Journal of Universal Computer Science, vol. 25, no. 8 (2019), 868-886
submitted: 26/12/18, accepted: 17/7/19, appeared: 28/8/19 © J.UCS

Towards Secure and Efficient “white-box” Encryption

Gurgen Khachatrian
(American University of Armenia, Yerevan, Armenia
gurgenkh@aua.am)

Sergey Abrahamyan
(Institute for Informatics and Automation Problems, Yerevan, Armenia
serj.abrahamyan@gmail.com)

Abstract: In many applications implemented in mistrusted environments all operations with the secret key during an encryption operation are “obfuscated” in a way that while an attacker has access to all routines of operations, it will nevertheless be hard for him to determine the value of the secret key used during these operations. This kind of execution of encryption operation is called a white-Box implementation. The design of a secure cryptosystem robust in the context of white-box attack is a difficult task which has been addressed by many researchers in the last two decade. The existing implementations of white-box algorithms have been mainly based on AES block cipher, however, all the known systems have been broken. In this paper a design and security analysis of a novel white-box encryption based on SAFER+ block cipher algorithm is presented which is shown to be secure against major attacks successfully applied to AES-based cryptosystems, such as the so-called BGE attack and others.

Key Words: White-box cryptography, Encryption, Security analysis
Category: E.3

1 Introduction

In the so called “black-box” encryption model cryptographic keys are protected by different methods such as passwords or in tamper-resistant modules. In this case the attacker of the system can see only inputs and outputs of the encryption engine and has no access to intermediate values inside the black box. In the white-box context an encryption routine is represented via lookup tables based on encryption secret keys which are accessible to the public in each of rounds. The main purpose of these look-up tables is to hide cryptographic keys when performing correct encryption operations. As such white-box encryption allows anyone who has access to the white-box look-up tables to implement an encryption operation in the way that only the owner of a secret key can decrypt a result and get a valid plaintext. The security of the white-box encryption is the complexity of guessing a secret key or making a decryption operation without knowing a secret key. In this way white-box look-up tables can be considered as a Public key in the sense that anyone can encrypt a message, but only the holder of a secret key can decrypt it. Based on this idea the white-box encryption (or analogously, decryption) can find numerous applications where it would be much more efficient to use white-box encryption and symmetric decryption instead of computationally expensive public key operations. The major problem with white-box cryptography is its security. The academic study of white-box cryptography was initiated in the seminal work by Chow et al. [Chow et al., 2003, Chow et al., 2002].Subsequently there were numerous attempts to design white-box encryption routine based on Advanced Encryption Standard(AES), all of which were later broken [Bringer et al., 2006, Chow et al., 2002, Yaying et al., 2009, Lepoint et al., 2013, Lepoint et al., 2014, Billet et al., 2003]. Attempts to securely implement AES white-box encryption were further continued in [Karrroumi, 2011] (broken in [Mulder et al., 2013b]), [Yaying et al., 2009](broken in [Mulder et al., 2013a]). However, all of these implementations share the same ideas and their cryptanalysis are mainly based on the BGE approach [Billet et al., 2003] to be described later in this paper. For example, implementation in [Karrroumi, 2011] based on AES dual ciphers was claimed to be secure against BGE attack in the original paper, however it is shown that the BGE attack can be applied with minor modifications to this implementation. The same applies to algorithms described in [Yaying et al., 2009]. In the present paper we propose a white-box encryption based on SAFER+ algorithm [Massey et al., 1998] and show that it is secure against different types of attacks including those that found out to be successful against AES white-box implementations. It should be mentioned that a similar attempt to use SAFER+ algorithm was proposed by [Khachatrian at al., 2016], however later this approach showed some security flaws.

The paper is organized as follows: In the section 2 black and white-box encryption rationale of SAFER+ is represented. Section 3 is devoted to the detailed security analysis of SAFER+ white-box encryption. Section 4 includes results of the computational speed of SAFER+ white-box encryption and memory requirements. The paper ends with the conclusion.

2 White Box Design Based on SAFER+

The underlying symmetric encryption algorithm for the white-box encryption is based on SAFER+ algorithm with a 128-bit key running 6 identical rounds plus an extra key addition at the end of 6-th round. This algorithm in fact is a "Black-box" SAFER+ since only input and output values can be accessible to an attacker while all intermediate values, which involves secret keys, remain unknown. The first operation of the round r is the "addition" of the round subkey $\mathbf{k}_{2r-1}$ to the 16 byte round input. The result of the "addition" is then processed by a non-linear layer. The second round subkey $\mathbf{k}_{2r}$ is then added to the output of the non-linear layer. The 16 byte result of this addition is then multiplied by the matrix M. One should mention that all operations are on bytes implemented modulo 256. The round encryption procedure is graphically shown in Fig. 1.

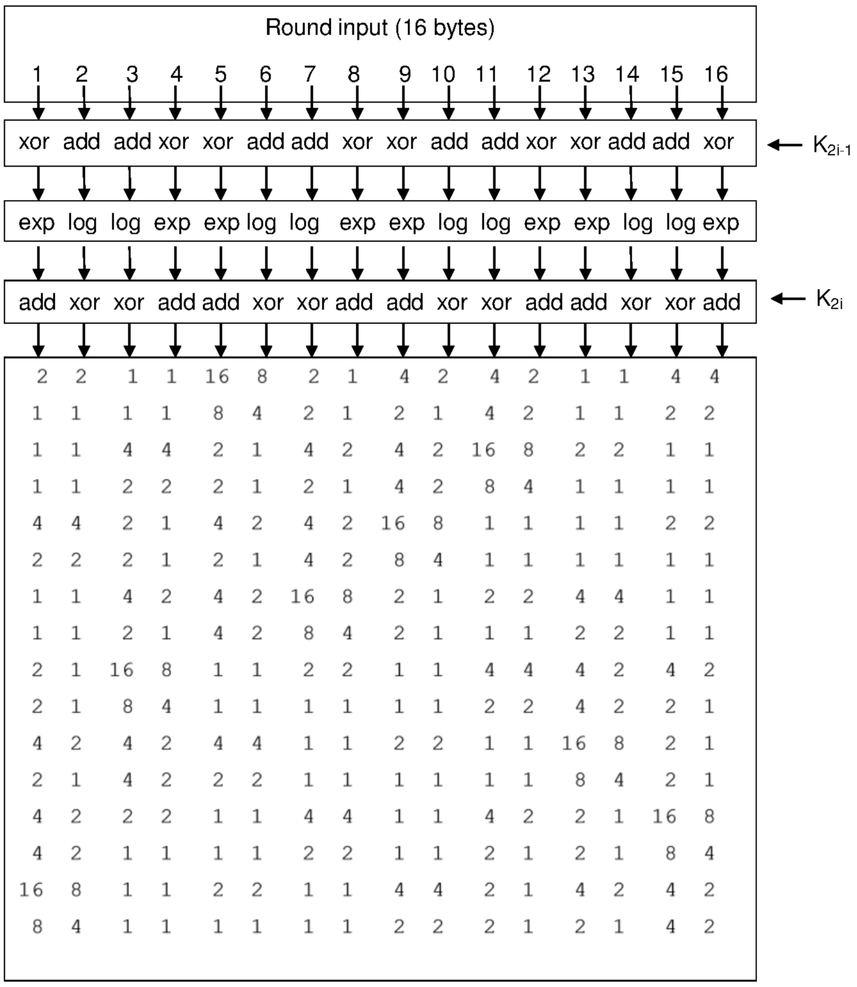


Figure 1: Round Structure of SAFER+.

Below is the meaning of the functions exp and log used during SAFER+ round.

a) exp is based on exponential function $45^x = Y \ mod\ 257$ where X and Y are any numbers between 0 and 255 (actually in this arithmetic 256 is replaced by 0). The second one denoted by log is based on a logarithmic function $\log_{45}(x) = Y \ mod\ 257$.

There are two operations between X and Y bytes. The XOR operation stands for regular XOR of X and Y while the add operation stands for sum between X and Y with $mod\, 256$.

For the SAFER+ white-box(further called SAFER+WB) we‘re merging several steps of encryption rounds into look-up tables, which will be named E-boxes and addition tables, and obfuscate the table‘s inputs and outputs with input/output encodings (bijections) similar to those first introduced by Chow at.al in [Chow et al., 2003, Chow et al., 2002]. As in SAFER+ we split the 16 bytes indices for each round into two sets according to the order in which exponential (exp) and logarithmic (log) functions are applied. As such, set $A = \{1, 4, 5, 8, 9, 12, 13, 16\}$ and $B = \{2, 3, 6, 7, 10, 11, 14, 15\}$. The round confusion operations, namely key additions and non-linear layer evaluation, can be combined into 16 tables which map 1 byte of input to 1 byte of output in following manner:

For $1 \leq r \leq 6$

$$E_i^r(x) := \exp(x \oplus k_{i1}^r) + k_{i2}^r \qquad for\ i \in A,$$

$$E_i^r(x) := \log(x + k_{i1}^r) \oplus k_{i2}^r \qquad for\ i \in B.$$

For security reasons to be explained later we randomly split the outputs of these tables into 2 numbers that sum up to the real value modulo 256. These boxes hereinafter referred to as E-boxes. As such taking this into account, we will need 16 tables which map 1 byte to 2 bytes for round 1 and 2 bytes to 2 bytes for rounds 2 to 6. This is because when E-box output after first round is split into two parts these two parts transfer after multiplication with the matrix M another two parts which in turn become inputs for the next round E-box. And this process is propagated through all six rounds. Random permutations are also applied into outputs of E-boxes. Sixteen random permutations $f_i(i = 1 \ldots 16)$ will be generated at a white-box table generation phase, where $f_i : Z_{256} \rightarrow Z_{256}$. Let us denote the i^{th} mapping for the input value x for the round $r = 1$ by $E_{i1}^1(x)$ and $E_{i2}^1(x)$, and correspondingly i^{th} mapping for input values x_1 and x_2, for rounds $r = 2 \ldots 6$, by $E_{i1}^r(x_1, x_2)$ and $E_{i2}^r(x_1, x_2)$ respectively. Thus our E-boxes will have the following structure:
For $r = 1$

$$f_i(E_{i1}^1(x)) := f_i(\exp(IP^{-1}(x) \oplus k_{i1}^1) + k_{i2}^1 + R_{i1}) \qquad for\ i \in A,$$

$$f_i(E_{i2}^1(x)) := f_i(R_{i2}) \qquad for\ i \in A,$$

$$f_i(E_{i1}^1(x)) := f_i(\log(IP^{-1}(x) + k_{i1}^1) \oplus k_{i2}^1 + R_{i1}) \qquad for\ i \in B,$$

$$f_i(E^1_{i2}(x)) := f_i(R_{i2}) \qquad for\ i \in B.$$

For $2 \leq r \leq 6$

$$f_i(E^r_{i1}(x_1, x_2)) := f_i(\exp((Z^{-1}_{r_i}(x_1) + Z^{-1}_{r_i}(x_2)) \oplus k^r_{i1}) + k^r_{i2} + R_{i1})\ for\ i \in A,$$

$$f_i(E^r_{i2}(x_1, x_2)) := f_i(R_{i2}) for\ i \in A,$$

$$f_i(E^r_{i1}(x_1, x_2)) := f_i(\log((Z^{-1}_{r_i}(x_1) + Z^{-1}_{r_i}(x_2)) + k^r_{i1}) \oplus k^r_{i2} + R_{i1})\ for\ i \in B,$$

$$f_i(E^r_{i2}(x_1, x_2)) := f_i(R_{i2}) \quad for\ i \in B.$$

R_{i1} and R_{i2} are generated randomly such that $R_{i1} + R_{i2} \equiv 0\ (mod\ 256)$. *IP* denotes an initial permutation which is applied at the beginning of WB implementation. More detailed explanation of *IP* is given later. It should be mentioned that $Z^{-1}_{r_i}$ denotes a reverse permutation applied to the output of previous round for the i-th byte. A corresponding list of $Z^{-1}_{r_i}$ permutations is given in (5). For example Z_{r_1} corresponds to Z_9. For the security reason which will be clear from security analysis, R_{i1} is chosen such that

$$(E_i1^r(x) + R_{i1}) \leq 128.$$

A schematic structure of E-boxes is given in Fig. 2 (for the first round) and Fig. 3 (for rounds 2-6).

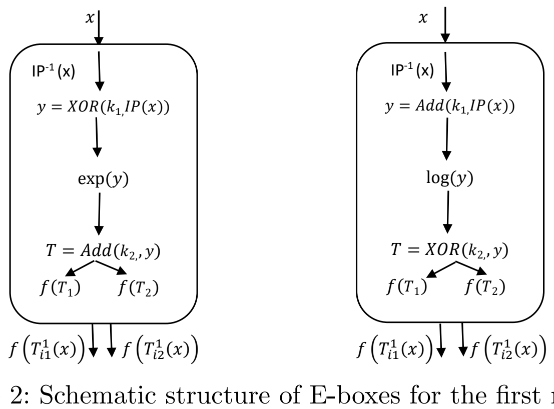


Figure 2: Schematic structure of E-boxes for the first round.

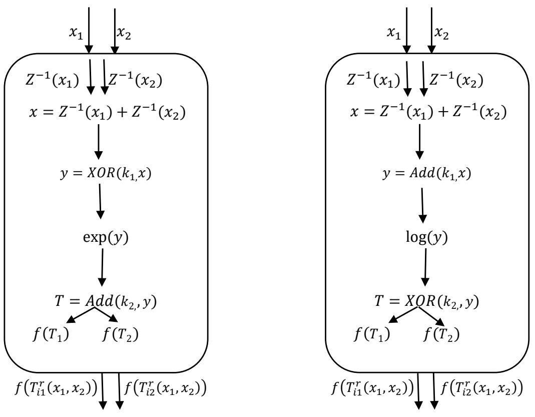


Figure 3: Schematic structure of E-boxes for rounds 2-6.

One should note that in SAFER+ a matrix multiplication operation comes next after second subkey addition. Assuming that the round input is $\bar{x} = (x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \ldots x_{16,1}, x_{16,2})$ for rounds $r = 2\ to\ 6$ (for round $r = 1$ $\bar{x} = (x_1, x_2 \ldots x_{16})$). Then the round output vector $\bar{y} = (y_1, y_2 \ldots y_{16})$ can be presented as follows:

$$\bar{y^r} = (E^r_{1,1}(x_{1,1}, x_{1,2}) + E^r_{1,2}(x_{1,1}, x_{1,2}); E^r_{2,1}(x_{2,1}, x_{2,2}) + E^r_{2,2}(x_{2,1}, x_{2,2});$$

$$\ldots E^r_{16,1}(x_{16,1}, x_{16,2}) + E^r_{16,2}(x_{16,1}, x_{16,2})) \times M = (E^r_{1,1}(x_{1,1}, x_{1,2}); E^r_{2,1}(x_{2,1}, x_{2,2});$$

$$\ldots E^r_{16,1}(x_{16,1}, x_{16,2})) \times M + (E^r_{1,2}(x_{1,1}, x_{1,2}); E^r_{2,2}(x_{2,1}, x_{2,2});$$

$$\ldots E^r_{16,2}(x_{16,1}, x_{16,2})) \times M = \bar{y^r_1} + \bar{y^r_2},$$

where $M = \begin{pmatrix} m_{1,1} & \ldots & m_{1,16} \\ \vdots & \ddots & \vdots \\ m_{16,1} & \ldots & m_{16,16} \end{pmatrix}$ is the SAFER+ diffusion matrix described in Fig. 1 and + stands for vector addition operation modulo 256.

Thus,

$$y^r_{i,1} = E^r_{1,1}(x_{1,1}, x_{1,2}) \cdot m_{1,i} + E^r_{2,1}(x_{2,1}, x_{2,2}) \cdot m_{2,i} + \cdots + E^r_{16,1}(x_{16,1}, x_{16,2}) \cdot m_{16,i} \qquad (1)$$

and

$$y^r_{i,2} = E^r_{1,2}(x_{1,1}, x_{1,2}) \cdot m_{1,i} + E^r_{2,2}(x_{2,1}, x_{2,2}) \cdot m_{2,i} + \cdots + E^r_{16,2}(x_{16,1}, x_{16,2}) \cdot m_{16,i}.$$

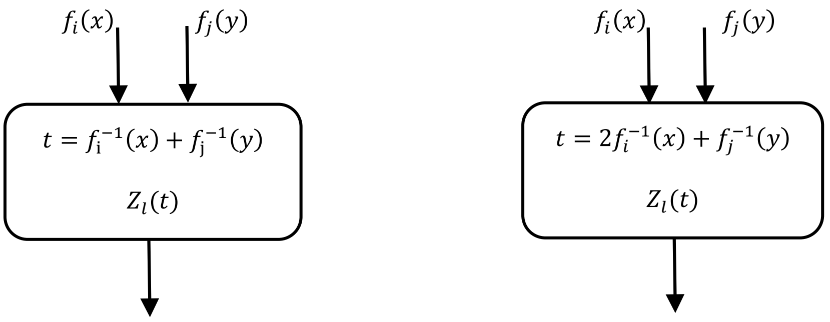
In white-box implementation two split outputs are multiplied with matrix M separately. Thus as a round output we will have two 16-tuple vectors.

In order to obfuscate the linear part of SAFER+ transformation i.e. result of matrix multiplication it will be implemented by using Addition tables. Addition tables are look-up tables which have two x and y bytes as inputs and map them to one output byte z. There will be two types of addition tables, each type addition table uses three different Z_l, f_i^{-1}, f_j^{-1} byte to byte permutations and for the first type there is a functionality:

$$T_{i1} = Z_{l_1}(f_i^{-1}(x) + f_j^{-1}(y)),$$

and for the second type there is a functionality

$$T_{i2} = Z_{l_2}(2f_i^{-1}(x) + f_j^{-1}(y)).$$



$f_i(x)$
$f_j(y)$
$t = f_i^{-1}(x) + f_j^{-1}(y)$
$Z_l(t)$
$f_i(x)$
$f_j(y)$
$t = 2f_i^{-1}(x) + f_j^{-1}(y)$
$Z_l(t)$

Figure 4: Scematic structure of addition tables.

where $i \neq j,\ i, j = 1 \ldots 16$ and $l = 1 \ldots 14$ and f_i^{-1} is an inverse permutation applied to the output of E-box for the i-th byte. Z_l is a permutation applied to the output of lookup table. All lookup tables will be of two types as mentioned earlier and will be based on permutations f_i^{-1}, f_j^{-1}, Z_l or Z_l^{-1}. It is easy to see that in each column of matrix M elements $m_{2i-1,j}$ and $m_{2i,j}$ are either equal to each other or $2m_{2i-1,j} = m_{2i,j}$. In each odd column of M they are $(16, 8)(4, 4)(2, 2)(1, 1)$ pairs and in each even column they are $(8, 4)(2, 2)(1, 1)$ pairs. As such matrix multiplication phase of SAFER+ encryption can be executed by using two types of look-up tables mentioned above. For example the execution of the first byte of the output vector can be represented via look-up tables as follows: For shortness $E_{i,j}^r(x_{i,1}, x_{i,2})$ will be denoted by $E_{i,j}^r$.

Thus, according to (1) we will have

$$y_{1,i}^{r+1} = 2E_{1,1}^r + E_{2,1}^r + E_{3,1}^r + E_{4,1}^r + 4E_{5,1}^r + 2E_{6,1}^r + E_{7,1}^r + E_{8,1}^r + 2E_{9,1}^r + 2E_{10,1}^r +$$

$$+4E_{11,1}^r + 2E_{12,2}^r + 4E_{13,1}^r + 4E_{14,1}^r + 16E_{15,1}^r + 8E_{16,1}^r =$$

$$= (2(2(2(2E_{15,1}^r + E_{16,1}^r) + (E_{13,1}^r + E_{14,1}^r)) + (E_{9,1}^r + E_{10,1}^r)) + (E_{7,1}^r + E_{8,1}^r))+$$

$$+(2(2E_{5,1}^r + E_{6,1}^r) + (E_{3,1}^r + E_{4,1}^r)) + (E_{1,1}^r + E_{6,1}^r). \qquad (2)$$

The formulae (2) can be implemented via 15 two types lookup tables in a cascade manner to obfuscate the correct inputs for the next round. This schematically is shown in Fig. 5. Lookup tables will be numbered like T_i and they will be specified by input and output permutations. For example $T_1 := Z_1(f_1^{-1}; f_2^{-1})$ and $T_{13} := Z_6^{-1}(2f_9^{-1}; f_{10}^{-1})$ meaning that for the table T_1 input permutations are f_1^{-1} and f_2^{-1} and an output permutation is Z_1 while for the table T_{13} input permutations are f_9^{-1} and f_{10}^{-1}, an input corresponding to f_9^{-1} is multiplied by two and an output permutation is Z_6^{-1}. A complete list of lookup tables will be provided later in List 1. In Fig. 5 tables numbers are indicated in the center of boxes. For lookup tables also showing corresponding input and output permutations used for a given table. The basic principle of using a cascade of lookup tables which will be clear from security analysis is that tables are chosen such that input permutations for any lookup table are chosen to be different.

One should note that Fig. 5 corresponds to the transformation for the first byte of the next round which is determined by the first column of the matrix M. The schematic view for the rest 15 bytes will be determined according to corresponding columns of the diffusion matrix M i.e. the schematic view for the i-th byte will be determined by the i-th column of the Matrix M. Now it will be shown that for the implementation for all 16 bytes in the manner described for the first byte only 43 different lookup tables will be needed instead of $15 \times 16 = 240$ which means that many lookup tables can be reused. Note that the same set of 43 lookup tables will be used for each round. Another additional 16 lookup tables will be used for the final round in order to combine split outputs and sum up the last key of SAFER+.

Below the complete list of 43 addition tables is introduced which are used to implement the matrix multiplication.

$$T_1 := Z_1(f_1^{-1}; f_2^{-1}); T_2 := Z_2(f_3^{-1}; f_4^{-1}); T_3 := Z_3(f_5^{-1}; f_6^{-1}); T_4 := Z_4(f_7^{-1}; f_8^{-1});$$

$$T_5 := Z_5(f_9^{-1}; f_{10}^{-1}); T_6 := Z_6(f_{11}^{-1}; f_{12}^{-1}); T_7 := Z_7(f_{13}^{-1}; f_{14}^{-1}); T_8 := Z_8(f_{15}^{-1}; f_{16}^{-1});$$

$$T_9 := Z_5(2f_1^{-1}; f_2^{-1}); T_{10} := Z_7(2f_3^{-1}; f_4^{-1}); T_{11} := Z_6(2f_5^{-1}; f_6^{-1});$$

$$T_{12} := Z_1(2f_7^{-1}; f_8^{-1}); T_{13} := Z_6(2f_9^{-1}; f_{10}^{-1}); T_{14} := Z_4(2f_{11}^{-1}; f_{12}^{-1});$$

$$T_{15} := Z_3(2f_{13}^{-1}; f_{14}^{-1}); T_{16} := Z_2(2f_{15}^{-1}; f_{16}^{-1}); T_{17} := Z_8(2Z_1^{-1}; Z_6^{-1});$$

$$T_{18} := Z_4(2Z_2^{-1}; Z_7^{-1}); T_{19} := Z_1(2Z_3^{-1}; Z_8^{-1}); T_{20} := Z_2(2Z_4^{-1}; Z_5^{-1});$$

$$T_{21} := Z_6(2Z_5^{-1}; Z_4^{-1}); T_{22} := Z_5(2Z_6^{-1}; Z_2^{-1}); T_{23} := Z_3(2Z_7^{-1}; Z_1^{-1});$$

$$T_{24} := Z_7(2Z_8^{-1}; Z_3^{-1}); T_{25} := Z_9(Z_5^{-1}; Z_4^{-1}); T_{26} := Z_7(Z_2^{-1}; Z_4^{-1});$$

$$T_{27} := Z_{10}(Z_3^{-1}; Z_8^{-1}); T_{28} := Z_1(Z_3^{-1}; Z_7^{-1}); T_{29} := Z_4(Z_2^{-1}; Z_5^{-1});$$

$$T_{30} := Z_{11}(Z_6^{-1}; Z_7^{-1}); T_{31} := Z_8(Z_1^{-1}; Z_3^{-1}); T_{32} := Z_8(Z_1^{-1}; Z_6^{-1});$$

$$T_{33} := Z_2(Z_5^{-1}; Z_6^{-1}); T_{34} := Z_6(Z_1^{-1}; Z_8^{-1}); T_{35} := Z_5(Z_4^{-1}; Z_7^{-1});$$
$$T_{36} := Z_{12}(Z_1^{-1}; Z_2^{-1}); T_{37} := Z_3(Z_2^{-1}; Z_8^{-1}); T_{38} := Z_4(Z_5^{-1}; Z_9^{-1});$$
$$T_{39} := Z_{14}(Z_6^{-1}; Z_{10}^{-1}); T_{40} := Z_7(Z_4^{-1}; Z_6^{-1}); T_{41} := Z_6(Z_1^{-1}; Z_{10}^{-1});$$
$$T_{42} := Z_{13}(Z_3^{-1}; Z_{10}^{-1}); T_{43} := Z_4(Z_1^{-1}; Z_5^{-1});$$

List 1: Complete list of addition tables.

The cascade implementation based on lookup tables like in Fig. 5 can be also represented via successive application for respective lookup tables taken from the List 1. For example, a cascade implementation for the byte 1 depicted in Fig. 5 can also be represented as

$$T_{25}(T_9; T_{18}(T_{26}(T_4; T_{20}(T_{22}(T_2; T_{11}); T_{14})); T_{20}(T_5; T_{18}(T_{16}; T_7)))).$$

It can be easily checked that an implementation according to the formulae (2) corresponds to the implementation according to Fig. 5.

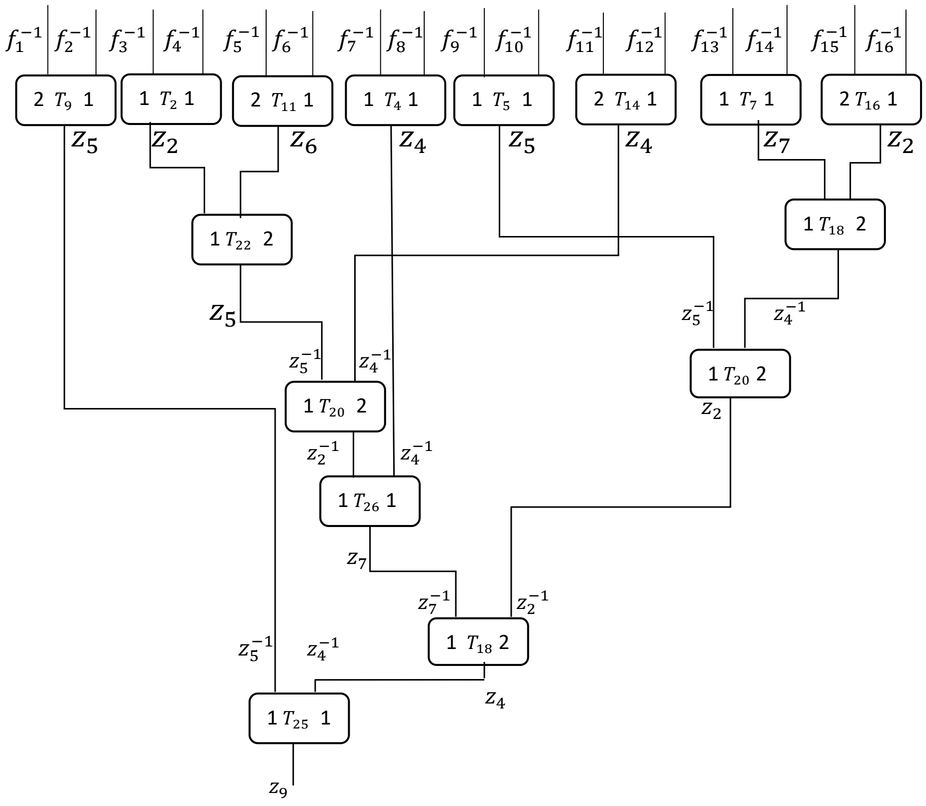


Figure 5: Cascade implementation of matrix multiplication via addition tables.

Now we are able to prove the following lemma:

Lemma 1: Cascade implementations to determine outputs for all 16 bytes by using lookup tables can be implemented as follows:

Byte1 output $T_{25}(T_9;T_{18}(T_{26}(T_4;T_{20}(T_{22}(T_2;T_{11});T_{14}));T_{20}(T_5;T_{18}(T_{16};T_7))))$

Byte2 output $T_{35}(T_{38}(T_9;T_{25}(T_{22}(T_2;T_{11});T_4));T_{26}(T_{14};T_{20}(T_5;T_{18}(T_7;T_{16}))))$

Byte3 output $T_{27}(T_8;T_{23}(T_{28}(T_{24}(T_{23}(T_1;T_{10});T_{12});T_{15});T_{24}(T_3;T_{17}(T_{13};T_6))))$

Byte4 output $T_{39}(T_{34}(T_{28}(T_{23}(T_1;T_{10});T_{24}(T_3;T_{17}T_{13:T_6})));T_3);T_{27}(T_{15};T_8))$

Byte5 output $T_{30}(T_{21}(T_{22}(T_2;T_{21}(T_9;T_4));T_{29}(T_{22}(T_{11};T_{20}(T_5;T_{14}));T_{16}));T_7)$

Byte6 output $T_{40}(T_{11};T_{29}(T_{16};T_{35}(T_{29}(T_{22}(T_2;T_{21}(T_9;T_4));T_{20}(T_5;T_{14}));T_7)))$

Byte7 output $T_{31}(T_{23}(T_{10};T_{13});T_{19}(T_{23}(T_1;T_{24}(T_3;T_{12}));T_{32}(T_6;T_{19}(T_8;T_{15}))))$

Byte8 output $T_{30}(T_{10};T_{41}(T_{27}(T_{23}(T_1;T_{24}(T_3;T_{12}));T_{32}(T_{13};T_6));T_{19}(T_{15};T_8)))$

Byte9 output $T_{25}(T_{18}(T_7;T_{16});T_{22}(T_{21}(T_4;T_{22}(T_2;T_{11}));T_{33}(T_5;T_{21}(T_9;T_{14}))))$

Byte10 output $T_{35}(T_{40}(T_{21}(T_{22}(T_2;T_{11});T_4);T_{18}(T_7;T_{16}));T_{38}(T_5;T_{25}(T_9;T_{14})))$

Byte11output $T_{27}(T_3;T_{17}(T_{19}(T_8;T_{23}(T_1;T_{10}));T_{34}(T_{12};T_{19}(T_{15};T_{17}(T_{13};T_6)))))$

Byte12output $T_{30}(T_{42}(T_3;T_{27}(T_8;T_{15}));T_{34}(T_{19}(T_{12};T_{23}(T_1;T_{10}));T_{17}(T_9;T_{14})))$

Byte13output $T_{36}(T_1;T_{20}(T_{35}(T_{10};T_{18}(T_{16};T_{24}(T_3;T_{12})));T_{18}(T_7;T_{20}(T_5;T_{14}))))$

Byte14output $T_{43}(T_1;T_{35}(T_{10};T_{29}(T_{16};T_{35}(T_{24}(T_3;T_{12});T_{18}(T_7;T_{20}(T_5;T_{14}))))))$

Byte15 output $T_{30}(T_{21}(T_9;T_4);T_{24}(T_{17}(T_6;T_{19}(T_{15};T_8));T_{37}(T_2;T_{17}(T_{11};T_{13}))))$

Byte16 output $T_{39}(T_{21}(T_9;T_4);T_{27}(T_{32}(T_{11};T_{13});T_{37}(T_2;T_{17}(T_6;T_{19}(T_{15};T_8)))))$

To prove the lemma it must be shown that cascade implementations for all 16 bytes render the same result as the multiplication of the output for the current round by the matrix M. The proof will be provided only for the first output byte. The proof for the rest of output bytes can be shown in similar way. According to SAFER+ structure in order to compute the output for the first byte of the next round the output of the current round should be multiplied with the first column of matrix M. As such an expression (2) must be computed. Note that each tables input permutations are inverse permutations applied to the output of previous E-boxes. On the other hand, in accordance with List 1 tables $T_9, T_2, T_{11}, T_4, T_5, T_{14}, T_7, T_{16}$ will be presented respectively as $Z_5(2E^r_{1,1} + E^r_{2,1}); Z_2(E^r_{3,1} + E^r_{4,1}); Z_6(2E^r_{5,1} + E^r_{6,1}); Z_4(E^r_{7,1} + E^r_{8,1}); Z_5(E^r_{9,1} + E^r_{10,1}); Z_4(2E^r_{11,1} + E^r_{12,1}); Z_7(2E^r_{13,1} + E^r_{14,1}); Z_2(2E^r_{15,1} + E^r_{16,1})$; where $E^r_{i,1}(x_{i,1}, x_{i,2})$ is the first output of respective i-th E-box. Next let us consider tables T_{18} and T_{22}. Since inputs for tables T_{18} and T_{22} will be respectively outputs of tables T_2, T_{11} and T_7, T_{16}, the outputs of tables T_{18}, T_{22} will be respectively $Z_5(E^r_{3,1}+E^r_{4,1}+4E^r_{5,1}+2E^r_{6,1})$ and $Z_4(E^r_{13,1}+E^r_{14,1}+4E^r_{15,1}+2E^r_{16,1})$. The output of table T_{22} together with output of table T_{18} are inputs for table T_{20} and the output of table T_{18} together with output of table T_5 are inputs for another table T_{20}. As such an output of two different tables T_{20} will be $Z_2(E^r_{3,1}+E^r_{4,1}+4E^r_{5,1}+2E^r_{6,1}+2E^r_{7,1}+2E^r_{8,1})$ and $Z_2(2E^r_{13,1}+2E^r_{14,1}+8E^r_{15,1}+4E^r_{16,1}+E^r_{9,1}+E^r_{10,1})$. The output one of tables T_{20} along with the output of table T_4 merges into table T_{26} which in its turn joins with the output of the next table T_{20} into the table T_{18}. Thus, at the output of table T_{18} we will have $Z_4(E^r_{3,1}+E^r_{4,1}+4E^r_{5,1}+2E^r_{6,1}+E^r_{7,1}+E^r_{8,1}+2E^r_{9,1}+2E^r_{10,1}+4E^r_{11,1}+2E^r_{12,2}+4E^r_{13,1}+4E^r_{14,1}+16E^r_{15,1}+8E^r_{16,1})$. And for the last table inputs are output of table T_{18} and table T_9.Finally the output of our last table T_{25} will be $Z_9(2E^r_{1,1}+E^r_{2,1}+E^r_{3,1}+E^r_{4,1}+4E^r_{5,1}+2E^r_{6,1}+E^r_{7,1}+E^r_{8,1}+2E^r_{9,1}+2E^r_{10,1}+4E^r_{11,1}+2E^r_{12,2}+4E^r_{13,1}+4E^r_{14,1}+16E^r_{15,1}+8E^r_{16,1})$ which is just equivalent to the result of multiplication of the first column of the matrix M with an output of the current r-th round permuted by Z_9. Note that Z_9 is compensated in the next round E-box by applying a reverse permutation Z_9^{-1}. Q.E.D.

At the end of 6^{th} round there will be 16 pairs of outputs. As indicated above 16 look-up tables will be generated which have the following functionality

$$z = FP(f_i^{-1}(x) + f_j^{-1}(y) + k_i^{13})$$

and

$$z = FP((f_i^{-1}(x) + f_j^{-1}(y)) \oplus k_i^{13})$$

where FP is a final permutation.

Similar to AES white-box SAFER+WB also uses external encoding. The latter consists of two 128-bit to 128-bit bijections, one of which called initial permutation *(IP)*, is applied on the input of the plaintext, and another - a Final permutation - *(FP)* is applied on the output of the ciphertext. Thus for external encoding two 128-bit to 128-bit initial permutation *(IP)* and *(FP)* should be generated. More precisely 16 concatenated randomly selected 8 bit to 8 bit bijections as a external encoding will be used. The purpose of external input/output encodings is to prevent attackers from exploiting specific weaknesses [Jacob at al., 2003]. Thus in order to decrypt ciphertext resulted from SAFER+WB implementation one should compute

$$IP \circ D_k \circ FP^{-1},$$

where D_k is an ordinary cipher decryption function based on a secret key k.

Lemma 2: The above described white box encryption result is equivalent to the "black- box" encryption result of SAFER+ accurate to the input and output permutations applied to the white-box input and output.

Proof: The proof is straightforward: The way how E-boxes are implemented implies that after each permutation at the output a reverse permutation is applied at the input of the next E-box which cancels the previous permutation. The same is true for the implementation of matrix multiplication implemented via cascade of addition tables in Fig. 5.

First we will prove it for rounds $r = 1 \ \ldots \ 5$. And after that we will prove it for the round 6.

Let us assume that on the input of round $r, (r = 1..5)$ the input values of SAFER+WB coincide with the input values of SAFER+. This means that the sum of two input values of SAFER+WB coincides with the input value of SAFER+. It will be proved that at the output of round r the sum of white-box output values coincides with the value of the SAFER+. Let the input values for the round r of SAFER+ be $(x_1; x_2; \cdots; x_{16})$ and the input values of SAFER+WB be $(Z_9(x_{1,1}), Z_9(x_{1,2}), Z_2(x_{2,1}), Z_2(x_{2,2}) \cdots, Z_{14}(x_{16,1}), Z_{14}(x_{16,2}))$ for the first round $(x_1; x_2; \cdots; x_{16})$. Note that $x_i = x_{i,1} + x_{i,2} \mod 256$.

According to SAFER+ structure the j-th output byte of round r is equal to

$$y_j = \begin{cases} \sum_{i=1}^{16} (\exp(x_i \oplus k^r_{i1}) + k^r_{i2}) \cdot m_{i,j} & i \in A \\ \sum_{i=1}^{16} (\log(x_i + k^r_{i1}) \oplus k^r_{i2}) \cdot m_{i,j} & i \in B \end{cases}$$

According to SAFER+WB design y_j is split into two parts as follows:

$$y^r_{1,j} = Z_9 \left(\sum_{i=1}^{16} E^r_{i,1}(x_{i,1}, x_{i,2}) \cdot m_{i,j} \right)$$

$$y^r_{2,j} = Z_9 \left(\sum_{i=1}^{16} E^r_{i,2}(x_{i,1}, x_{i,2}) \cdot m_{i,j} \right).$$

It should be mentioned that first and second type look-up tables are chosen in the way that $m_{i,j}$ is equal to the corresponding element of the Matrix M.

Now let us consider the sum of $y^r_{1,j}$ and $y^r_{2,j}$. Taking into account that output permutations is annihilated in the next round we will consider the above mentioned sum without output permutations.

$$y^r_{1,j} + y^r_{2,j} = \sum_{i=1}^{16} (E^r_{i,1}(x_{i,1}, x_{i,2}) \cdot m_{i,j} + E^r_{i,2}(x_{i,1}, x_{i,2}) \cdot m_{i,j}) =$$

$$= \sum_{i=1}^{16} (E^r_{i,1}(x_{i,1}, x_{i,2}) + E^r_{i,2}(x_{i,1}, x_{i,2})) \cdot m_{i,j} =$$

$$= \begin{cases} \sum_{i=1}^{16} (\exp((x_{i,1} + x_{i,2}) \oplus k^r_{i1}) + k^r_{i2} + R_{i1} + R_{i2}) \cdot m_{i,j} & i \in A \\ \sum_{i=1}^{16} (\log((x_{i,1} + x_{i,2}) + k^r_{i1}) \oplus k^r_{i2} + R_{i1} + R_{i2}) \cdot m_{i,j} & i \in B \end{cases}$$

$$= \begin{cases} \sum_{i=1}^{16} (\exp(x \oplus k^r_{i1}) + k^r_{i2}) \cdot m_{i,j} & i \in A \\ \sum_{i=1}^{16} (\log(x + k^r_{i1}) \oplus k^r_{i2}) \cdot m_{i,j} & i \in B \end{cases} \qquad (3)$$

Since these two inputs are added up at the beginning of the r-th ($r = 1 \dots 5$) round the result presented in (3) will be obtained, which in fact coincides with the output of the r-th ($r = 1 \dots 5$) round of SAFER+. The i-th byte of the last round output of SAFER+ has the view:

$$y_i = \begin{cases} \sum_{i=1}^{16} (\exp(x_i \oplus k^r_{i1}) + k^r_{i2}) \cdot m_{1,i} \oplus k^{r+1}_{i1} & i \in A \\ \sum_{i=1}^{16} (\log(x_i + k^r_{i1}) \oplus k^r_{i2}) \cdot m_{1,i} + k^{r+1}_{i1} & i \in B \end{cases}$$

Meantime the analogous i-th byte output for SAFER+WB has the view:

$$y_j = \begin{cases} (y_{j1} + y_{j2}) \oplus k^{r+1}_{i1} & i \in A \\ (y_{i1} + y_{i2}) + k^{r+1}_{i1} & i \in B \end{cases} =$$

$$= \begin{cases} \sum_{i=1}^{16} (E^r_{i,1}(x_{i,1}, x_{i,2}) + E^r_{i,2}(x_{i,1}, x_{i,2})) \cdot m_{i,j} \oplus k^{r+1}_{i1} & i \in A \\ \sum_{i=1}^{16} (E^r_{i,1}(x_{1,1}, x_{1,2}) + E^r_{i,2}(x_{1,1}, x_{1,2})) \cdot m_{i,j} + k^{r+1}_{i1} & i \in B \end{cases}$$

$$= \begin{cases} \sum_{i=1}^{16} (\exp((x_i) \oplus k^r_{i1}) + k^r_{i2} + R_{i1} + R_{i2}) \cdot m_{1,i} \oplus k^{r+1}_{i1} & i \in A \\ \sum_{i=1}^{16} (\log((x_i) + k^r_{i1}) \oplus k^r_{i2} + R_{i1} + R_{i2}) \cdot m_{1,i} + k^{r+1}_{i1} & i \in B \end{cases}$$

$$= \begin{cases} \sum_{i=1}^{16} (\exp(x_i \oplus k^r_{i1}) + k^r_{i2}) \cdot m_{1,i} \oplus k^{r+1}_{i1} & i \in A \\ \sum_{i=1}^{16} (\log(x_i + k^r_{i1}) \oplus k^r_{i2}) \cdot m_{1,i} + k^{r+1}_{i1} & i \in B \end{cases}$$

and coincides with the same byte output for SAFER+ after six rounds including an addition of final keys. Hence the Lemma 2 is proved.

Thus SAFER+WB encryption for each of six rounds can be accomplished by applying values of respective E-boxes as an inputs to corresponding addition tables (see Fig. 6). After six rounds of such iteration, the final ciphertext covered by Final Permutation (FP) will be the result of the application of last sixteen addition tables to the output of the sixth round.

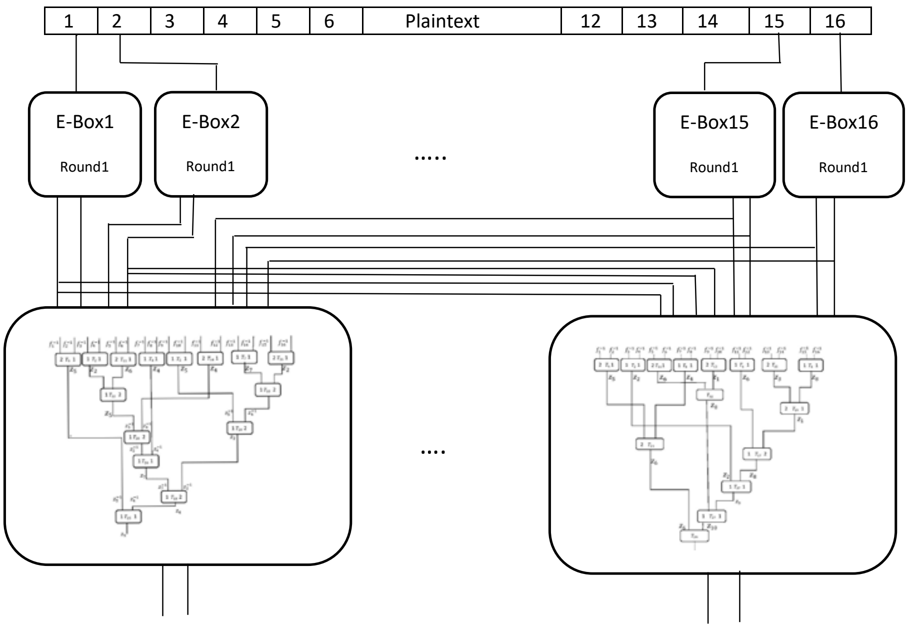


Figure 6: SAFER+ round implementation via look-up tables.

3 SAFER+ White-Box Encryption Security Analysis

Previous efforts have demonstrated a detailed cryptanalysis of SAFER+ algorithm (see [Massey et al., 1998, Nakahara et al., 2001, Zhao et al., 2013]) where it was shown that SAFER+ is secure against differential and linear cryptanalysis attacks after 6 rounds of encryption. White box encryption considered in this paper is using two additional permutations of bytes applied at the beginning and at the end of SAFER+ 6 round encryption. Clearly these 2 additional layers will not reduce the security of black box encryption from the point of view of differential and linear cryptanalysis.

As in real cryptography we assume that an attacker has an access to all E-boxes (knows all pairs of inputs and outputs) and can see all encryption steps. We also assume that a potential attacker knows all details of our SAFER+ encryption and white-box encryption steps as we have described earlier in this paper. The only thing an attacker does not know is a secret Master key. It will be shown now that an attacker having a complete information about white-box design will not be able to decrypt a randomly chosen ciphertext such that the resulting decrypted message (after passing all white-box encryption steps) will result in the same randomly chosen plaintext. Note that the major attack against white-box is a Billet et al. [Billet et al., 2003] attack which will be called as BGE attack and we will analyze why this attack can not be applied for SAFER+ WB design. Let us explain first how BGE attack works for AES.

Billet et al. [Billet et al., 2003] found that for each output encodings Q_i (Z_i for our case) of AES white-box it is possible to build an approximation $\breve{Q}_i$ that differs from it by unknown affine transformation that is $\breve{Q}_i = Q_i \circ A_i$ where A_i consists of an invertible linear transformation followed by an exclusive-or with a constant. These approximations are built by analyzing a set of look-up tables. In AES white-box round output y_0 is a function of four variables:

$$y_0 = f(x_0, x_1, x_2, x_3) = Q_0(02 \cdot T'_0(x_0) \oplus 03 \cdot T'_1(x_1) \oplus 01 \cdot T'_2(x_2) \oplus 01 \cdot T'_3(x_3)), \quad (4)$$

where $T'_i(x_i)$ is a look-up table that maps bytes to bytes and implements part of round table (AddRoundKey and Sub-Bytes transformations) and Q_0.

However if x_2, x_3 are kept constant then y_0 can be considered as a function of only x_0, x_1. For the fixed x_2, x_3 let $f_{x_1}(x_0)$ denotes the function $f(x_0, x_1, c_2, c_3)$. Let us construct a look-up table for $y_0 = f_{00}(x_0)$ and $y_0 = f_{01}(x_0)$. From equation (4) we see that $f_{00}(x_0) = Q_0(02 \cdot T'_0(x_0) \oplus \beta)$ and $f_{01}(x_0) = Q_0(02 \cdot T'_0(x_0) \oplus \beta')$ where β and β' are unknown 8 bit constants.

From the look-up tables f_{00} and f_{01} one can construct a look-up table for $f_{00} \circ f_{01}^{-1}$ which has the following structure: $f_{00} \circ f_{01}^{-1} = Q_0(Q_0^{-1} \oplus \beta'')$ where $\beta'' = \beta' + \beta$. By changing the value of β'' it is possible to construct 256 bijections of the form $Q_0(Q_0^{-1} \oplus \beta)$. Using this 256 bijections a new bijection $\breve{Q}_i = Q_i \circ A_i$ can be constructed, which differs from Q_i by an unknown affine transformation. Then applying this bijection, it is possible to simplify the output encodings. The simplification is achieved by composing new output encodings with the existing ones. After each Q_i the bijection $\breve{Q}_i^{-1}$ is applied. These two bijections are composed to render

$$\breve{Q}_i^{-1} \circ Q_i = A_i^{-1} \circ Q_i^{-1} \circ Q_i = A_i^{-1}$$

which is in fact an affine bijection. Note that output encodings for a given round correspond to input encodings for the subsequent round and the output encoding approximations also lead to input encoding approximations. Therefore the input encodings can also be simplified. Each Q_i has the form $M_i(x) \oplus q_i$, where M_i is a linear bijection and q_i is an 8-bit constant. The next step in the attack recovers M_i and q_i (see the original paper for details). As a result, the value of the input and output encodings as well as outputs of the T-boxes can be determined completely. By possessing input and output encodings an attacker can easily recover key byte from a look-up table.

For SAFER+WB output y_0 is a function of sixteen variables which in fact can be presented as 32 variables since according to our white-box construction every output is split into two outputs. Thus the analogue of the equation (4) in the case of our white-box will be
$y_{01} = Z_9\left(\sum_{j=0}^{16}(m_{i,0}T_{j1}^r(x_1,x_2))\right)$ and $y_{02} = Z_9\left(\sum_{j=0}^{16}(m_{i,1}T_{j2}^r(x_1,x_2))\right)$.

In fact according to SAFER+WB construction instead of y_0 we have y_{01} and y_{02} where $y_0 = y_{01} + y_{02}$. If 31 inputs of the round input are kept constant then we have $y_{01} = f_c(x_0) = Z_9(m_{i,0}T_{i1}^r(x_1,c_1) + \beta)$ where a value c_1 corresponds to the second input for the first byte and β corresponds to constant values for remaining 30 inputs. Changing one of 30 values for the same constant c_1 will result $f_{c1}(x_0) = Z_t(m_{i,0}T_{i1}^r(x_1,c_1) + \beta')$. In order to mount BGE attack in analogue with AES cipher we need to construct $f_c \circ f_{c1}^{-1} = Z_t(Z_t^{-1} + \beta'')$ where $\beta'' = \beta + \beta' \mod 256$. However $E_{i1}^r(x_1,c)$ is not a bijection due to the condition (2) which implies that any byte at the input of such permutation has an output which is restricted to have a value not more than 128. Therefore $y_{01} = f_{c1}(x_0)$ is not a bijection and consequently $f_c \circ f_{c1}^{-1} = h_0(h_0^{-1} + \beta'')$ cannot be a bijection as well. Thus it is impossible to construct an approximation bijection that differs from the real one by an affine transformation which will allow to mount an attack described in [Billet et al., 2003].

One way-ness of SAFER+ based white-box encryption: One way-ness feature prevents an attacker to perform a decryption operation based on E-box information without knowing actual keys used inside each box. This kind of attack can also be called a reverse Engineering attack. Let us analyze the complexity of making correct decryption operation for the rounds $r \leq 6$. This is equivalent to determine the output and input permutations

$$f_1(), f_2(), \ldots, f_{16}() \quad \text{and}$$

$$Z_9; Z_2; Z_{10}; Z_{14}; Z_{11}; Z_7; Z_8; Z_{11}; Z_9; Z_5; Z_{10}; Z_{11}; Z_{12}; Z_4; Z_{11}; Z_{14} \qquad (5)$$

used in E-boxes. Note that indexes for Z-permutations correspond to respective bytes, for example Z_{10} is the permutation used for 3-th and 11-th bytes as it follows from (5). If these outputs are determined there is no need to know what keys are used inside of each box and we simply can determine E-box inputs based on their outputs and as such to get decryption implemented for the current round. Thus the complexity of determination for the given $x_i (i = 1; 16)$ E-boxes outputs $f_1(), f_2(), \ldots, f_{16}()$ needs to be estimated. Let us assume that (a) all values of $Z^{-1}(x_i)$ are known, which is equivalent to know all secret permutations used for the last tables of transformation, and (b) all 16 secret permutations $f_1(), f_2(), \ldots, f_{16}()$ used at the outputs of the E-boxes are known. In that case the vector $Z_9^{-1}(y_1), Z_2^{-1}(y_1), \ldots, Z_{14}^{-1}(y_{16})$ could be multiplied by the reverse matrix M^{-1} of SAFER+ linear transformation. Finally, it would be possible to obtain $f_i(E_{i1}^r(x_1, x_2))$ i.e. output values for E-boxes by using the knowledge of all 16 permutation functions $f_i()$. Now it must be shown that it is impossible to extract any information about the above mentioned permutations by using information provided by all the addition tables. Let us start with table 1 that has two inputs $f_1()$ and $f_2()$ and one output $Z_1()$ The white-box ambiguity for the given table is the number of distinct constructions resulting in the same table. This number should be large enough not to allow finding the permutations through exhaustive search. By fixing any two of the f_1, f_2 and Z_1 permutations, the third permutation always can be chosen in order to get a specified addition table. However, given one input permutation and the value of a specified element transformed by the other input permutation, the third permutation can easily be found. Thus the white-box ambiguity of such tables is 256!*256, which makes the exhaustive search infeasible. The same reasoning applies to any of 15 similar tables. On the other hand, if one of the tables is using permutations f_i, f_j at the input and f_{ij} permutation at the output, and if any 2 out of 3 of these permutations are the same, then it can be shown that not more than 256 permutation values will be found for a given table. This is the major reason why all lookup tables used in cascade implementation are chosen to have different permutations for both inputs and an output.

4 Complexity of Implementation and Memory Requirements

Our reference software implementation for SAFER+WB showed that performance degradation compared with corresponding SAFER+ black box implementation is about 20%. Usually the performance estimation is made in terms of number of clock cycles. However in the case of white-box it is a common practice to made a performance estimation based on number of look-up tables used during implementation, as well as on the number of other operations such as “xor” or other.

In fact during SAFER+WB implementation only table lookup operations are used. For each round 16 table lookup operations are used for implementing E-boxes and for each round‘s output bytes 15 table lookup operations are used in order to carry out matrix multiplication operation. Taking into account that SAFER+WB consists of 6 rounds and each round has 32 output bytes $6(16 + 32 \times 15) = 2976$ successive table lookup operations will be required in the case if no parallel processing is possible. In this case the similar operations can be implemented by using $6(1+1\times 5)=36$ successive lookup operations at the cost of 32 parallel processors. As follows from the Fig. 5 by using parallel processors a cascade implementation of calculating an output byte can be carried out in 5 successive steps.

Now let us estimate a memory required for SAFER+WB implementation.

SAFER+WB encryption uses 96 E-box tables. First 16 E-box tables for the first round have the size of 256×2 bytes. Another 80 E-box tables for rounds 2 to 6 each of them having size of $256\times 256\times 2bytes = 128KB$. The size of each addition box is $65536\ bytes = 64KB$, and there are $43+16$ boxes in total. Thus the total required memory size will be approximately $14MB$.

5 Conclusions and Future Work

In this paper a novel white-box design based on SAFER+ is presented. It is shown that the new design called as SAFER+WB is secure against the so called BGE attack presented in [Billet et al., 2003]. It is also shown that SAFER+WB is secure against so called reverse Engineering attack. Implementation speed and memory requirements for SAFER+WB are also provided. In the next step of our research we will focus on the integration of SAFER+WB with DRM and public-key systems.

References

[Billet et al., 2003] Billet, O., Gilbert, H., Ech-Chatbi, Ch.(2004). Cryptanalysis of a White-Box AES Implementation. In Selected Areas in Cryptography (SAC 2004), Lecture Notes in Computer Science, vol 3357. Springer, Berlin, Heidelberg 2004, 227-240.

[Bringer et al., 2006] Bringer, J., Chabanne, H., Dottax, E.(2006). White box Cryptography: Another Attempt, Cryptology ePrint Archive, Report 2006/468, https://eprint.iacr.org/2006/468.pdf.

[Chow et al., 2003] Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.(2002). A White-Box DES Implementation for DRM Applications, In: Feigenbaum, J.(ed.) DRM 2002. LNCS, vol. 2696, Springer, Heidelberg, 2003, 1-15.

[Chow et al., 2002] Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C(2002). White-Box Cryptography and an AES Implementation. In: Nyberg K., Heys H. (eds) Selected Areas in Cryptography. SAC 2002. Lecture Notes in Computer Science, vol 2595, 250-270.

[Jacob at al., 2003] Jacob, M., Boneh, D., Felten, E.W(2002). Attacking an Obfuscated Cipher by Injecting Faults. In Feigenbaum, J., ed.: Digital Rights Management DRM 2002. Volume 2696 of Lecture Notes in Computer Science, Springer Verlag, 2003, 16-31.

[Karrroumi, 2011] Karroumi, M.(2010). Protecting White-Box AES with Dual Ciphers. In: Rhee KH., Nyang D. (eds) Information Security and Cryptology - ICISC 2010. ICISC 2010. Lecture Notes in Computer Science, vol 6829. Springer, Berlin, Heidelberg, 2011, 278-291.

[Khachatrian at al., 2016] Khachatrian, G., Karapetyan, M.,(2016). White-Box Encryption Algorithm Based on SAFER+, in proceedings of international workshop of Information theory and Data science From information age to Big data Era, Yerevan Armenia, October 3-5, 2016, 77-88.

[Lepoint et al., 2013] Lepoint, T., Rivain, M.(2013). Another Nail in the Coffin of White-Box AES Implementations, Cryptology ePrint Archive, Report 2013/455, http://eprint.iacr.org/2013/455.pdf, 2013.

[Lepoint et al., 2014] Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.(2014). Two Attacks on a White-Box AES Implementation. In: Lange T., Lauter K., Lisonk P. (eds) Selected Areas in Cryptography –SAC 2013. Lecture Notes in Computer Science, vol 8282. Springer, Berlin, Heidelberg, 265-285.

[Massey et al., 1998] Massey, J., Khachatrian, G., Kuregian, M.(1998) Nomination of SAFER+ as a Candidate Algorithm for Advanced Encryption Standard (AES), Represented at the first AES conference, Ventura, USA, August 20-25, 1998.

[Mulder et al., 2013a] De Mulder, Y., Roelse, P., Preneel, B.(2013). Cryptanalysis of the Xiao Lai White-Box AES Implementation. In: Knudsen L.R., Wu H. (eds) Selected Areas in Cryptography. SAC 2012. Lecture Notes in Computer Science, vol 7707. Springer, Berlin, Heidelberg, 34-49.

[Mulder et al., 2013b] De Mulder, Y., Roelse, P., Preneel, B.(2013). Revisiting the BGE Attack on a White-Box AES Implementation, IACR Cryptology ePrint Archive, 2013, https://eprint.iacr.org/2013/450.pdf.

[Nakahara et al., 2001] Nakahara, J., Prenel, B., Vandewalle, J.(2001). Linear Cryptanalysis of Reduced-Round Versions of the SAFER Block Cipher Family. In: Goos G., Hartmanis J., van Leeuwen J., Schneier B. (eds) Fast Software Encryption. FSE 2000. Lecture Notes in Computer Science, vol 1978. Springer, Berlin, Heidelberg, 244-261

[Yaying et al., 2009] Yaying, X., Xuejia, L.(2009) A Secure Implementation of White-Box AES, In Computer Science and its Applications, 2009.CSA09. 2nd International Conference on, pages 1-6, 2009

[Zhao et al., 2013] Zhao, J., Wang, M., Chen, J., Zheng, Y.(2013) New Impossible Differential Attack on SAFER+ and SAFER++.Lecture Notes in Computer Science, vol 7839, Information security and cryptology von Springer, Heidelberg, 2013, 170-183.