A STUDY ON SPACE-HARD WHITE-BOX CRYPTOGRAPHY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HATİCE KÜBRA GÜNER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

JULY 2023

Approval of the thesis:

A STUDY ON SPACE-HARD WHITE-BOX CRYPTOGRAPHY

submitted by **HATİCE KÜBRA GÜNER** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Kestel _____
Dean, Graduate School of **Applied Mathematics**

Assoc. Prof. Dr. Oğuz Yayla _____
Head of Department, **Cryptography**

Assoc. Prof. Dr. Oğuz Yayla _____
Supervisor, **Cryptography, METU**

Dr. Ceyda Mangır _____
Co-supervisor, **Cryptographer, Ankara**

Examining Committee Members:

Prof. Dr. Sedat Akleylek
Computer Engineering Department, OMU _____

Assoc. Prof. Dr. Oğuz Yayla
Cryptography Department, METU _____

Prof. Dr. Murat Cenk
Cryptography Department, METU _____

Prof. Dr. Zülfükar Saygı
Mathematics Department, TOBB _____

Assoc. Prof. Dr. Fatih Sulak
Mathematics Department, Atılım University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: HATİCE KÜBRA GÜNER

Signature :

ABSTRACT

A STUDY ON SPACE-HARD WHITE-BOX CRYPTOGRAPHY

GÜNER, HATİCE KÜBRA
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz Yayla
Co-Supervisor : Dr. Ceyda Mangır

July 2023, 59 pages

Protecting secret keys from malicious observers is a major problem for cryptographic algorithms in untrusted environments. White-box cryptography suggests hiding the key in the cipher code with an appropriate method such that extracting the key becomes impossible in the white-box setting. The key is generally embedded into the confusion layer with suitable methods. One of them is using encoding techniques. Nevertheless, many encoding methods are vulnerable to algebraic attacks and side-channel analysis. Another is the space hardness concept, which creates large lookup tables that cannot be easily extracted from the device. In (M,Z)-space hard algorithms, the secret key is embedded in large tables created as a substitution box with a suitable block cipher. So the key extraction problem in the white-box setting turns into a key recovery problem in the black-box case. One of the main issues in (M,Z)-space hard algorithms is accelerating the run-time of the white-box/black-box implementation. In this study, we aim to use the advantage of the efficiency of lightweight components to speed up the diffusion layer of white-box algorithms without decreasing the security size. Therefore, we compare the linear layer of NIST Lightweight Standardization candidates for efficiency and suitability to white-box settings in existing space hard ciphers. The performance results of the algorithms are compared with WARX and SPNbox. According to the results, using the lightweight components in the diffusion layer accelerates the performance of white-box algorithms by at least 16%. Additionally, we propose an LS-design based white-box algorithm with better run-rime

performance and an LS-design based table creation method to take advantage of the bitslice implementation against side-channel attacks. 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. At the same time, in the white-box setting, the LS-design based method is also implemented to the 256-bit block size.

Keywords: White-box Cryptography, Software Protection, Space-hard Ciphers, Lightweight Components, Efficiency.

ÖZ

UZAY-ZOR BEYAZ KUTU KRİPTOGRAFİSİ ÜZERİNE BİR ÇALIŞMA

GÜNER, HATİCE KÜBRA
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz Yayla
Ortak Tez Yöneticisi : Dr. Ceyda Mangır

Temmuz 2023, 59 sayfa

Gizli anahtarları kötü niyetli gözlemcilerden korumak, güvenilmeyen ortamlarda kriptografik algoritmalar için büyük bir sorundur. Beyaz kutu kriptografisi, beyaz kutu ayarında anahtarın çıkarılması imkansız hale gelecek şekilde uygun bir yöntemle şifre kodundaki anahtarın saklanmasını önerir. Anahtar genellikle uygun yöntemlerle karıştırma katmanına gömülür. Bunlardan biri kodlama teknikleri kullanmaktır. Bununla birlikte, birçok kodlama yöntemi cebirsel saldırılara ve yan kanal analizine karşı savunmasızdır. Bir diğeri, cihazdan kolayca çıkarılamayan büyük arama tabloları oluşturan alan sabitliği konseptidir. (M,Z)-uzay sabit algoritmalarında, gizli anahtar, uygun bir blok şifre ile bir ikame kutusu olarak oluşturulan büyük tablolara gömülür. Böylece, beyaz kutu ayarındaki anahtar çıkarma sorunu, kara kutu durumunda bir anahtar kurtarma sorununa dönüşür. (M,Z)-uzayı sabit algoritmalarındaki ana sorunlardan biri, beyaz kutu/kara kutu uygulamasının çalışma zamanını hızlandırmaktır. Bu çalışmada, güvenlik boyutunu düşürmeden beyaz kutu algoritmalarının difüzyon katmanını hızlandırmak için hafif bileşenlerin etkinliğinin avantajını kullanmayı amaçlıyoruz. Bu nedenle, NIST Hafif Standardizasyon adaylarının doğrusal katmanını verimlilik ve mevcut uzay sabit şifrelerindeki beyaz kutu ayarlarına uygunluk açısından karşılaştırıyoruz. Algoritmaların performans sonuçları WARX ve SPNbox ile karşılaştırılmıştır. Sonuçlara göre, difüzyon katmanında hafif bileşenlerin kullanılması, beyaz kutu algoritmalarının performansını en az $\%16$ hızlandırıyor. Ek olarak, yan kanal saldırılarına karşı bitslice uygulamasından yararlanmak için daha

iyi çalışma zamanı performansına sahip LS tasarımı tabanlı bir beyaz kutu algoritması ve LS tasarımı tabanlı tablo oluşturma yöntemi öneriyoruz. Metodumuzun çalışma zamanı performansını SPNbox algoritması ile karşılaştırdığımızda, beyaz kutu uygulaması için $\%28$ ve kara kutu uygulaması için $\%27$ iyileştirme elde ediyoruz. Aynı zamanda, beyaz kutu ayarında, LS tasarımına dayalı yöntemı 256 bitlik blok boyutuna da uyguluyoruz.

Anahtar Kelimeler: Beyaz Kutu Kriptografisi, Yazılım Koruması, Uzay-Zor Şifreler, Hafif Bileşenler, Verimlilik

To my mother

ACKNOWLEDGMENTS

I would like to indicate my appreciation to my thesis supervisor Assoc. Prof. Dr. Oğuz Yayla for his guidance, valuable advices, and the encouragements during the long thesis period. His experiences have guided me in overcoming the hurdles I have encountered.

I would like to express my gratitude to my thesis co-supervisor Dr. Ceyda Mangır for her endless support, advices and consideration of all the details with me. I am grateful to her for encouraging me during the study and patient guidance in my troubles.

I would like to express my deep grateful to my family for making me feel that they are always by my side with their love and endless support. Their sincere encouragement brightens my life.

I would like to thank all my friends who were with me with their interest and relevance, and who did not withhold their support from me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBRV	Abbreviation
AES	Advanced Encryption Standard
ARX	Addition-Rotation-XOR
ASASA	Affine-Substitution-Affine-Substitution-Affine
DES	Data Encryption Standard
LS	Lbox-Sbox
LTS	Long Trail Strategy
MAS	Maximum Achievable Security
MLC	Maximum Linear Correlation
MDP	Maximum Differential Probability
MDS	Maximum Distance Separable
NIST	National Institute of Standards and Technology
SAC	Strict Avalanche Criteria
SIMD	Single Instruction Multiple Data
WEM	White-Box Even-Mansour
WTS	Wide Trail Strategy

CHAPTER 1

INTRODUCTION

Products used in an untrusted environment are vulnerable to capturing encryption keys by a malicious observer, as the observer has the ability to gain access to the cryptographic algorithms and the encryption keys. From DRM products and cloud servers to endpoint users such as mobile phones, laptops, or lightweight devices require protection against third parties. White-box cryptography suggests software protection using an appropriate method to hide the key in the algorithm phases. The key is generally embedded into the confusion layer with suitable methods. According to Delerablée et al. [22], the security primitives of white-box implementations are unbreakability, one-wayness, incompressibility, and traceability. One-wayness is required to avoid decryption by inverting the white-box algorithm. Traceability is needed to prevent unauthorized usage of the white-box algorithm. Incompressibility is stated as one of the primitive properties of the white-box algorithms to avoid extraction of the embedded key from the device.

The first white-box algorithms, white-box DES [16] in 2002 and white-box AES [15] in 2003, were proposed by Chow et al. The secret key was embedded into the Sbox by transforming the algorithm layers into lookup tables with internal and external encodings. The suggested encoding methods to prevent key extraction were broken by the algebraic attacks [5, 30]. Some other white-box AES variants [49, 33] were proposed but were also broken [21, 39].

A dedicated white-box block cipher based on ASASA structure was proposed by Biryukov et al. [6]. The paper defines weak white-box security and offers a memory-hard white-box block cipher against code-lifting attacks. In code-lifting attacks, an

attacker uses the original implementation as a large secret key for encryption and decryption on a different device. Unfortunately, key recovery attacks were applied in [26, 43] against the ASASA structure. Even if the structure was broken, the proposed methods inspired space-hard approaches for white-box algorithms.

After DES, AES, and ASASA white-box block cipher designs, a new dedicated algorithm called space-hard ciphers has been proposed by Bogdanov and Isobe [9, 10]. The secret key is hidden in large lookup tables created with a small block cipher in the new structure. The constructed tables are used as a nonlinear layer in the algorithm. With this approach, key extraction in the white-box setting becomes a key recovery problem in the black-box case. Against code lifting attacks, (M, Z)-space hardness is defined such that the algorithm provides Z-bit security until the size of the leakage from the code (table) is reached M bits.

Two main issues exist for (M, Z)-space hard white-box algorithms. The first one is updating the lookup tables after a particular leakage to provide security against code-lifting attacks. Generally, the security size of the algorithms is limited to leaking $\frac{1}{4}$ of the lookup tables [9, 10, 14, 34, 42]. When the leak limit is reached, the tables must be updated, either on the device or by remotely loading from the server [34].

The other issue is accelerating the black-box and the white-box implementations. The encryption/decryption process of the space-hard ciphers is mainly based on Feistel [9, 25, 36, 35] and SPN [10, 34, 42] structures. The black-box implementations become costly since a small-scale block cipher is used in the nonlinear layer. While it is essential to speed up the nonlinear layer on the black-box side, in the white-box implementation, the nonlinear layer only takes a value from the lookup table. When examining SPN-based white-box implementations, the most time-consuming part is the linear layer with the MDS matrix, which consists of matrix multiplication and modular reduction. Hence, speeding up the white-box and even the black-box implementations depend on the linear layer. Among the (M, Z)-space hard ciphers, SPNbox aims to improve performance against Feistel-based space-hard cipher Space [9] by taking advantage of parallelism and single instruction multiple data (SIMD) instructions. Another space-hard cipher WARX [42], is proposed with the motivation that existing white-box implementations run slowly or need ample storage space. WARX

improves the performance by decreasing the round number of the algorithm through a random MDS matrix in the linear layer.

Lightweight algorithms are resource-constraint designs with heavyweight security and fewer computational requirements. Some of the recent lightweight algorithms offer new design ideas, including getting rid of the computational cost of the MDS matrix [23, 45]. This study aims to propose a new space-hard white-box algorithm that uses lightweight components other than MDS matrices used in the current algorithms. With this motivation, the linear components of the NIST Lightweight competition [19] candidates were examined in the (M, Z)-space hard white-box setting for efficiency without reducing the security level. The linear components of Spook [4] were found suitable for an efficient space-hard white-box algorithm. A new LS-design [27] based (M, Z)-space hard algorithm using Spook's linear components is implemented in this paper to accelerate the run-time performance of the existing one. Also, a new table creation method based on LS-design is proposed to take advantage of the bitslice implementation against side-channel analysis.

This thesis is organized as follows:

Chapter 2 discusses SPN-based space hard ciphers. The aim of the white-box implementations and specifications of the algorithms are defined.

In Chapter 3, the linear layers of NIST Lightweight competition finalists and second-round candidates are examined to evaluate suitable designs for (M, Z)-space hard white-box algorithms. The lookup table is used as a substitution box in the nonlinear layer, and the linear components of the lightweight designs are used as a linear layer in the white-box conversions. The tables are created using the WARX method for 16-bit word size algorithms and the SPNbox method for 32-bit word size algorithms. We suggest fixing the security size to $leaksize \cdot 2^{-keysize}$ bits to more precisely calculate the round numbers of the algorithms based on the recommended security level. The run-time of the white-box schemes is compared with the algorithms WARX and SPNbox-32. We observed that lightweight components in the white-box setting are faster than WARX and SPNbox-32, with appropriate round numbers and security sizes.

In Chapter 4, a new space-hard white-box algorithm and a new table construction method based on the LS-design are proposed. The linear layer of the white-box algorithm is taken from the Spook design. In the table creation method, the S-box of Scream-v3 Algorithm [28] is used in the nonlinear layer, and the linear layer is taken from an LS-design based lightweight algorithm Mysterion [31]. According to the implementation results, the new white-box/black-box implementations accelerate the run-time performance compared to the SPNbox-32's for 128-bit block size.

CHAPTER 2

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. The nonlinear layer of the white-box algorithm is created with this small block cipher. Thus, the secret key is embedded in the nonlinear layer and the nonlinear layer is used as the substitution box. In this case, the key extraction issue in the white-box setting becomes a key recovery issue in the black-box setting.

In this chapter, the first space hard cipher Space and current SPN-based space hard algorithms are examined. Details of the algorithms are specified in Section 2.1. The white-box and black-box security considerations are stated in Section 2.2.

2.1 Specifications of the Algorithms

2.1.1 Space

The first (M, Z)-space hard white-box algorithm proposed by Bogdanov and Isobe is the Space [9] based on the Feistel structure, which uses the nonlinear layer *F* function as a lookup table. The lookup table is created using AES components to hide the secret key. The number of round is recommended as at least 128 to obtain desired space-hard security level. The weak (M, Z)-space hardness is defined as a security concept for space hard white-box algorithms.

Definition 1 (Weak (M, Z)-space hardness [9])**.** *A white-box block cipher is called weak (M, Z)-space hard if it is not possible to encrypt/decrypt a randomly selected*

text with a probability greater than 2^{-Z} until the size of the leakage from the code (table) is reached to M bits.

In this approach, the table must be updated when the leak size reaches the defined limit. The security against obtaining a valid plaintext/ciphertext pair is defined with strong (M, Z)-space hardness. In the strong (M, Z)-space hardness criteria, an attacker aims to obtain a valid plaintext/ciphertext pair instead of obtaining ciphertext for an arbitrary plaintext. The table must be updated when the defined leakage limit is reached.

Definition 2 (Strong (M,Z)-space hardness [9])**.** *A white-box block cipher is called strong (M,Z)-space hard if it is not possible to encrypt/decrypt a valid text pair with a probability greater than 2^{-Z} until the size of the leakage from the code (table) is reached to M bits.*

2.1.2 SPNbox

The space-hard white-box algorithm SPNbox [10] is designed by Bogdanov A. et al. The SPNbox aims to improve performance against Space using parallelism and SIMD instructions. It is based on SPN structure using key-dependent lookup table T in the nonlinear layer. The table T is generated using a small-scale block cipher with AES components.

2.1.2.1 Table Construction

The key dependent substitution box T is constructed with a small SPN-type block cipher. The table generation method consists of AES components with n_{in} block size, and state as $x = (x_0, ..., x_{l-1})$ where $l = n_{in}/8$. In the implementation, the round keys are generated using a reliable key derivation function such as SHAKE [24]. One round of the algorithm is

$$T_{n_{in}} : GF(2^8)^l \rightarrow GF(2^8)^l$$

$$x^r \rightarrow (RK^r \circ MC_{n_{in}} \circ SBox)(x^{r-1})$$

The keys are xored with the state in the RK^r layer. The MDS matrix of AES is used at the $MC_{n_{in}}$ layer to provide diffusion in the small block cipher. The MDS matrices $A_{n_{in}}$ for n_{in} = 16, 24, 32 are recommended as

- A_{16}= $\begin{bmatrix} 2, 1 \\ 3, 2 \end{bmatrix}$
- A_{24}= $\begin{bmatrix} 2 & 1 & 1 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}$
- $A_{32} = cir(2, 1, 1, 3)$

The identity matrix is used in the diffusion layer for n_{in} = 8. Confusion is provided with AES S-box in the nonlinear $Sbox$ layer. The numbers of rounds are specified as r_8 = 64, r_{16} = 32, r_{24} = 20, r_{32} = 16. All elements of the field $2^{n_{in}}$ are encrypted by the table creation method, and the size of the key dependent table T is $n_{in} \cdot 2^{n_{in}}$ bits. The table creation method is given in Algorithm 1.

Algorithm 1 T-Table generation algorithm of SPNbox

1: **INPUT**: Round keys $(k_0, k_1, ..., k_{r_{n_{in}}}), k_i \in \mathbb{F}_2^{n_{in}}$
2: **OUTPUT**: x
3: **for** $x = 0$ to $2^{n_{in}} - 1$ **do**
4: $x \leftarrow x \oplus k_0$
5: **for** $i = 1$ to $r_{n_{in}}$ **do**
6: $x \leftarrow Sbox(x)$
7: $x \leftarrow x \cdot A_{n_{in}}$
8: $x \leftarrow x \oplus k_i$
9: **end for**
10: **end for**

2.1.2.2 The Algorithm

SPN structure based SPNbox family consists of SPNbox-n_{in} members with *n*-bit block size, n_{in}-bit word size (n_{in} = 8, 16, 24, 32), and *k*-bit key. The SPNbox-24

has a 120-bit block length and key size, while the remaining algorithms have 128-bit block and key length. SPNbox-n_{in}'s state vector is defined as $x^r = (x_0^r, ..., x_{t-1}^r)$ where r represents the related round and $t = n/n_{in}$. The encryption process consists of 10 rounds with the nonlinear γ, the linear θ, and the affine σ^r layers.

$$x^r = (\sigma^r \circ \theta \circ \gamma)(x^{r-1})$$

The nonlinear γ layer is a substitution layer using the generated key dependent table T. The linear layer θ consists of matrix multiplication with a $t \times t$ MDS matrix defined for cases n_{in} = 8, 16, 24, 32 as

- $M_8 = had(8, 16, 8a, 1, 70, 8d, 24, 76, a8, 91, ad, 48, 5, b5, af, f8)$
- $M_{16} = had(1, 3, 4, 5, 6, 8, b, 7)$
- $M_{24} = cir(1, 2, 5, 3, 4)$
- $M_{32} = cir(1, 2, 4, 6)$

M_{32} and M_{16} are the MDS matrices of the block ciphers Anubis [2] and Khazad [1], respectively. M_8 is an optimized Hadamard-Cauchy matrix [40]. The affine layer σ^r applies xor with the related round constant defined as

$$c_i^r = (r-1) \cdot t + i + 1 \; for \; 0 \leq i \leq t-1$$

The pseudo-codes of the white-box and black-box implementations of the SPNbox are given in Algorithm 2 and Algorithm 3.

2.1.3 Yoroi

Another SPN-based space-hard white-box algorithm called Yoroi was designed by Koike and Isobe [34]. The motivation behind the Yoroi design is to eliminate the need to re-encrypt data on the server side after updating the table in the SPNbox. Therefore, longevity is defined as securing the white-box algorithm without changing the master key of the lookup table and updating the table only on the client side to prevent code-lifting attacks. (Z)-longevity parameter is considered instead of (M, Z)-space hardness criterion.

Algorithm 2 WBI of SPNbox

INPUT: $x_i^0 \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}$, T-Table
OUTPUT: $x_i^r \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}$
for $r = 1$ to *R* **do**
  for $i = 0$ to $t-1$ **do**
    $x_i^r \leftarrow T(x_i^r)$
  end for
  $(x_0^r, \cdots, x_{t-1}^r) \leftarrow M_{n_{in}} \cdot (x_0^r, \cdots, x_{t-1}^r)$
  for $i = 0$ to $t-1$ **do**
    $c_i^r \leftarrow (r-1) \cdot (t) + i + 1$
    $x_i^r \leftarrow x_i^r \oplus c_i^r$
  end for
end for

Algorithm 3 BBI of SPNbox

INPUT: $x_i^0 \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}, k_j \in \mathbb{F}_2^{n_{in}},\ j \in \{0, \cdots, r_{n_{in}}\}$
OUTPUT: $x_i^r \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}$
for $r = 1$ to *R* **do**
  for $i = 0$ to $t-1$ **do**
    $x_i^r \leftarrow x_i^r \oplus k_0$
    for $j = 1$ to $r_{n_{in}}$ **do**
      $x_i^r \leftarrow Sbox(x_i^r)$
      $x_i^r \leftarrow x \cdot A_{n_{in}}$
      $x_i^r \leftarrow x \oplus k_j$
    end for
  end for
  $(x_0^r, \cdots, x_{t-1}^r) \leftarrow M_{n_{in}} \cdot (x_0^r, \cdots, x_{t-1}^r)$
  for $i = 0$ to $t-1$ **do**
    $c_i^r \leftarrow (r-1) \cdot (t) + i + 1$
    $x_i^r \leftarrow x_i^r \oplus c_i^r$
  end for
end for

Definition 3 ((Z)-longevity)**.** *A cryptographic function that retains the same functionality while leaking to the adversary is called (Z)-longevity if, in terms of computational cost, the probability of correctly encrypting/decrypting a randomly selected text is higher than* $2^{(-Z)}$.

Similar to the idea in the (M, Z)-space hardness concept, the table needs to be updated after the table leak reaches the limit on the client side. The longevity is achieved by applying encryption of another small block cipher to the output of the lookup table. This additional small-scale block cipher is applied to u-bit of the n_{in}-bit entries of the table where $u < n_{in}$. When the table update is required, the key of the added small-scale block cipher is changed while the master key of the original table remains the same.

The Yoroi structure differs from the SPNbox design by using a partial MDS matrix in the white-box algorithm. Since encryption and decryption are applied sequentially in the table creation method, a partial MDS matrix is required to ensure consistency between rounds. Three lookup tables are used on the client side.

2.1.3.1 The Table Construction

The Yoroi differs from the construction of the lookup table as another small-scale block cipher is added to the existing lookup table in the white-box model. A small-scale block cipher E_K and D_K is added to the SPNbox's method without breaking the black-box side's consistency. The tables T_1, T_2 and T_3 are constructed using the T table such as

$$T_1 = E_K(msb_u(T(x)))||lsb_v(T(x)),$$

$$T_2 = E_K(msb_u(T(D_K(msb_u(x))||lsb_v(x))))||lsb_v(T(D_K(msb_u(x))||lsb_v(x))),$$

$$T_3 = T(D_K(msb_u(x))||lsb_v(x)).$$

The small-scale AES [17] or PRESENT [37] can be used in E_K block cipher. The table construction method is stated in Algorithm 4. In the white-box implementation, the key of E_K and D_K are changed to update Yoroi's substitution tables T_1, T_2, and T_3. There is no need to update the master key of the table T directly in the black-box implementation.

Algorithm 4 T-Tables generation algorithm of Yoroi

INPUT: T-table, small scale encryption key k
OUTPUT: T_1, T_2, T_3
// Generate T_1
while $x < 2^{n_{in}}$ **do**
    $x \leftarrow T(x)$
    $x \leftarrow E_K(msb_u(x))||lsb_v(x)$
end while
// Generate T_2
while $x < 2^{n_{in}}$ **do**
    $x \leftarrow D_K(msb_u(x))||lsb_v(x)$
    $x \leftarrow T(x)$
    $x \leftarrow E_K(msb_u(x))||lsb_v(x)$
end while
// Generate T_3
while $x < 2^{n_{in}}$ **do**
    $x \leftarrow D_K(msb_u(x))||lsb_v(x)$
    $x \leftarrow T(x)$
end while

2.1.3.2 The Algorithm

The Yoroi has a 128-bit block size, n_{in} = 16/32 bits word size, and a 128-bit key with 8/16 rounds. The state vector is defined as $x^r = (x_0^r, x_1^r, ..., x_{t-1}^r)$, where $t = 128/n_{in}$. Each element x_i^r is taken as $(msb_u(x_i^r)||lsb_v(x_i^r))$ where $u + v = n_{in}$ and $v = 4$. The Yoroi algorithm consists of the nonlinear γ_i, the affine σ_i, and the linear θ layers. In the last round, ten rounds of fixed-key AES are applied in the function A to achieve diffusion, as in WhiteBlock [25] and WEM [14] algorithms.

$$x^R = A \circ \gamma_R \circ (\bigcirc_{i=1}^{R-1}(\theta \circ \sigma_i \circ \gamma_i))(x^0).$$

The nonlinear γ_i consists of key-dependent n_{in}-bit T_1, T_2, and T_3 substitution boxes. In the first round, T_1, in the last round, T_3 and in the rest of the rounds, T_2 are applied. In the affine layer σ_i, the state is xored with constants $c_i = i$ for $1 \leq i \leq R -$

1. The linear layer θ consists of an $t \times t$ MDS matrix over $GF(2^v)$ applied to the state's least significant v bits. The pseudo-codes of the white-box and the black-box implementations are given in Algorithm 5 and Algorithm 6.

Algorithm 5 WBI of Yoroi

INPUT: $(x_i^0 \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}, T_1, T_2, T_3$
OUTPUT: $x_i^R \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}$
$(x_0^0, \cdots, x_{t-1}^0) \leftarrow (T_1(x_0^r), \cdots, T_1(x_{t-1}^r))$
$(x_1^0, \cdots, x_{t-1}^0) \leftarrow (x_0^0 \oplus c_0, \cdots, x_{t-1}^0 \oplus c_0)$
$(lsb(x_0^0), \cdots, lsb(x_{t-1}^0)) \leftarrow M_v \cdot (lsb(x_0^0), \cdots, lsb(x_{t-1}^0))$
for $r = 2$ to *R*-1 **do**
    $(x_0^r, \cdots, x_{t-1}^r) \leftarrow (T_2(x_0^r), \cdots, T_2(x_{t-1}^r))$
    $(x_0^r, \cdots, x_{t-1}^r) \leftarrow (x_0^r \oplus c_r, \cdots, x_{t-1}^r \oplus c_r)$
    $(lsb(x_0^r), \cdots, lsb(x_{t-1}^r)) \leftarrow M_v \cdot (lsb(x_0^r), \cdots, lsb(x_{t-1}^r))$
end for
$(x_0^R, \cdots, x_{t-1}^R) \leftarrow (T_3(x_0^R), \cdots, T_3(x_{t-1}^R))$
$(x_0^R, \cdots, x_{t-1}^R) \leftarrow A(x_0^R, \cdots, x_{t-1}^R)$

Algorithm 6 BBI of Yoroi

INPUT: $(x_i^0 \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}, T$
OUTPUT: $x_i^R \in \mathbb{F}_2^{n_{in}},\ i \in \{0, \cdots, t-1\}$
for $r = 1$ to *R* **do**
    $(x_0^r, \cdots, x_{t-1}^r) \leftarrow (T(x_0^r), \cdots, T_(x_{t-1}^r))$
    $(x_0^r, \cdots, x_{t-1}^r) \leftarrow (x_0^r \oplus c_r, \cdots, x_{t-1}^r \oplus c_r)$
    $(lsb(x_0^r), \cdots, lsb(x_{t-1}^r)) \leftarrow M_v \cdot (lsb(x_0^r), \cdots, lsb(x_{t-1}^r))$
end for
$(x_0^R, \cdots, x_{t-1}^R) \leftarrow (T_(x_0^R), \cdots, T_(x_{t-1}^R))$
$(x_0^R, \cdots, x_{t-1}^R) \leftarrow A(x_0^R, \cdots, x_{t-1}^R)$

2.1.4 WARX

The space-hard white-box algorithm WARX [42] is proposed by Liu J. et al. with the motivation that existing white-box implementations run slowly or need ample storage space. Therefore, WARX proposes using ARX (Addition/Rotation/XOR) ap-

proach [23] to generate the lookup table used in the nonlinear layer. Additionally, speeding up the run-time performance of the white-box implementation is suggested by reducing the number of round by one according to SPNbox-16 [10] and WEM [14] with the random MDS matrix recommendation.

2.1.4.1 Table Construction

The lookup table T used in the nonlinear layer is generated with mSPARX-16 stated in Algorithm 7, inspired by SPARX algorithm [23]. It consists of 24 rounds with xoring the round keys and output of ARX-based mSPECKEY stated in Algorithm 8. The round keys are generated using a trusted key derivation function.

Algorithm 7 mSPARX-16

1: INPUT: $x \in \mathbb{F}_2^{16}$, round keys $(k_0, ..., k_{23})$
2: OUTPUT: $x \in \mathbb{F}_2^{16}$
3: $x \leftarrow x \oplus k_0$
4: **for** $i = 0$ to 23 **do**
5: $\quad x \leftarrow mSPECKEY(x) \oplus k_i$
6: **end for**

Algorithm 8 mSPECKEY

1: **INPUT**: Plaintext x
2: **OUTPUT**:
3: $l \leftarrow lsb(x, 8)$
4: $m \leftarrow msb(x, 8)$
5: $l \leftarrow l \ggg 7$
6: $l \leftarrow l \boxplus m$
7: $m \leftarrow m \lll 2$
8: $m \leftarrow m \oplus l$
9: **RETURN**: $l||m$

2.1.4.2 The Algorithm

WARX is an SPN-based white-box block cipher with a 16-bit word size, 128-bit key length, and seven rounds. Each round consists of nonlinear, linear, and constant addition layers. The random MDS matrix used in the linear layer is generated by multiplying a well-designed MDS matrix with a randomly generated diagonal matrix.

$$M_{rand} = diag(r_{c_0}, ..., r_{c_7}) \cdot MDS$$

where $r_i \in 2^{16}$ and randomly generated. The MDS matrix of lightweight Khazad algorithm [1] is used to generate the random matrix in WARX. The pseudo-codes of the white-box and the black-box implementations of WARX are specified in Algorithm 9 and Algorithm 10.

Algorithm 9 WBI of WARX

1: INPUT: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 7\}$, T-Table
2: OUTPUT: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 7\}$
3: **for** $r = 7$ to 1 **do**
4: $\quad (x_0, \cdots, x_7) \leftarrow (x_0, \cdots, x_7) \bigoplus (8(r-1)+1, 8(r-1)+2, ..., 8(r-1)+8)$
5: $\quad (x_0, \cdots, x_7) \leftarrow (x_0, \cdots, x_7) \cdot (M_{rand}^{-1})^T$
6: $\quad (x_0, \cdots, x_7) \leftarrow (T(x_0), T(x_1), ..., T(x_7))$
7: **end for**

The attack types to white-box implementations are detailed since the WARX structure has randomness in the linear layer, unlike other space-hard ciphers. Moreover, guessing the unknown part of the lookup table is added to the success probability of the (M, Z)-space hardness criteria. Therefore, the round number of the white-box algorithms is computed more accurately.

2.1.5 Other Space-Hard Ciphers

Other than these (M, Z)-space hard algorithms, Feistel-based WhiteBlock [25], FPL [36] and Galaxy [35], Even-Mansur structure based WEM [14] were proposed to accelerate run-time performance while keeping the security strength of the white-box algorithms.

Algorithm 10 BBI of WARX

INPUT: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 7\}$, $k_j \in \mathbb{F}_2^{16},\ j \in \{0, \cdots, 23\}$
OUTPUT: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 7\}$
for $r = 7$ to 1 **do**
    $(x_0, \cdots, x_7) \leftarrow (x_0, \cdots, x_7) \bigoplus (8(r-1)+1, 8(r-1)+2, ..., 8(r-1)+8)$
    $(x_0, \cdots, x_7) \leftarrow (x_0, \cdots, x_7) \cdot (M_{rand}^{-1})^T$
    for $i = 0$ to 7 **do**
        $x_i \leftarrow x_i \oplus k_0$
        for $j = 0$ to 23 **do**
            $x_i \leftarrow mSPECKEY(x_i) \oplus k_j$
        end for
    end for
end for

2.2 Security

The security of the algorithms is considered for white-box and black-box attacks, respectively.

2.2.1 White-box Security

The white-box security of an algorithm is evaluated by its resistance to key extraction from the lookup table and by its inability to use the table as a large key outside of the white-box environment called code lifting attacks [9].

2.2.1.1 Key Extraction Security

In the space-hard ciphers, extracting the secret key from the white-box structure turns into recovering the key from the lookup table in black-box setting [9] since the secret key is embedded into the table. Therefore, the algorithm is as resistant to key extraction attacks as the reliability of the table creation method. A malicious observer should not be able to obtain the secret key from the table values used in the internal steps of encryption or from the leaked portion of the table.

The table generation method of SPNbox consists of AES components with 16 rounds. The differential probability of the S-box is 2^{-6} and the linear correlation value is 2^{-3}. The number of branches of the MDS matrix is 5. The minimum number of active S-boxes was calculated using the method described by Mouha et al. in [44]. According to the active S-boxes number in Table 2.1, the table creation method provides 128-bit security with ten rounds against key extraction attacks.

Table 2.1: Active S-box number of table creation method of SPNbox-32.

round	1	2	3	4	5	6	7	8	9
Active Sbox	1	5	6	10	11	15	16	20	21
round	10	11	12	13	14	15	16	17	18
Active Sbox	25	26	30	31	35	36	40	41	45

The base table creation method of the Yoroi is the same with the SPNbox's method. Therefore. key extraction security of Yoroi is depends on the security strength of the using additional small block cipher to construct T_1, T_2, and T_3 tables as stated in the article [34].

WARX's table creation method is similar to the ARX-box of the SPARX algorithm [23]. The optimal differential characteristic probability is 2^{-19} and the optimal linear trail is 2^{-9} after 9 rounds as detailed in the article [42]. Therefore, the table is secure against single trail differential and linear cryptanalysis with 24 rounds.

2.2.1.2 Code Lifting Security

Resistance to code-lifting attacks is a crucial security measurement for space-hard ciphers since lookup tables are used as a large key in the design. Incompressible tables are needed to limit leakage of the code (table) and prevent table decomposition and code lifting attacks [22]. Code lifting security is defined with weak (M, Z)-space hardness in [9].

The space hardness attack types are discussed as known-space attacks, chosen-space attacks, and adaptively-chosen-space attacks in [10] according to the attacker's control capacity on the white-box environment. The success probability 2^{-Z} of space hardness is computed according to Theorem 1 for known-space and chosen-space

attack types.

Theorem 1. *[9] For the size M of the table T is known from an adversary, a randomly chosen text can be encrypted with a success probability* $(M/T)^{t \cdot R}$*, where* t *is the number of table entries in a round and* R *is the round number.*

Space hardness is specified for the Yoroi algorithm with Theorem 2 by including three tables used in the white-box implementation.

Theorem 2. *[34] For the size* M_1, M_2 *and* M_3 *of tables* T_1, T_2 *and* T_3 *are known from an adversary, a randomly chosen text can be encrypted with a success probability*

$$(\frac{M_1}{T_1})^t \times (\frac{M_2}{T_2})^{t \cdot (R-2)} \times (\frac{M_3}{T_3})^t$$

From the longevity property in Yoroi, the tables are updated without changing the least significant t-bit. Therefore, the success probability of space hardness is intuitively proven in the article according to the updated tables.

The attack types in WARX are detailed as a cipher, linear layer, nonlinear layer, and hybrid attacks according to the leakage of code and table segments since a random MDS matrix is used in the linear layer. Hybrid attacks consist of a certain number of combinations of each identified attack. When computing space hardness success probability, the white-box security level is limited to $tablesize \cdot 2^{blocksize}$ bits. Additionally, correctly guessing the unknown table entries is included in the probability of encrypting the randomly selected plaintext.

2.2.2 Black-box Security

The black-box security of the block ciphers is important against key recovery attacks, as the same key is used until the table is updated.

2.2.2.1 Differential Cryptanalysis

To compute the security strength of a block cipher against differential attacks, firstly maximum differential probability of the nonlinear layer is needed. The differential

probability of $T : 2^{n_{in}} \rightarrow 2^{n_{in}}$ is defined as

$$DP(\alpha, \beta) = 2^{-n_{in}} \cdot \#\{x \mid T(x) \oplus T(x \oplus \alpha) = \beta\}$$

where $\alpha, \beta \in 2^{n_{in}}$. The maximum differential probability (MDP) is taken as

$$\max_{\forall(\alpha,\beta)} DP(\alpha, \beta)$$

The MDP values for SPNbox-n_{in} are stated according to paper [29], as 2^{-4}, 2^{-11}, $2^{-18.42}$, and 2^{-26} for $n_{in} = 8, 16, 24, 32$ word sizes, respectively. The branch number of the used MDS matrix is five, and the desired security is provided after four rounds for $n_{in} = 8, 16, 24$ and after two rounds for $n_{in} = 32$.

The maximum differential probability of the table WARX is given as 2^{-19}, and 9 T-box is active after two rounds using the MDS matrix. Therefore, WARX is provided desired security level with seven rounds.

2.2.2.2 Linear Cryptanalysis

Similar to differential cryptanalysis, the maximum linear correlation is needed to determine the security level against linear attacks. The linear correlation of $T : 2^{n_{in}} \rightarrow 2^{n_{in}}$ is defined as

$$LC(\alpha, \beta) = (2^{1-n_{in}} \cdot \#\{x \mid \alpha \bullet x = \beta \bullet f(x)\} - 1)^2$$

where $\alpha, \beta \in 2^{n_{in}}$ and $\bullet$ represents the inner product. The maximum linear correlation (MLC) is obtained from

$$\max_{\forall(\alpha,\beta)} LP(\alpha, \beta)$$

The MLC values for SPNbox-n_{in} are stated in [10] as $2^{-3.67}$, $2^{-10.62}$, $2^{-18.02}$, and $2^{-2.61}$ for $n_{in} = 8, 16, 24, 32$ cases. The active T-box number is 51 after 6 rounds for $n_{in} = 8$, 18 after 4 rounds for $n_{in} = 16$, 12 after 4 rounds for $n_{in} = 24$, and 5 after 2 rounds for $n_{in} = 32$.

The WARX's substitution box has 2^{-9} MLP. Hence, the WARX provides expected resistance to linear cryptanalysis after two rounds with nine active T-boxes.

2.2.2.3 Slide Attacks

The slide attacks are based on the similarity of the rounds instead of the total round number of a block cipher. A different round constant is used for each round in SPNbox-n_{in}, Yoroi, and WARX to prevent slide attacks.

2.2.2.4 Algebraic Attacks

The idea of an algebraic attack comes from solving low-order or multivariate non-linear equations consisting of plaintext and ciphertext samples to obtain the key. Algebraic attacks are not applicable to the ciphers because of their T-box generation method.

CHAPTER 3

SPACE-HARD WHITE-BOX IMPLEMENTATIONS WITH THE LIGHTWEIGHT COMPONENTS

The linear layer is the most time-consuming part of a space-hard white-box algorithm. For this reason, we aim to observe the efficiency of linear lightweight components in the white-box setting. Ten finalists and 19 second-round candidates of the NIST Lightweight competition [19] were examined by compatibility of the space-hard white-box conditions. The nonlinear layer of the white-box conversions are utilized from WARX and SPNbox. Since the structure of these layers is based on 16-bit and 32-bit word sizes respectively, only the algorithms with 16/32 bits word sizes are taken into consideration. By the security reasons stated in Section 3.2.3, the linear components of Sparkle [3] from finalists, Spook [4], and Saturnin [13] among the second-round candidates are used in the performance comparisons.

Saturnin [13] is an SPN-based lightweight algorithm. It uses Sbox and MDS matrices to provide confusion and diffusion properties. Sparkle and Spook algorithms are permutation constructs with no key scheduling or key addition layer. Sparkle is based on ARX design consisting of addition, rotation, and xor operations [23]. The ARX design uses the long trail strategy (LTS) approach [23], based on the powerful built-in Sbox and the use of a linear layer with reduced computational cost. Spook has two permutations named Clyde and Shadow for different block sizes but uses the same bitslice implemented structure based on LS-design [27]. The permutation-based designs are well suited for generating (M, Z)-space hard white-box algorithms. They can be considered a more efficient linear layer than the existing space-hard ciphers with the desired white-box security.

In this chapter, linear components of the lightweight algorithms are examined in the white-box setting. Algorithm specifications are stated in Section 3.1. Round numbers of the white-box algorithms are computed according to the code lifting security criteria in Section 3.2. The run-time performance of 16-bit designs is compared with WARX, and 32-bit designs are compared with SPNbox-32 in Section 3.3.

3.1 Specifications of the White-box Algorithms

This section outlines the table creation method and white-box conversion features.

3.1.1 Table creation method

The lookup table used as a substitution box in a white-box context is created using a small-block cipher [10, 42]. Round keys of the small-block cipher are generated using a trusted extendable output function (XOF) with a secret key, and all values in the $2^{wordsize}$ field are encrypted with the cipher. The generated table is used as a substitution box to ensure nonlinearity in the white-box implementation. In any case, SHAKE [24] algorithm was used to generate the round keys. In the conversions, WARX's table creation method is used for 16-bit word size algorithms, and SPNbox's method is used for 32-bit algorithms. The table creation method of WARX 2.1.4 is based on ARX design and the table size is 128 KiB. The SPNbox's table creation method 2.1.2 uses AES components, and the table size is 16 GiB.

3.1.2 Saturnin

Saturnin [13] is a 256-bit block cipher with 16-bit word size and *super-rounds*, where even and odd rounds are implemented successively in each round. The state of Saturnin is taken as a 4×4 square in 16-bit words. In a super round of Saturnin, S_box and *MDS* layers are implemented in the even part. For the odd part, the layers are divided into two branches after applying S_box. If the round number is equivalent to 1 according to mod 4, the round consists of SR_slice, *MDS*, SR_slice_inv, adding round constants, and $XOR\_key\_rotated$ layers. In the other branch, the lay-

ers SR_slice and SR_slice_inv change to SR_sheet and SR_sheet_inv layers. In the linear layer, an MDS matrix is applied to 4 nibbles in parallel and defined as

$$M : GF(2^4)^4 \rightarrow GF(2^4)^4$$

$$(u_0, u_1, u_2, u_3) \rightarrow \begin{bmatrix} \alpha^2(u_0) \oplus \alpha^2(u_1) \oplus \alpha(u_1) \oplus u_2 \oplus u_3 \\ u_0 \oplus \alpha(u_1) \oplus u_1 \oplus \alpha^2(u_2) \oplus u_2 \oplus \alpha^2(u_3) \oplus \alpha(u_3) \oplus u_3 \\ u_0 \oplus u_1 \oplus \alpha^2(u_2) \oplus \alpha^2(u_3) \oplus \alpha(u_3) \\ \alpha^2(u_0) \oplus u_0 \oplus \alpha^2(u_1) \oplus \alpha(u_1) \oplus u_1 \oplus u_2 \oplus \alpha(u_3) \oplus u_3 \end{bmatrix}$$

The α transformation used in the matrix is taken as

$$\alpha : GF(2)^4 \rightarrow GF(2)^4$$

$$(v_0, v_1, v_2, v_3) \rightarrow (v_1, v_2, v_3, v_0 \oplus v_1)$$

Linear transformations SR_slice and SR_sheet are used to provide diffusion into 16-bit state words. In SR_slice, rotation is applied inside each 4-bit part of the state-word, while in SR_sheet, 4-bit parts of the state-word are rotated. The round constants RC0 and RC1 are generated using two different LFSR with feedback polynomials $x^{16} + x^5 + x^3 + x^2 + 1$ and $x^{16} + x^6 + x^4 + x + 1$.

3.1.2.1 White-box conversion

The linear layer and round constants of wSaturnin are taken from the lightweight design [13]. The $XOR\_key\_rotated$ layer is removed, and the lookup table and S_box layer are replaced. Also, instead of applying layers SR_slice and SR_sheet only in even rounds, SR_slice is applied on odd rounds and SR_sheet on even rounds. The wSaturnin is implemented in 8 rounds. The pseudo-code of wSaturnin is stated in Algorithm 11.

The computational cost of the ***MDS*** layer is $38 \cdot xor$ operations. SR_slice, and SR_slice_inv layers have $24 \cdot and + 12 \cdot or$ operations, respectively. Moreover, SR_sheet, and SR_sheet_inv layers have $12 \cdot or$ operations, respectively. There are $2 \cdot xor$ operations from adding round constants. Thus, computational cost of the wSaturnin is $320 \cdot xor + 192 \cdot or + 192 \cdot and$ operations.

Algorithm 11 wSaturnin.

Input: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 15\}$, T-Table
Output: $x_i \in \mathbb{F}_2^{16},\ i \in \{0, \cdots, 15\}$
for $i = 0$ to R-1 **do**
  for $j = 0$ to 15 **do**
    $x_j \leftarrow T(x_j)$
  end for
  if $((i\&1) == 0)$ **then**
    $(x_0, \cdots, x_{15}) \leftarrow SR\_slice((x_0, \cdots, x_{15})$
    $(x_0, \cdots, x_{15}) \leftarrow MDS(x_0, \cdots, x_{15})$
    $(x_0, \cdots, x_{15}) \leftarrow SR\_slice\_inv((x_0, \cdots, x_{15})$
  else
    $(x_0, \cdots, x_{15}) \leftarrow SR\_sheet((x_0, \cdots, x_{15})$
    $(x_0, \cdots, x_{15}) \leftarrow MDS((x_0, \cdots, x_{15})$
    $(x_0, \cdots, x_{15}) \leftarrow SR\_sheet\_inv((x_0, \cdots, x_{15})$
  end if
  $x_0 \leftarrow x_0 \oplus RC0[i]$
  $x_8 \leftarrow x_8 \oplus RC1[i]$
end for

3.1.3 Sparkle

The Sparkle permutation [3] is a combination of SPN and Feistel structures with 32-bit word size and 256/384/512-bit block size variants. The Sparkle permutation is created as an SPN cipher, while the layers are in a simple Feistel structure. It consists of a step counter layer, an ARX-based *Alzette* box for nonlinearity, and a linear diffusion layer $\mathcal{L}_w$ that swaps the branches. The linear transformation $\mathcal{L}_w$ is defined as

$$\mathcal{L}_w : GF(2^w)^l \rightarrow GF(2^w)^l$$

$$((x_0, y_0), \cdots, (x_{l-1}, y_{l-1})) \rightarrow ((u_0, v_0), \cdots, (u_{l-1}, v_{l-1}))$$

where (u_i, v_i) pairs are computed from the equations

$$u_i \leftarrow x_i \oplus \alpha(\bigoplus_{j=0}^{l-1} y_j),\ v_i \leftarrow y_i \oplus \alpha(\bigoplus_{j=0}^{l-1} x_j),\ i \in 0, \cdots, l-1$$

$$\alpha(t) \leftarrow (t \lll w/2) \oplus (t \,\&\, 0 \times FFFF)$$

3.1.3.1 White-box conversions

The 256-bit variant are examined for internal details of the white-box conversion. Although Sparkle is designed with a 32-bit word size, we also implement the 16-bit word size option when converting to the white-box setting. The *Alzette* box is discarded, and the constructed lookup tables are used in the nonlinear layer. The pre-defined round constants c_i are taken directly from the lightweight design. The white-box conversions are stated in Algorithm 12. The round numbers of wSparkle-16 and wSparkle-32 are computed as 8 and 14, respectively, in the white-box setting.

Algorithm 12 wSparkle-16/wSparkle-32.

Input: $(x_i, y_i) \in \mathbb{F}_2^w \times \mathbb{F}_2^w,\ i \in \{0, \cdots, l\},\ w = 16, 32,\ l = 256/w$, T-Table
Output: $(x_i, y_i) \in \mathbb{F}_2^w \times \mathbb{F}_2^w,\ i \in \{0, \cdots, l\}$
$(c_0, \cdots, c_l) \leftarrow algorithm\_constants$
for $i = 0$ to R-1 **do**
    $y_0 \leftarrow y_0 \oplus c_{(i \bmod 8)}$
    $y_1 \leftarrow y_1 \oplus i$
    for $i = 0$ to l **do**
        $(x_i, y_i) \leftarrow (T(x_i), T(y_i))$
    end for
    $(x_0, y_0), ..., (x_l, y_l) \leftarrow \mathcal{L}_w((x_0, y_0), ..., (x_l, y_l))$
end for

One round of wSparkle-16 consists of $2 \cdot xor$ operations from round constants, and $26 \cdot xor + 2 \cdot or$ operations from the linear layer. Hence, computational cost of the wSparkle-16 is $224 \cdot xor + 16 \cdot or$ operations. Similarly, one round of the wSparkle-32 has $2 \cdot xor$ operations from round constants, and $14 \cdot xor + 2 \cdot or$ operations from the linear layer, so the total computational cost is $224 \cdot xor + 28 \cdot or$ operations for the wSparkle-32.

3.1.4 Spook

The Spook is an LS-design based lightweight algorithm. It has the Clyde permutation with a 128-bit block size, and the Shadow permutation with 384/512-bit block sizes. The permutations are based on bitslice implemented LS-design and consist of nonlinear Sbox, linear Lbox, and round constant addition layers in one round. Shadow's difference from Clyde is linear Dbox and another round constant layers. Bitslice implemented Lbox takes two 32-bit state words to increase the branch number. The inner operations are specified as in Table 3.1. Since Lbox is applied to 128-bit subblocks, the additional linear component Dbox is used to diffuse 128-bit blocks to each other in the Shadow.

Table 3.1: Lbox

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

3.1.4.1 White-box conversions

The white-box conversions are applied to 128/256/384 bits block sizes with 32-bit word size. Since the lightweight design does not have a 256-bit block variant, the white-box conversion with some modifications has been performed for this block size. The lightweight permutation is converted to the white-box setting by adding the lookup table as a nonlinear layer at the beginning of the permutation. The nonlinear layer of the lightweight permutation is discarded. In the white-box implementations, two Lboxes are used in a round successively, and we change the input order of Lboxes according to the round number to increase diffusion property. 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. At the same time, the 256-bit wShadow is performed by changing the linear Dbox with a 256-bit variation since the Dbox in lightweight design cannot be directly adapted.

Implementation details of the Dboxes are given in Algorithm 13 and Algorithm 14. The round constants generated from a 4-bit LFSR, as well as linear components, are taken from the lightweight design to the white-box conversions.

Algorithm 13 Dbox for 256-bit.

Input: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 7\}$
Output: $x_i \in (\mathbb{F}_2^{32}),\ i \in \{0, \cdots, 7\}$
for $i = 0$ to 3 **do**
    $x_i \leftarrow x_i \oplus (x_{i+4} \lll 15)$
    $x_{i+4} \leftarrow x_i \oplus (x_i \lll 19)$
end for

Algorithm 14 Dbox for 384-bit.

Input: $x_i \in (\mathbb{F}_2^{32}), i \in \{0, \cdots, 11\}$
Output: $x_i \in (\mathbb{F}_2^{32}), i \in \{0, \cdots, 11\}$
for $i = 0$ to 3 **do**
    $u \leftarrow x_{4 \cdot i}$
    $v \leftarrow x_{4 \cdot i+1}$
    $t \leftarrow x_{4 \cdot i+2}$
    $x_{4 \cdot i} \leftarrow u \oplus v \oplus t$
    $x_{4 \cdot i+1} \leftarrow u \oplus t$
    $x_{4 \cdot i+2} \leftarrow u \oplus v$
end for

The pseudo-code of the white-box conversions are specified in Algorithm 15. Round numbers for 128/256/384 bits block sizes of the white-box implementations are calculated as 12/14/15, respectively. The wClyde has $24 \cdot xor + 24 \cdot or$ from $lbox\_layer$ and $4 \cdot xor$ operations from add_rc in one round. Total computational cost of the wClyde is $336 \cdot xor + 288 \cdot or$ operations. One round of the wShadow256 has $48 \cdot xor + 48 \cdot or$ from $lbox\_layer$, $8 \cdot xor$ and $8 \cdot or$ operations from $dbox\_layer$, and $16 \cdot xor$ operations from add_rc. Therefore, total computational cost of the wShadow256 is $1008 \cdot xor + 784 \cdot or$ operations. Similarly, one round of the wShadow384 has $72 \cdot xor + 72 \cdot or$ from $lbox\_layer$, $16 \cdot xor$ operations from $dbox\_layer$, and $24 \cdot xor$ operations from add_rc. Total computational cost of the wShadow384 is $1680 \cdot xor + 1080 \cdot or$ operations.

3.1.5 Computational Cost Comparisons

The computational cost of the white-box algorithms is stated in Table 3.2. The cost of WARX is discarded since it uses a library for finite field computations. The word size of the opeartions is 16-bit for wSaturnin and wSparkle-16, while 32-bit operations are applied for the remaining algorithms. wSaturnin has more operations when compared

Algorithm 15 wClyde/wShadow256/wShadow384.

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 $4 \cdot n - 1$ **do**
    $x_j \leftarrow T(x_j)$
  end for
  for $j = 0$ to n-1 **do**
    $(x_{4 \cdot j}, x_{4 \cdot j+1+(j\&1)}) \leftarrow lbox\_layer(x_{4 \cdot j}, x_{4 \cdot j+1+(j\&1)})$
    $(x_{4 \cdot j+2-(j\&1)}, x_{4 \cdot j+3}) \leftarrow lbox\_layer(x_{4 \cdot j+2-(j\&1)}, x_{4 \cdot j+3})$
    $(x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}) \leftarrow add\_rc((x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}), 2 \cdot i, j)$
  end for
  if n>1 **then**
    $(x_0 \cdots x_{4 \cdot n}) \leftarrow dbox\_layer(x_0 \cdots x_{4 \cdot n})$
    for $j = 0$ to n-1 **do**
      $(x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}) \leftarrow add\_rc((x_{4 \cdot j}, \cdots, x_{4 \cdot j+3}), 2 \cdot i + 1, j)$
    end for
  end if
end for

to wSparkle-16. For 32-bit word size algorithms, the most costly algorithm is the white-box wShadow384, and the cheapest one is the white-box wSparkle-32.

Table 3.2: Computational cost

Algorithm	Cost
wSaturnin	$320 \cdot xor + 192 \cdot or + 192 \cdot and$
wSparkle-16	$224 \cdot xor + 16 \cdot or$
SPNbox-32	$512 \cdot xor + 192 \cdot and + 192 \cdot mul$
wClyde	$336 \cdot xor + 288 \cdot or$
wSparkle-32	$224 \cdot xor + 28 \cdot or$
wShadow256	$1008 \cdot xor + 784 \cdot or$
wShadow384	$1680 \cdot xor + 1080 \cdot or$

3.2 Security

The white-box security of an algorithm is evaluated by its resistance to key extraction from the lookup table and by its inability to use the table as a large key outside of the white-box environment called code lifting [9].

3.2.1 Key Extraction Security

The key extraction security of the white-box conversions relies on the security of the table creation methods of WARX [42] and SPNbox [10]. WARX's table creation method is similar to ARX-box of the SPARX [23]. The optimal differential characteristic probability and the details of the optimal linear trail are given in the article [42]. Security of the table construction method of SPNbox-32 is discussed in Section 2.2. According to the number of active Sboxes, the method provides 128-bit security with 10 rounds. On the other hand, at least 18 rounds are required to provide 256-bit security.

3.2.2 Code Lifting Security

Resistance to code-lifting attacks is a crucial security measurement for space-hard ciphers since lookup tables are used as a large key in the design. Therefore, incompressible tables are needed to limit leakage of the code (table) and prevent code lifting attacks [22]. Code lifting security is defined by weak (M, Z)-space hardness criteria [9].

When computing the probability of success of the encryption/decryption in a white-box context, we also include correctly guessing the corresponding entry of the table in Equation 3.1 if the entry is not located in the leaked part of the lookup table, as specified in [42]. Let the size of the entire table in memory be T, and the size of the leaked part of the table be M. We can generalize the success probability as

$$p = (\frac{M}{T} + (\frac{1}{T-M}) \cdot (1 - \frac{M}{T}))^{r \cdot t} \tag{3.1}$$

where r represents the round number, and t represents the table lookup number for one round. If the corresponding entry is in the leaked part of the table, the probability of being correct is 1. The probability of encountering such entries is at most $\frac{M}{T}$, depending on the leak size. If the corresponding value is not in the leaked part, the probability of correctly guessing the value is $\frac{1}{T-M}$. The probability of encountering such an entry is $1 - \frac{M}{T}$.

The maximum achievable security for a white-box algorithm is calculated by considering the leaked size of the lookup table [25, 14, 42] and limited to $keysize - log_2(T)$ bits. Since $\frac{M}{T}$ is considered a small rate, the level of security is generalized to the leak of the entire table. However, we think it is more convenient to take the security level as $keysize - log_2(M)$ bits for more precise calculations on round numbers. The round number may be smaller than the desired number assuming the entire table is leaked. If the leak limit is exceeded, the lookup table must be updated to provide the recommended size of security.

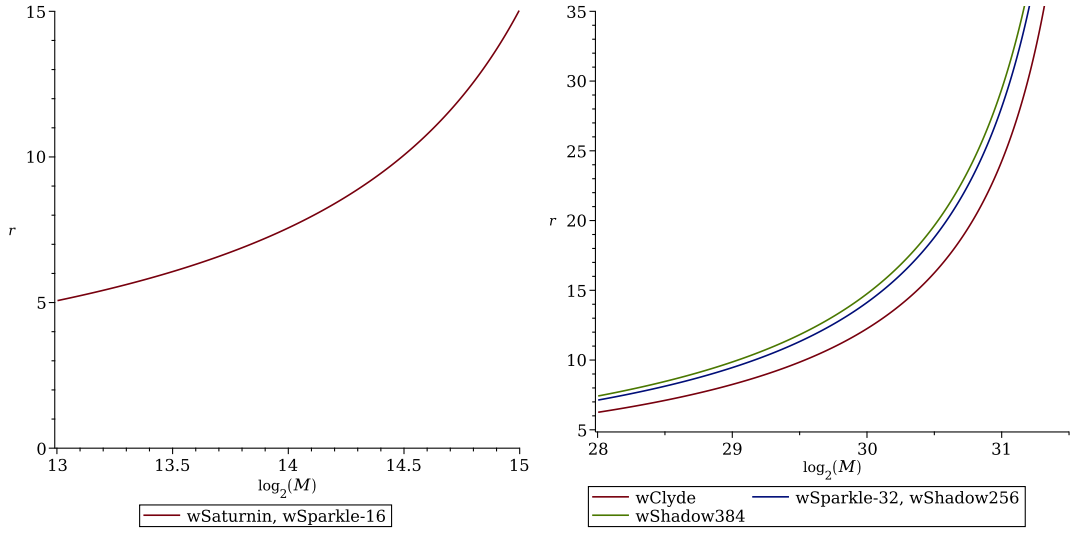
The numbers of rounds of the white-box implementations are calculated according to Equation (3.2) and the results are given in Table 3.3. The table and leakage sizes are taken as T = $wordsize \cdot 2^{wordsize}$-bit and M = $\frac{T}{4}$-bit.

$$M \cdot 2^{-keysize} = (\frac{M}{T} + (\frac{1}{T-M}) \cdot (1 - \frac{M}{T}))^{rt} \tag{3.2}$$

Table 3.3: Round numbers

Algorithm	$(keysize, wordsize)$	$t = \frac{keysize}{wordsize}$	$r = \frac{\log_2(M) - keysize}{t \cdot \log_2(\frac{M+1}{T})}$
wSaturnin	(256,16)	16	8
wSparkle-16	(256,16)	16	8
wClyde	(128,32)	4	12
wSparkle-32	(256,32)	8	14
wShadow256	(256,32)	8	14
wShadow384	(384,32)	12	15

Similar to [9], we assumed that space-hardness is fixed to the leakage of $\frac{1}{4}$ of the lookup table. The relation between the leak size of the table and the round numbers is shown in Figure 3.1. As the defined leak size increases, the round number of the white-box algorithm increases to provide the expected security level. For fixed leak



(a) (b)

Figure 3.1: Relation between leakage size and round numbers.

size, if an algorithm offers higher security than others with the same number of lookup table entries in one round, the algorithm has more rounds than the others. Also, as the lookup table number in a round increases, the number of round decreases.

3.2.3 Diffusion Criteria

The strict avalanche criteria [48] is used to measure the diffusion property of the white-box conversions. The SAC test aims to measure the effect on the output bits when one bit of the input is changed. The experiments were performed on 2^{20} random samples as described in [3]. For each sample, i-th bit of the input was complemented, and its effect on each output bit was examined, respectively. If the j output bit was changed then the entry of $M_{i,j}$ was increased by one. The minimum and maximum $M_{i,j}$ values are stated in Table 3.4 The mean μ value of binomial distribution of M is computed as

$$\mu = \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} M_{i,j} \tag{3.3}$$

where *n* represents the block length of the cipher. The variance *var* of the distribution matrix is computed as

$$var = \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (M_{i,j} - \mu)^2 \tag{3.4}$$

According to the SAC test results in Table 3.4, the WARX provides diffusion property after two rounds, but the wSaturnin provides it after 3 rounds. The wSparkle-16/wSparkle-32 needs four rounds to diffuse every bit of the output. The wClyde and the wShadow256/wShadow384 supply diffusion property after the two round, while SPNbox-32 provides full diffusion after first round.

Table 3.4: SAC test results with 2^{20} random samples.

Algorithm	(*n*, round)	Min. Value	Max. Value	μ	*var*
WARX	(128,2)	522177	526165	524285	262294
wSaturnin	(256,3)	522108	526384	524290	262414
wSparkle-16	(256,4)	521898	526455	524259	272673
SPNbox-32	(128,1)	522413	526444	524283	263596
wClyde	(128,2)	521725	526323	524290	265564
wSparkle-32	(256,4)	521984	526579	524289	260823
wShadow256	(256,2)	522136	526472	524287	261908
wShadow384	(384,2)	522129	526902	524288	260957

3.3 Performance Results

The run-time performance of the white-box and black-box implementations were compared with the algorithms WARX and SPNbox-32. Since the lookup tables differ according to word sizes, the wSaturnin and the wSparkle-16 were compared with WARX and other algorithms with SPNbox-32. WARX code using Givaro library [41] for finite field computations was taken from GitHub [32]. The SPNbox-32 was implemented without using a library by us. While using the linear layers of lightweight designs in the white-box setting, we referenced the NIST GitHub repository [47]. The white-box conversion algorithms [1], run on randomly generated 3072 bytes messages with 100000 cycles and -O3 optimization on a laptop equipped with x86-64 architecture and a 2.80 GHz Intel Core i7-1165G7 CPU. The operating system is Ubuntu 20.04.5 LTS with Linux Kernel 5.15.0, and the compiler is gcc-9.4.0. The performance results for white-box implementations are in Table 3.5 and Table 3.6. Simi-

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

larly, the performance results for black-box implementations are given in Table 3.7 and Table 3.8.

According to the performance results in Table 3.5, the wSaturnin is almost eleven times faster than the WARX, with a MAS level of 242-bit. At the same security level, the wSparkle-16 is 48 times faster than WARX. The code size of wSaturnin and wSparkle-16 is almost half of the WARX's code size, while WARX's memory usage is twelve times higher than the white-box conversions.

Table 3.5: Performance results of the 16-bit word size white-box implementations.

Algorithm	Key Size (bit)	Round	MAS (bit)	Cycle (per byte)	Memory Usage (KiB)	Code Size (KiB)
WARX	128	7	114	288	852.5	223
wSaturnin	256	8	238	26	72	133.7
wSparkle-16	256	8	238	6	72	131.5

Since the round number of SPNbox-32 was calculated as 16 instead of 10 according to the code lifting security criteria, it was taken as 16 in the experiments. Based on performance comparisons for 32-bit word size algorithms in Table 3.6, all white-box conversions are faster than SPNbox-32. The wClyde is 38% faster than SPNbox-32 with a 96-bit MAS level. The wSparkle-32 and the wShadow256 are faster than SPNbox-32 with 42% and 16%, respectively. The wShadow384 is 24% faster than SPNbox-32 with a 354-bit MAS level. Although the memory usage of all algorithms is the same, SPNbox-32 and wSparkle-32 have the smallest code size. The largest code size in white-box implementations belongs to the wShadow384.

Table 3.6: Performance results of the 32-bit word size white-box implementations.

Algorithm	Key Size (bit)	Round	MAS (bit)	Cycle (per byte)	Memory Usage (GiB)	Code Size (KiB)
SPNbox-32	128	16	128	140	16	1.9
wClyde	128	12	93	87	16	2.8
wSparkle-32	256	14	221	82	16	1.9
wShadow256	256	14	221	118	16	3.6
wShadow384	384	15	349	107	16	3.9

While the white-box algorithm is used in the client side, the black-box algorithms is utilized for server side. In the black-box impementations, instead of using a lookup table, the table generation method is integrated into the nonlinear layer. The table

creation method of WARX is implemented in the nonlinear layer of the black-box wSaturnin and the black-box wSparkle-16, while SPNbox's method is implemented in the remaining black-box algorithms.

According to the results in Table 3.7, the black-box wSaturnin is two times faster than WARX, while the black-box wSparkle-16 is 2.3 times faster than WARX. Similar to the white-box implementations, both wSaturnin's and wSparkle-16's code sizes are smaller than WARX, while WARX's memory usage is almost 12 times higher than the 16-bit implementations.

Table 3.7: Performance results of the 16-bit word size black-box implementations.

Algorithm	Key Size (bit)	Round	Cycle (per byte)	Memory Usage (KiB)	Code Size (KiB)
WARX	128	7	460	852.5	94
wSaturnin	256	8	220	72	5.4
wSparkle-16	256	8	199	72	5.2

According to performance results in Table 3.8, the black-box wClyde and the black-box wSparkle-32 are 11% and 14% faster than the black-box SPNbox-32. Unfortunately, the black-box implementations of wShadow256 and wShadow384 are 4% and 12% slower than SPNbox-32. Memory usage of the black-box implementations is the same. When the code sizes of the algorithms are compared, the smallest is the wSparkle-32, and the largest is the wShadow256.

Table 3.8: Performance results of the 32-bit word size black-box implementations.

Algorithm	Key Size (bit)	Round	Cycle (per byte)	Memory Usage (KiB)	Code Size (KiB)
SPNbox-32	128	16	2007	72	2.9
wClyde	128	12	1784	72	3.3
wSparkle-32	256	14	1722	72	2.7
wShadow256	256	14	2090	72	3.8
wShadow384	384	15	2241	72	3.5

The different leak sizes for 114-bit and 98-bit expected security levels are defined in Table 3.9. According to the table, there is an inverse relationship between the block size of the algorithms and the table leak sizes at the same security level. Because wShadow384's block size is larger than the others, providing the same security until a larger table leaks. The table leak size of SPNbox-32 is calculated as $T/2^{1.53}$ for the

98-bit security level since the table leak is not included when calculating the round number in [10].

Table 3.9: Table leakage size for 2^{-114} and 2^{-98} success probability.

Algorithm	Security Size (bit)	Table Size (T)	Leakage Size (bit)
WARX	114	128 KB	$T/2^{2}$
wSaturnin	114	128 KB	$T/2^{0.89}$
wSparkle-16	114	128 KB	$T/2^{0.89}$
SPNbox-32	98	16 GB	$T/2^{1.53}$
wClyde	98	16 GB	$T/2^{2.04}$
wSparkle-32	98	16 GB	$T/2^{0.88}$
wShadow256	98	16 GB	$T/2^{0.88}$
wShadow384	98	16 GB	$T/2^{0.54}$

CHAPTER 4

A NEW LS-DESIGN BASED WHITE-BOX BLOCK CIPHER

The reliability of the table creation method is important to prevent key extraction attacks in space-hard ciphers as the key is embedded in the table. LS-design based algorithms [27, 31] aim to prevent differential side-channel analysis with bitslice implementations. In this context, we proposed a white-box algorithm and a table generation method based on LS-design to take advantage of security considerations along with efficiency. The table construction method is detailed in Section 4.1. The details of the white-box algorithm is given in Section 4.2. Security of the designed method is discussed in Section 4.4 and performance comparisons are stated in Section 4.5.

4.1 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 4.1. 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.

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

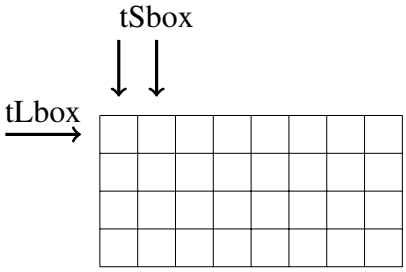


Figure 4.1: State of the Input

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 16.

Algorithm 16 Table Construction

1: INPUT: $x \in (\mathbb{F}_2^{32})$
2: OUTPUT: x
3: $x \leftarrow bitslice(x)$
4: **for** $i = 0$ to tr **do**
5: $\quad x \leftarrow addroundkey(x, key[i])$
6: $\quad x \leftarrow tsbox(x)$
7: $\quad x \leftarrow tlbox(x)$
8: $\quad x \leftarrow addroundconstant(x)$
9: **end for**
10: $x \leftarrow unbitslice(x)$

4.1.1 tSbox

The 8-bit tSbox is taken from Scream-v3 algorithm [28], the second round candidate of CAESAR competition [18]. 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 4.1. In the implementation, the nonlinear layer *tsbox* is pre-computed and used as a substitution box.

Table 4.1: 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
$x_0 = \neg((s_1 \,\&\, s_2) \oplus s_0)$
$x_1 = s_1 \oplus s_3$
$x_2 = s_2 \oplus x_0$
$s_4 = s_4 \oplus ((s_3 \oplus x_2) \,\&\, (s_2 \oplus x_1))$
$s_5 = s_5 \oplus x_2$
$s_6 = s_6 \oplus (s_3 \,\&\, x_0)$
$s_7 = s_7 \oplus (x_1 \,\&\, x_2)$

4.1.2 tLbox

The linear layer tLbox is taken from Mysterion algorithm [31]. 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 $\mathbb{F}_{2^q}$. 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.

4.1.3 Round Keys

The round keys are generated using extendable output function SHAKE [24] 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.

4.1.4 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.

4.2 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 17. The numbers of rounds for the white-box implementations' 128-bit and 256-bit block sizes are calculated as 12 and 14, respectively.

4.2.1 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 4.1.

4.2.2 Linear Layer

The linear layer Lbox is taken from the lightweight design Spook [4]. Bitslice implemented linear layer *lbox* takes two 32-bit inputs and is applied twice in one round.

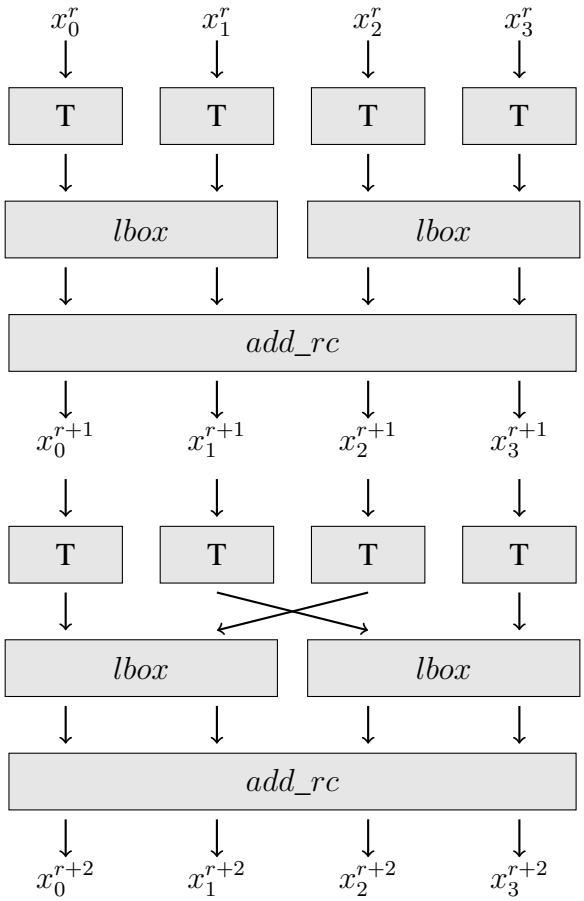


Figure 4.2: Two rounds of the white-box algorithm of 128-bit block size

The internal operations are specified as in Table 4.2. As detailed in Section 3.1.4, the order of the *lbox* entries is chosen according to the round number to increase the diffusion effect.

Table 4.2: 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 4.3. Each word of one subblock is rotated to the left and xored to the corresponding state of the other subblock.

Algorithm 17 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 4.3: Dbox

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

4.2.3 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 $f(x) = x^8 + x^5 + x^3 + x + 1$.

4.2.4 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.

4.3 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 18.

4.4 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.

4.4.1 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.

Algorithm 18 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

4.4.1.1 Differential and Linear Cryptanalysis

The LS-design is based on WTS approach [20]. The linear and differential probability are formalized in [27] as,

$$Pr_{\text{diff}}(2r) \leq Pr_{\text{diff}}^{\max}(S)^{r \cdot B(L)} \tag{4.1}$$

and

$$Pr_{\text{lin}}(2r) \leq Pr_{\text{lin}}^{\max}(S)^{r \cdot B(L)} \tag{4.2}$$

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 4.1 and 4.2, 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 [44], 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.

4.4.1.2 Slide Attacks

Slide attacks [7, 8] 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.

4.4.1.3 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 [12, 11]. Therefore, at least three rounds are required to reach the algebraic degree 31 against the attacks.

4.4.1.4 Structural Attacks

The invariant subspace attacks [38] and the nonlinear invariant attacks [46] 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.

4.4.2 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.

4.4.2.1 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 [9]. 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.

4.4.2.2 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 [22]. 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 [25, 14, 42]. 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 (4.3)

$$r = \frac{\log_2(L) - keysize}{n \cdot \log_2(\frac{L+1}{T})}, \tag{4.3}$$

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 4.3, 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.

4.4.3 The Black-Box Security

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

4.4.3.1 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 4.1 and 4.2, the desired security levels against differential and linear attacks are provided.

4.4.3.2 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.

4.5 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 4.4 for white-box implementation and in Table 4.5 for black-box implementation.

According to the performance results in Table 4.4, wClyde is 28% faster than SPNbox-32, and wShadow is 15% faster than SPNbox-32.

Table 4.4: 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

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 to performance results in Table 4.5, 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 4.5: 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

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

CHAPTER 5

CONCLUSION

White-box cryptography aims to provide software security for devices in untrusted environments where key security cannot be achieved by TPM or TEE similar hardware tools. 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 using a small block cipher. This large table is used as the nonlinear layer in the white-box algorithm design. One of the issues of these ciphers is to improve the run-time performance of white-box and black-box implementations.

This thesis examined lightweight designs for efficiency and suitability to white-box settings. With this approach, linear layers of the appropriate algorithms from the NIST Lightweight Standardization candidates were adapted to the white-box settings to speed up the run-time of the white-box/black-box implementations according to space hard ciphers WARX and SPNbox-32. The nonlinear layers, and if there was a round key addition part, of the lightweight designs were discarded in the white-box setting. White-box conversions of the selected algorithms have been tested with SAC security criteria. According to test results, linear components of Saturnin, Sparkle, and Shadow provided the diffusion property in the white-box conversions.

The round numbers of the white-box algorithms were calculated according to (M, Z)-space hardness criteria. In order to make more accurate calculations, the security size of the algorithms was taken as $leaksize - log_2(M)$ bits. According to the performance results, all white-box conversions were faster than (M, Z)-space hard al-

gorithms WARX and SPNbox-32 without decreasing the white-box security level. Using the lightweight components in the white-box settings enabled reasonably fast algorithm designs.

In this study, we proposed a new LS-design-based white-box algorithm and table construction method. With this white-box algorithm, we had performance improvement as 28% for white-box implementation and 27% for black-box implementation for 128-bit block size. Moreover, we proposed an LS-design-based white-box algorithm for 256-bit block size using the suggested table creation method.

REFERENCES

[1] P. Barreto and V. Rijmen, The khazad legacy-level block cipher, Primitive submitted to NESSIE, 97(106), 2000.

[2] P. S. Barreto, The anubis block cipher, NESSIE, 2000.

[3] C. Beierle, A. Biryukov, L. Cardoso dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, Q. Wang, A. Moradi, and R. Shahmirzadi, Schwaemm and esch: Lightweight authenticated encryption and hashing using the sparkle permutation family, 2021, https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/sparkle-spec-final.pdf.

[4] D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, I. Levi, C. Momin, O. Pereira, T. Peters, F.-X. Standaert, and F. 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.

[5] 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*, pp. 227–240, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, ISBN 978-3-540-30564-4.

[6] A. Biryukov, C. Bouillaguet, and D. Khovratovich, Cryptographic schemes based on the asasa structure: Black-box, white-box, and public-key (extended abstract), in P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pp. 63–84, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, ISBN 978-3-662-45611-8.

[7] A. Biryukov and D. Wagner, Slide attacks, in *Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings 6*, pp. 245–259, Springer, 1999.

[8] A. Biryukov and D. Wagner, Advanced slide attacks, in *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pp. 589–606, Springer, 2000.

[9] A. Bogdanov and T. Isobe, White-box cryptography revisited: Space-hard ciphers, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, p. 1058–1069, Association for Computing Machinery, New York, NY, USA, 2015, ISBN 9781450338325.

[10] A. Bogdanov, T. Isobe, and E. Tischhauser, Towards practical whitebox cryptography: Optimizing efficiency and space hardness, in J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pp. 126–158, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, ISBN 978-3-662-53887-6.

[11] C. Boura and A. 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(1), pp. 691–702, 2012.

[12] C. Boura, A. Canteaut, and C. De Cannière, Higher-order differential properties of keccak and luffa, in A. Joux, editor, *Fast Software Encryption*, pp. 252–269, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, ISBN 978-3-642-21702-9.

[13] A. Canteaut, S. Duval, G. Leurent, M. Naya-Plasencia, L. Perrin, T. Pornin, and A. Schrottenloher, Saturnin: a suite of lightweight symmetric algorithms for post-quantum security, 2019, `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/saturnin-spec-round2.pdf`.

[14] J. Cho, K. Y. Choi, I. Dinur, O. Dunkelman, N. Keller, D. Moon, and A. Veidberg, Wem: A new family of white-box block ciphers based on the even-mansour construction, in H. Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, pp. 293–308, Springer International Publishing, Cham, 2017, ISBN 978-3-319-52153-4.

[15] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot, White-box cryptography and an aes implementation, in K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography*, pp. 250–270, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-36492-4.

[16] 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*, pp. 1–15, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-44993-5.

[17] C. Cid, S. Murphy, and M. J. B. Robshaw, Small scale variants of the aes, in H. Gilbert and H. Handschuh, editors, *Fast Software Encryption*, pp. 145–162, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, ISBN 978-3-540-31669-5.

[18] C. Competitions, Caesar: Competition for authenticated encryption: Security, applicability, and robustness, https://competitions.cr.yp.to, accessed: 2022, November 20.

[19] CSRC, Lightweight cryptography, https://csrc.nist.gov/Projects/lightweight-cryptography, 2021, accessed: 2022, February 15.

[20] J. Daemen and V. Rijmen, The wide trail design strategy, in B. Honary, editor, *Cryptography and Coding*, pp. 222–238, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, ISBN 978-3-540-45325-3.

[21] Y. De Mulder, P. Roelse, and B. Preneel, Cryptanalysis of the xiao–lai white-box aes implementation, in *International conference on selected areas in cryptography*, pp. 34–49, Springer, 2012.

[22] 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*, pp. 247–264, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, ISBN 978-3-662-43414-7.

[23] D. Dinu, L. Perrin, A. Udovenko, V. Velichkov, J. Großschädl, and A. Biryukov, Design strategies for arx with provable bounds: Sparx and lax, in J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pp. 484–513, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, ISBN 978-3-662-53887-6.

[24] M. J. Dworkin et al., Sha-3 standard: Permutation-based hash and extendable-output functions, 2015.

[25] P.-A. Fouque, P. Karpman, P. Kirchner, and B. Minaud, Efficient and provable white-box primitives, in J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pp. 159–188, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, ISBN 978-3-662-53887-6.

[26] H. Gilbert, J. Plût, and J. Treger, Key-recovery attack on the asasa cryptosystem with expanding s-boxes, in R. Gennaro and M. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pp. 475–490, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, ISBN 978-3-662-47989-6.

[27] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varıcı, Ls-designs: Bitslice encryption for efficient masked software implementations, in C. Cid and C. Rechberger, editors, *Fast Software Encryption*, pp. 18–37, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, ISBN 978-3-662-46706-0.

[28] V. Grosso, G. Leurent, F.-X. Standaert, K. Varici, F. Durvaux, L. Gaspar, and S. Kerckhof, Scream & iscream side-channel resistant authenticated encryption with masking, 2015.

[29] P. Hawkes and L. O'Connor, Xor and non-xor differential probabilities, in J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pp. 272–285, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, ISBN 978-3-540-48910-8.

[30] M. Jacob, D. Boneh, and E. Felten, Attacking an obfuscated cipher by injecting faults, in *ACM Workshop on Digital Rights Management*, pp. 16–31, Springer, 2002.

[31] A. Journault, F.-X. Standaert, and K. Varıcı, Improving the security and efficiency of block ciphers based on ls-designs, in *Designs, Codes and Cryptography*, pp. 495–509, 2017.

[32] JunLiu9102, Warx-project, `https://github.com/JunLiu9102/WARX-Project`, accessed: 2021, September 20.

[33] M. Karroumi, Protecting white-box aes with dual ciphers, in K.-H. Rhee and D. Nyang, editors, *Information Security and Cryptology - ICISC 2010*, pp. 278–291, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, ISBN 978-3-642-24209-0.

[34] Y. Koike and T. Isobe, Yoroi: Updatable whitebox cryptography, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(4), p. 587–617, Aug. 2021.

[35] Y. Koike, K. Sakamoto, T. Hayashi, and T. Isobe, Galaxy: A family of stream-cipher-based space-hard ciphers, in J. K. Liu and H. Cui, editors, *Information Security and Privacy*, pp. 142–159, Springer International Publishing, Cham, 2020, ISBN 978-3-030-55304-3.

[36] J. Kwon, B. Lee, J. Lee, and D. Moon, Fpl: White-box secure block cipher using parallel table look-ups, in S. Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pp. 106–128, Springer International Publishing, Cham, 2020, ISBN 978-3-030-40186-3.

[37] G. Leander, Small scale variants of the block cipher present, 2010, g.leander@mat.dtu.dk 14684 received 16 Mar 2010.

[38] G. Leander, B. Minaud, and S. Rønjom, A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro, in *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*, pp. 254–283, Springer, 2015.

[39] T. Lepoint, M. Rivain, Y. De 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*, pp. 265–285, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, ISBN 978-3-662-43414-7.

[40] S. Li, S. Sun, C. Li, Z. Wei, and L. Hu, Constructing low-latency involutory mds matrices with lightweight circuits, IACR Transactions on Symmetric Cryptology, 2019(1), p. 84–117, Mar. 2019.

[41] linbox team, givaro, https://github.com/linbox-team/givaro, accessed: 2021, September 20.

[42] J. Liu, V. Rijmen, Y. Hu, J. Chen, and B. Wang, Warx: efficient white-box block cipher based on arx primitives and random mds matrix, Science China Information Sciences, pp. 1869–1919, 2021, https://doi.org/10.1007/s11432-020-3105-1.

[43] B. Minaud, P. Derbez, P.-A. Fouque, and P. Karpman, Key-recovery attacks on asasa, in *Journal of Cryptology*, volume 31, pp. 845–884, Springer, 2018.

[44] N. Mouha, Q. Wang, D. Gu, and B. Preneel, Differential and linear cryptanalysis using mixed-integer linear programming, in C.-K. Wu, M. Yung, and D. Lin, editors, *Information Security and Cryptology*, pp. 57–76, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, ISBN 978-3-642-34704-7.

[45] K. Stoffelen and J. Daemen, Column parity mixers, IACR Transactions on Symmetric Cryptology, 2018(1), p. 126–159, Mar. 2018.

[46] Y. Todo, G. Leander, and Y. Sasaki, Nonlinear invariant attack: Practical attack on full scream, i scream, and midori 64, in *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*, pp. 3–33, Springer, 2016.

[47] usnistgov, Lightweight-cryptography-benchmarking, https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking, 2021, accessed: 2022, February 15.

[48] A. F. Webster and S. E. Tavares, On the design of s-boxes, in H. C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pp. 523–534, Springer Berlin Heidelberg, Berlin, Heidelberg, 1986, ISBN 978-3-540-39799-1.

[49] Y. Xiao and X. Lai, A secure implementation of white-box aes, in *2009 2nd International Conference on Computer Science and its Applications*, pp. 1–6, IEEE, 2009.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Güner, Hatice Kübra

EDUCATION

Degree	**Institution**	**Year of Graduation**
Ph.D.	METU, Cryptography	2023
M.S.	METU, Cryptography	2015
B.S.	Hacettepe University, Mathematics	2012
High School	Fatih Sultan Mehmet High School	2007

PUBLICATIONS

- H.K. Güner, C. Mangır, O. Yayla, Performance Improvement of White-Box Algorithms Using Lightweight Components, Algebraic and Combinatorial Methods for Coding and Cryptography, ALCOCRYPT 2023.
- H.K. Güner, C. Mangır, O. Yayla, Improving Performance in Space-Hard Algorithms, In: Dolev, S., Gudes, E., Paillier, P. (eds), Cyber Security, Cryptology, and Machine Learning, CSCML 2023, Lecture Notes in Computer Science, vol 13914. Springer, Cham., https://doi.org/10.1007/978-3-031-34671-2_28.
- H.K. Güner, C. Mangır, and O. Yayla, White-Box Block Cipher Implementation Based on LS-Design, Cryptology ePrint Archive, Paper 2023/1096, 2023, https://eprint.iacr.org/2023/1096.