method is applied in 16 rounds. According

[1] https://github.com/hkcryp/wbc

TABLE 4. Performance results of the WBI.

Algorithm	Key Size	Round	MAS	WBI in Cycle (per byte)
SPNbox-32	128	16	128	138
wCylde	128	12	96	99
wShadow	256	14	221	117

to performance results in Table 5, the run-time performance of black-box wClyde is 27% faster than black-box SPNbox-32. Nevertheless, black-box wShadow is 26% slower than black-box SPNbox-32.

TABLE 5. Performance results of the BBI.

Algorithm	Key Size	Table	Cycle (per byte)
SPNbox-32	128	SPNbox	1801
wClyde	128	LS-design	1317
wShadow	256	LS-design	2272

5. Conclusion

White-box cryptography aims to provide software security for devices in untrusted environments where the key security cannot be provided by hardware tools such as TPM or TEE. In white-box cryptography, the key is embedded in encryption algorithm layers by appropriate methods. Using encoding methods has not provided security against key extraction attacks until now. However, (M,Z)-space-hard algorithms suggest hiding the secret key in a large lookup table handling a small block cipher. One of the issues of these ciphers is to improve the run-time performance of white-box and black-box implementations. This study proposes a new LS-design based white-box algorithm and table construction method. With this white-box algorithm, we have performance improvement as 28% for white-box implementation and 27% for black-box implementation with a 128-bit block size. Moreover, we propose an LS-design based white-box algorithm for 256-bit block size using the suggested table creation method.

References

1. Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaétan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaétan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, and Friedrich Wiemer, *Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher*, (2019), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Spook-spec-round2.pdf.
2. Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi, *Cryptanalysis of a white box aes implementation*, Selected Areas in Cryptography (Berlin, Heidelberg) (Helena Handschuh and M. Anwar Hasan, eds.), Springer Berlin Heidelberg, 2005, pp. 227–240.

3. Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich, *Cryptographic schemes based on the asasa structure: Black-box, white-box, and public-key (extended abstract)*, Advances in Cryptology – ASIACRYPT 2014 (Berlin, Heidelberg) (Palash Sarkar and Tetsu Iwata, eds.), Springer Berlin Heidelberg, 2014, pp. 63–84.
4. Alex Biryukov and David Wagner, *Slide attacks*, Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings 6, Springer, 1999, pp. 245–259.
5. ———, *Advanced slide attacks*, Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19, Springer, 2000, pp. 589–606.
6. Andrey Bogdanov and Takanori Isobe, *White-box cryptography revisited: Space-hard ciphers*, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '15, Association for Computing Machinery, 2015, p. 1058–1069.
7. Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser, *Towards practical whitebox cryptography: Optimizing efficiency and space hardness*, Advances in Cryptology – ASIACRYPT 2016 (Berlin, Heidelberg) (Jung Hee Cheon and Tsuyoshi Takagi, eds.), Springer Berlin Heidelberg, 2016, pp. 126–158.
8. Christina Boura and Anne Canteaut, *On the influence of the algebraic degree of f^{-1} on the algebraic degree of $g \circ f$*, IEEE Transactions on Information Theory **59** (2012), no. 1, 691–702.
9. Christina Boura, Anne Canteaut, and Christophe De Cannière, *Higher-order differential properties of keccak and luffa*, Fast Software Encryption (Berlin, Heidelberg) (Antoine Joux, ed.), Springer Berlin Heidelberg, 2011, pp. 252–269.
10. 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*, Topics in Cryptology – CT-RSA 2017 (Cham) (Helena Handschuh, ed.), Springer International Publishing, 2017, pp. 293–308.
11. Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot, *A white-box des implementation for drm applications*, Digital Rights Management (Berlin, Heidelberg) (Joan Feigenbaum, ed.), Springer Berlin Heidelberg, 2003, pp. 1–15.
12. Stanley Chow, Philip Eisen, Harold Johnson, and Paul C. Van Oorschot, *White-box cryptography and an aes implementation*, Selected Areas in Cryptography (Berlin, Heidelberg) (Kaisa Nyberg and Howard Heys, eds.), Springer Berlin Heidelberg, 2003, pp. 250–270.
13. Cryptographic Competitions, *Caesar: Competition for authenticated encryption: Security, applicability, and robustness*, https://competitions.cr.yp.to.
14. Joan Daemen and Vincent Rijmen, *The wide trail design strategy*, Cryptography and Coding (Berlin, Heidelberg) (Bahram Honary, ed.), Springer Berlin Heidelberg, 2001, pp. 222–238.
15. Yoni De Mulder, Peter Roelse, and Bart Preneel, *Cryptanalysis of the xiao–lai white-box aes implementation*, International conference on selected areas in cryptography, Springer, 2012, pp. 34–49.
16. Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain, *White-box security notions for symmetric encryption schemes*, Selected Areas in Cryptography – SAC 2013 (Berlin, Heidelberg) (Tanja Lange, Kristin Lauter, and Petr Lisoněk, eds.), Springer Berlin Heidelberg, 2014, pp. 247–264.
17. Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov, *Design strategies for arx with provable bounds: Sparx and lax*, Advances in Cryptology – ASIACRYPT 2016 (Berlin, Heidelberg) (Jung Hee Cheon and Tsuyoshi Takagi, eds.), Springer Berlin Heidelberg, 2016, pp. 484–513.
18. Morris J Dworkin et al., *Sha-3 standard: Permutation-based hash and extendable-output functions*, (2015).
19. Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud, *Efficient and provable white-box primitives*, Advances in Cryptology – ASIACRYPT 2016 (Berlin, Heidelberg) (Jung Hee Cheon and Tsuyoshi Takagi, eds.), Springer Berlin Heidelberg, 2016, pp. 159–188.
20. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı, *Ls-designs: Bitslice encryption for efficient masked software implementations*, Fast Software Encryption (Berlin, Heidelberg) (Carlos Cid and Christian Rechberger, eds.), Springer Berlin Heidelberg, 2015, pp. 18–37.

21. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof, *Scream & iscream side-channel resistant authenticated encryption with masking*, (2015).
22. Computer Security Resource Center Information Technology Laboratory, *Lightweight cryptography*, https://csrc.nist.gov/Projects/lightweight-cryptography.
23. Matthias Jacob, Dan Boneh, and Edward Felten, *Attacking an obfuscated cipher by injecting faults*, ACM Workshop on Digital Rights Management, Springer, 2002, pp. 16–31.
24. Anthony Journault, François-Xavier Standaert, and Kerem Varıcı, *Improving the security and efficiency of block ciphers based on ls-designs*, Designs, Codes and Cryptography, 2017, pp. 495–509.
25. Mohamed Karroumi, *Protecting white-box aes with dual ciphers*, Information Security and Cryptology - ICISC 2010 (Berlin, Heidelberg) (Kyung-Hyune Rhee and DaeHun Nyang, eds.), Springer Berlin Heidelberg, 2011, pp. 278–291.
26. Yuji Koike and Takanori Isobe, *Yoroi: Updatable whitebox cryptography*, IACR Transactions on Cryptographic Hardware and Embedded Systems **2021** (2021), no. 4, 587–617.
27. Yuji Koike, Kosei Sakamoto, Takuya Hayashi, and Takanori Isobe, *Galaxy: A family of stream-cipher-based space-hard ciphers*, Information Security and Privacy (Cham) (Joseph K. Liu and Hui Cui, eds.), Springer International Publishing, 2020, pp. 142–159.
28. Jihoon Kwon, Byeonghak Lee, Jooyoung Lee, and Dukjae Moon, *Fpl: White-box secure block cipher using parallel table look-ups*, Topics in Cryptology – CT-RSA 2020 (Cham) (Stanislaw Jarecki, ed.), Springer International Publishing, 2020, pp. 106–128.
29. Gregor Leander, Brice Minaud, and Sondre Rønjom, *A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro*, Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, Springer, 2015, pp. 254–283.
30. Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel, *Two attacks on a white-box aes implementation*, Selected Areas in Cryptography – SAC 2013 (Berlin, Heidelberg) (Tanja Lange, Kristin Lauter, and Petr Lisoněk, eds.), Springer Berlin Heidelberg, 2014, pp. 265–285.
31. Jun Liu, Vincent Rijmen, Yupu Hu, Jie Chen, and Baocang Wang, *Warx: efficient white-box block cipher based on arx primitives and random mds matrix*, Science China Information Sciences (2021), 1869–1919, https://doi.org/10.1007/s11432-020-3105-1.
32. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel, *Differential and linear cryptanalysis using mixed-integer linear programming*, Information Security and Cryptology (Berlin, Heidelberg) (Chuan-Kun Wu, Moti Yung, and Dongdai Lin, eds.), Springer Berlin Heidelberg, 2012, pp. 57–76.
33. Yosuke Todo, Gregor Leander, and Yu Sasaki, *Nonlinear invariant attack: Practical attack on full scream, i scream, and midori 64*, 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 II 22, Springer, 2016, pp. 3–33.
34. Yaying Xiao and Xuejia Lai, *A secure implementation of white-box aes*, 2009 2nd International Conference on Computer Science and its Applications, IEEE, 2009, pp. 1–6.

Institute of Applied Mathematics, Middle East Technical University, 06800, Çankaya, Ankara, Turkey
Email address: hatice.kubra.89@gmail.com

Institute of Applied Mathematics, Middle East Technical University, 06800, Çankaya, Ankara, Turkey
Email address: ceyda.mangir@alumni.metu.edu.tr

Institute of Applied Mathematics, Middle East Technical University,, 06800, Çankaya, Ankara, Turkey
Email address: oguz@metu.edu.tr