INSTITUTE FOR INFORMATICS AND AUTOMATION PROBLEMS OF NAS RA

Martun M. Karapetyan

WHITE-BOX CRYPTOGRAPHY AND APPLICATIONS

Thesis

For obtaining candidate degree in technical sciences in specialty 13.05 “Mathematical modeling, numerical methods and program complexes”

Scientific adviser: Dr. of tech. sc., prof. Gurgen H. Khachatryan

Yerevan 2017

Contents

INTRODUCTION

Relevance of the Subject

Conventional encryption algorithms are designed to be secure in the "black-box" attack context, i.e. the attacker can observe the input and output of the encryption algorithm, but cannot access the intermediate values generated during the software execution.

Yet in some cases, the encryption algorithm runs in a hostile environment, where the attacker can see not only the input and output values but also has full access to all the intermediate values generated during the software execution and can change the execution at will. This means that an attacker can use different emulators, disassemblers and debuggers such as IDA Pro system and others to analyze binary code of the software, see values of variables during execution and change the execution routine. Any cryptographic software running on these type of devices is operating in the white-box attack context (later WBAC), which means that conventional black-box ciphers don't provide the necessary level of security for these devices.

White-box (later WB) cryptography algorithms are designed to be executed in such untrusted environments and are said to operate in the WBAC. These algorithms are based on the black-box algorithms, but instead of using the secure cryptographic key directly, special look-up tables are built based on the key and the algorithm runs using those tables. All the operations are performed using these tables in such a clever way, that the attacker having access to all the tables and being able to observe the process of each encryption operation and even modify it at will, is unable to extract the cryptographic key used for the construction of the tables.

WB cryptography is a crucial part of several security-centric commercial products, such as Digital Rights Management (later DRM) applications. It is used to effectively protect cryptographic keys, which has often been the Achilles heel in DRM systems in the past. Other benefits of using a WB algorithm are software traceability, tamper resistance, renewability, and so on. This makes WB algorithms the most crucial element in the modern DRM systems, such as iTunes, Amazon [**1**], and others. The initial versions of iTunes employed black-box encryption algorithms and were broken in a matter of days, stopping

Apple to sell movies on iTunes. In 2005, WB algorithms were used to increase the level of difficulty of breaking the "FairPlay", thus fixing the problem of content distribution control.

Wide usage of cloud storage systems providers like Dropbox [**2**], Google Drive [**3**], Box [**4**] and Microsoft OneDrive [**5**] for storing emails, multimedia files, different records like financial transactions, healthcare, tax documents, etc. resulted to the spread of *data as a service* (DaaS) business model [**6**], [**7**]. The motivation for DaaS architecture was the *software as a service* (SaaS) [**8**] architecture. In SaaS software deployment model the software hosted on a web server is providing a service to manage your data. This way the user of the DaaS system does not need to maintain the software and hardware infrastructure, as it was for the conventional database systems. These DaaS providers have access to all of the user's files, which is a concern for some users. This gave birth to cloud encryption gateways like Sookasa [**9**], nCryptedCloud [**10**], Boxcryptor [**11**] and SkyCryptor [**12**], startups that provide encryption services to the users who want to keep their files secure, while still storing them in the clouds of their choice. Users of these startups end up storing only encrypted files on the cloud, thus taking away the ability of the provider to access all the files.

The ability to search over this big encrypted data is a crucial feature of such service providers. Though each user's data is stored in an encrypted form on the servers of the DaaS provider company, given a search token, the users are able to search over their files in a secure manner. So the users are able to perform the file addition, modification, deletion, search and file sharing operations without revealing any information about the content of his/her files or the search keyword itself. Traditionally public key (later PK) encryption algorithms were used in Searchable Encryption (later SE) schemes, yet WB algorithms can replace them to provide efficient search functionality.

Another important application of WB cryptography is Oblivious transfer (later OT) protocols. OT is an important cryptographic primitive, which has different applications and plays an essential role in secure multiparty computation protocols. Most existing OT protocols are based on public-key operations, which have slow execution speeds, making the protocol infeasible to be used in practical applications. In the recent result [**13**] a novel

OT protocol is presented, which replaces the public-key operations with WB cryptography techniques resulting to sufficient performance improvement. The protocol also requires less communication bandwidth. Such optimizations make secure function evaluation (later SFE) applications practical for real-life applications.

WB algorithms can also be used as a fast alternative to public key cryptography in most of applications, resulting to significant performance improvements.

The first WB implementation of AES [**14**] cipher was presented by Chow, Eisen, Johnson and van Oorschot in 2002 [**15**]. The steps of AES algorithm were implemented using WB look-up tables. WB diversity and ambiguity criteria were first introduced, and diversity and ambiguity of all the tables used in the implementation were computed by the authors. The security of each table used was proved. Yet the authors stated that the security of a group of tables is not analyzed and needs additional research.

Chow's AES WB was shown to be vulnerable to the BGE attack presented by Billet, Gilbert and Ech-Chatbi in 2004 [**16**]. This attack targets a group of tables from Chow's WB, viewing AES round as four look-up tables, each mapping 4-bytes of input state to 4-bytes of output state. This group of tables providing 4-byte to 4-byte mapping did not have the same diversity and ambiguity properties as each particular table making the group. So it was impossible to break each box separately, but a group of tables was successfully attacked. The attack had a work factor of 2^{30}, so could successfully extract the secure key from Chow's white-boxes in a matter of seconds.

In 2009 another attempt was made to build a secure AES WB by Yaying Xiao and Xuejia Lai [**17**]. Xiao and Lai used the same structure of Chow's tables, but instead of cancelling out the mixing bijections (later MB) in the same round, tried to cancel them in the following round. This would make the attacker analyze tables of all the rounds together, protecting Chow's tables from the BGE attack. Suddenly, this implementation was also broken in 2013 by Yoni Mulder, Peter Roelse, and Bart Preneel [**18**].

In 2010, another WB AES implementation was presented by Karroumi [**19**], which was supposed to withstand the BGE attack. Karroumi used dual AES ciphers [**20**], [**21**] to build the WB tables. In 2013, De Mulder, Roelse, and Preneel showed, that Karroumi's and

Chow's implementations are equivalent, i.e. the BGE attack can be successfully applied to both [**22**].

In 2013, yet another successful attack was introduced by T.Lepoint and M. Rivain [**23**], which could be used to attack both Chow's and Karroumi's implementation with 2^{22} work steps. The attack was based on collisions on some table outputs. This new attack was not only faster, compared to BGE attack, but was also much easier to understand and implement.

Another approach to practically break WB algorithm was recently introduced by Joppe W. Bos and Charles Hubain and Wil Michiels and Philippe Teuwen which uses Differential Computation Analysis [**24**], the software version of the differential power analysis [**25**].

These WB implementations and effective attacks were researched, and a brief review of each algorithm and attack is presented in this dissertation.

The first part of the research done in the scope of this dissertation refers to the creation of a safe symmetric WB encryption algorithm based on Safer+ block cipher [**26**]. This algorithm can replace its black-box alternative on the smartphones and other vulnerable devices as well as get used in the search over encrypted data systems. Other important applications include usage of the WB algorithm in oblivious transfer (later OT) protocols and as a faster alternative to conventional public key encryption algorithms. As all existing WB schemes ([**15**], [**17**], [**19**]) were successfully attacked by different methods, the presented algorithm is the only secure WB implementation to use. Performance analyses and memory requirements are provided, showing high performance of the algorithm. Security analyses were provided, describing several attacks on weakened variants and comments regarding security against BGE attack.

A pioneering work by Diffie and Hellman [**27**] describing Diffie-Hellman key exchange gave birth to the public key (later PK) cryptography. A fundamental work by Rivest, Shamir and Adleman [**28**] presented the RSA cryptosystem. Another important development for PK cryptosystems was the invention of Elliptic curve (later EC) cryptosystems [**29**].

The second part of this dissertation describes a white-box PK encryption algorithm based on permutation polynomials. This algorithm can safely replace the conventional public key

schemes like RSA [**28**] and EC [**29**] and is significantly faster compared to them, both for encryption and decryption operations. The cryptosystem was first proposed by G. Khachatrian, M. Kuregian in [**30**]. In current dissertation we provide modifications for this system to improve the security and performance. The resulting algorithm is faster compared to its predecessor at the price of slightly higher memory requirements. Several implementation details are presented and performance analyses are made comparing it to its predecessor and rival public key cryptosystems.

Finally, we provide some details of integration of the WB algorithms into the Secure Search engine of SkyCryptor [**12**].

Objects of the Research

There were several attempts to build a secure WB implementation of a symmetric block cipher, but all the existing algorithms were later successfully attacked with different methods. Our research concentrated on those implementations and known attacks in effort to build a novel WB cryptosystem which will resist all the attacks that were successful on similar schemes.

Several WB implementations of AES block cipher [**14**] were researched including implementations of Chow, Eisen, Johnson and van Oorschot [**15**], Karroumi [**19**], Yaying Xiao and Xuejia Lai [**17**].

Different attacks on the existing ciphers were researched including the BGE attack [**16**], its modification [**22**] and the collision attack by T.Lepoint, M. Rivain [**23**].

The PK encryption scheme proposed in [**30**] was researched and improved for higher security and better performance.

Research and Implementation Methods

In the research general methods and concepts of classical cryptography are used.

C++ programming language and CryptoPP **[**31**]** library were used to implement the algorithms and test them. Performance tests were made for comparing the speed of presented schemes with the rival algorithms.

Scientific Novelty

1. A novel white-box algorithm was created based on Safer+ block cipher. The general security analyses of the cryptosystem were presented, including comments regarding resistance to the widely successful BGE attack [**16**].
2. The public key encryption algorithm based on permutation polynomials [**30**] was modified for improved security and performance. The algorithm uses WB reduction for encryption operation.

Practical Significance

The white-box (WB) algorithm based on Safer+ presented in this dissertation can be used in different Digital Right Management (later DRM) systems, including video-on-demand services and applications requiring distribution control for protected content playback. Symmetric WB encryption is a crucial part of digital media distribution from iTunes, Amazon Prime Video.

Oblivious transfer (later OT) protocol has a number of applications including an essential role in secure multiparty computation protocols. Yet most existing OT protocols are based on computationally ineffective public key operations. In the recent result [**13**] a novel OT protocols is introduced, which replaces usages of PK operations with WB cryptography techniques which results to significant performance improvements. The resulting algorithm runs several magnitudes of orders faster and requires less communication bandwidth compared with the older protocols. Yet this approach required usage of a secure WB encryption algorithm, which didn't exist before, since all the candidates got broken in a few years after getting published. The WB algorithm presented in this dissertation can be safely used in the OT protocol described.

Searchable encryption (later SE) is a technique for storing user's files on a server in an encrypted form without sacrificing the user experience of accessing the data. In recent years many practical schemes have been developed for performing SE with optimal search time and efficient memory consumption with reasonable security leakage. The most practical schemes are based on the secure index approach. In [**32**] a method for

constructing delegable (multi-user) SE scheme is presented which uses WB algorithm for symmetric encryption. This is yet another practical application to the secure WB based on Safer+ cipher presented.

The improved PK encryption scheme based on permutation polynomials can replace existing public key systems like RSA [**28**], Diffie-Helman [**27**] or Elliptic curve cryptosystems [**29**]. Replacing the conventional public key systems with their WB alternative presented results to significant performance improvement in different applications.

Practical Implementation

Based on the results of the research a white-box (WB) algorithm based on Safer+ and another WB algorithm based on permutation polynomials were implemented and tested. The algorithms were integrated into SkyCryptor's [**12**] secure search engine. SkyCryptor is a startup, which provides zero-knowledge encryption for users who want to store their files in any cloud in an encrypted form. SkyCryptor relies on the WB algorithms presented in this dissertation for keyword encryption and proxy re-encryption operations, which are crucial for searching over the user's encrypted data and file sharing features.

The corresponding implementation certificate is appended to the dissertation.

Provisions Presented to the Defense

- A novel white-box algorithm based on SAFER+ block cipher, secure against the widely successful BGE attack.
- An improved public key encryption scheme based on permutation polynomials with white-box reduction on encryption procedure.
- A C++ library which implements the novel SAFER+ white-box algorithm for proof of concept and performance testing.
- A C++ library which implements the new PK system based on permutation polynomials.

Approbation

The results of the work on dissertation have been presented at international workshop on Information theory and Data Science, "From Information Age to Big Data Era", Yerevan, Armenia, October 3-5, 2016.

Publications

The main topics of this dissertation are published in three publications:

[**33**], [**34**], [**35**].

CHAPTER 1

PRIOR ART AND BASE FOR THE CONTRIBUTION

For a fixed plaintext space M, key space K and ciphertext space C, a black-box cipher is a list of 3 algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, where Gen randomly selects a key k∈K, $\mathrm{Enc}(k; M) \stackrel{\text{def}}{=} \mathrm{Enc}_k(M)$ maps M into C with respect to a selected key k, and $\mathrm{Dec}(k; C) \stackrel{\text{def}}{=} \mathrm{Dec}_k(C)$ maps C into M in such a way, that for any given $k \in K$ and $m \in M$, $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$.

A WB encryption/decryption scheme corresponding to the block cipher algorithm $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is a list of four $(\mathrm{WBGenEnc}, \mathrm{WBGenDec}, \mathrm{WBEnc}, \mathrm{WBDec})$ algorithms such that given the secret key $k \in \boldsymbol{K}$, algorithms operate as follows:

- *WB Encryption Tables T_{enc} are generated from k with algorithm WBGenEnc:*

$$T_{enc} \leftarrow \mathrm{WBGenEnc}\,(K)$$

- *WB Decryption Tables T_{dec} are generated from k with algorithm WBGenDec:*

$$T_{dec} \leftarrow \mathrm{WBGenDec}\,(K)$$

- Plaintext $m \in \boldsymbol{M}$ can be encrypted to the corresponding ciphertext c given the WB tables T_{enc} using WBEnc:

$$c \leftarrow \mathrm{WBEnc}_{T_{enc}}(m)$$

- The ciphertext c can be decrypted to the plaintext $m \in M$ using the T_{dec} and algorithm WBDec:

$$m \leftarrow \mathrm{WBDec}_{T_{dec}}(c)$$

- For any $m \in \boldsymbol{M}$, $T_{enc} \leftarrow \mathrm{WBGenEnc}\,(k)$ the following statement holds:

$$m = \mathrm{Dec}_K(\mathrm{WBEnc}_{T_{enc}}(m))$$

- For any $m \in \boldsymbol{M}$, $T_{dec} \leftarrow \mathrm{WBGenDec}\,(k)$ the following statement holds:

$$m = \mathrm{WBDec}_{T_{dec}}(\mathrm{Enc}_k(m))$$

In some cases only the encryption boxes are considered, i.e. the algorithms WBGenEnc and WBEnc are presented, because in most of the applications WB encryption is done on client devices, and decryption is done on the secure server, i.e. the decryption algorithm runs in a protected environment and does not need to be WB secure.

In this chapter we present the existing WB implementations of AES symmetric encryption algorithm and known successful attacks on them. All known WB AES implementations were

successfully attacked. Later we present a novel WB implementation of SAFER+ algorithm, which can safely replace AES WB in all of its applications.

1.1 WHITE-BOX IMPLEMENETATIONS OF AES AND KNOWN ATTACKS

There were numerous attempts to design a WB implementation of the Advanced Encryption Standard (later AES) or Data Encryption Standard (later DES), all of which were later broken.

The first WB implementation of AES block cipher was first presented in 2002 [**15**]. The BGE attack [**16**] presented in 2004 showed that the secure master key can be extracted from Chow's implementation in 2^{30} work steps [**16**].

A WB implementation of DES [**36**] was also presented by Chow, Eisen, Johnson, and van Oorschot in 2002 [**37**], which was broken in [**38**], [**39**]. Another attempt was made by Hamilton and Neumann [**40**], which was trying to fix vulnerabilities in Chow's DES implementation. Yet several other attacks on DES were introduced later, which operate on all implementations of DES [**41**], [**42**].

In 2006, perturbed AES tables were presented [**43**], which were broken in [**44**].

In 2009 another attempt was made to build a secure AES WB by Yaying Xiao and Xuejia Lai [**17**], broken in 2013 by Yoni Mulder, Peter Roelse, and Bart Preneel [**18**].

In 2010, Karroumi presented a modified version of Chow's algorithm based on dual ciphers, which was designed to withstand the BGE attack and increase the work factory of it to 2^{93} [**19**]. In 2013, De Mulder, Roelse and Preneel proved that Karroumi's and Chow's implementations are identical and the BGE attack can be applied to Karroumi's implementation with minor modifications [**22**]. Also several speed optimizations were presented, after which just 2^{22} work steps are required to extract the key.

In 2013, yet another successful attack was introduced by T.Lepoint and M. Rivain [**23**], [**45**], which could be used to attack both Chow's and Karroumi's implementation with a work factor of 2^{22} work steps.

Another work was presented in 2009, describing an attack on any WB implementation of any SLT cipher which satisfies the given condition on diffusion matrix [**46**].

In 2013, a paper was published describing the general security notion expected from a WB algorithm. The authors came up with several desired security properties for WB programs, such as traceability, one-wayness and incompressibility[**47**].

Despite the general results concerning impossibility of software program obfuscation [**48**], the implementation of secure WB for block ciphers can be possible. The fact that all publicly known white-boxes have been successfully broken does not proof the impossibility of existence of another WB algorithm for another block cipher.

1.1.1 AES BLACK-BOX ENCRYPTION

AES is a substitution-permutation network cipher for symmetric encryption also known as the Rijndael cipher [**14**]. It supports key lengths of 128, 192 or 256 bits and has 10, 12 or 14 rounds, respectively. AES-128 will be considered as the primary setting in the rest of this paper. Each round updates a 16-byte state and consists of four operations: SubBytes, ShiftRows, MixColumns and AddRoundKey, except the final round, where the MixColumns operation is omitted.

ShiftRows transformation shifts 2nd, 3rd and 4th rows of the table 1, 2 and 3 times to the left, respectively.

MixCoumns is a transformation applied on the columns of the state matrix. Each column is multiplied with a fixed matrix $MC = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$.

AddRoundKey operation simply XORs the state with the next key, generated by the key schedule.

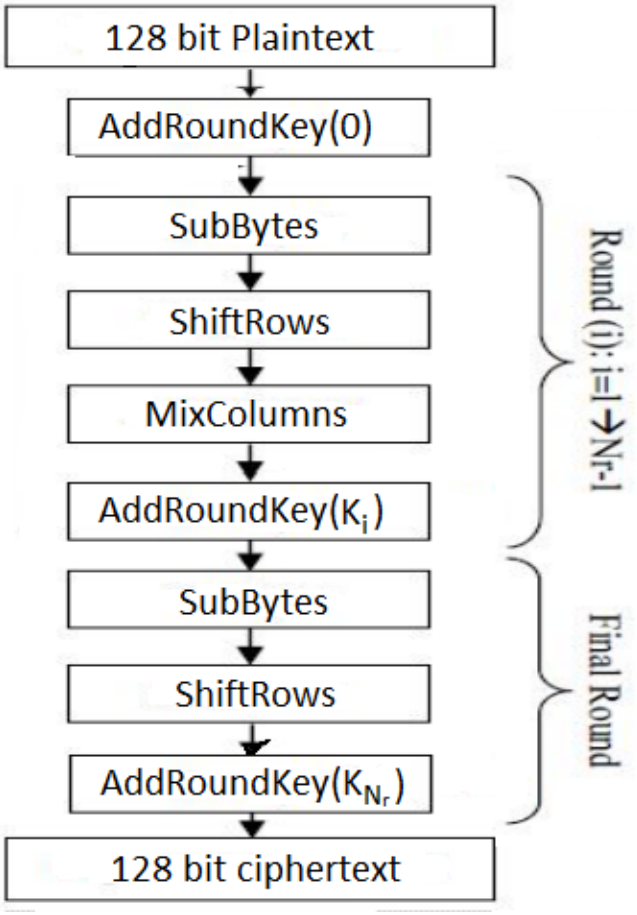

Figure 1.1: Structure of AES black-box encryption [**14**].

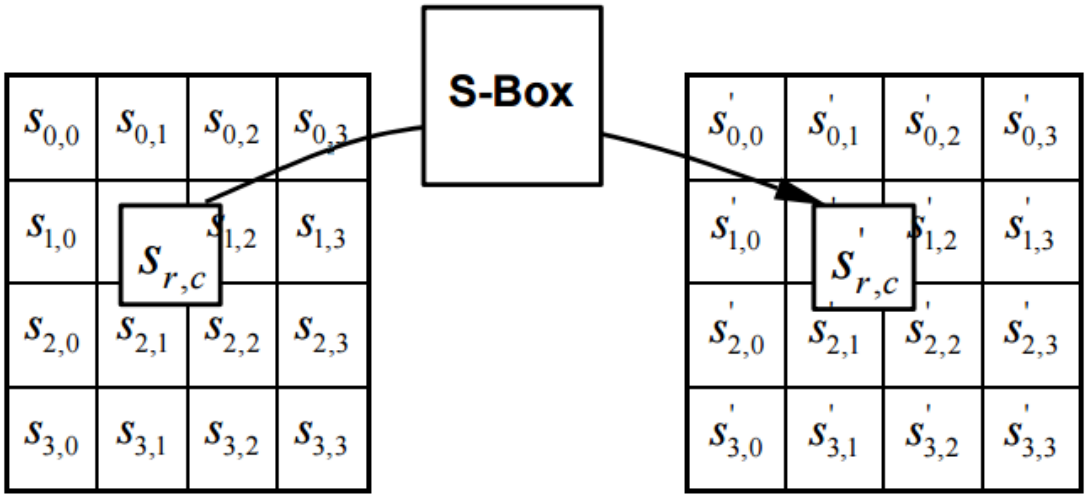

Figure 1.2: AES SubBytes operation [**14**].

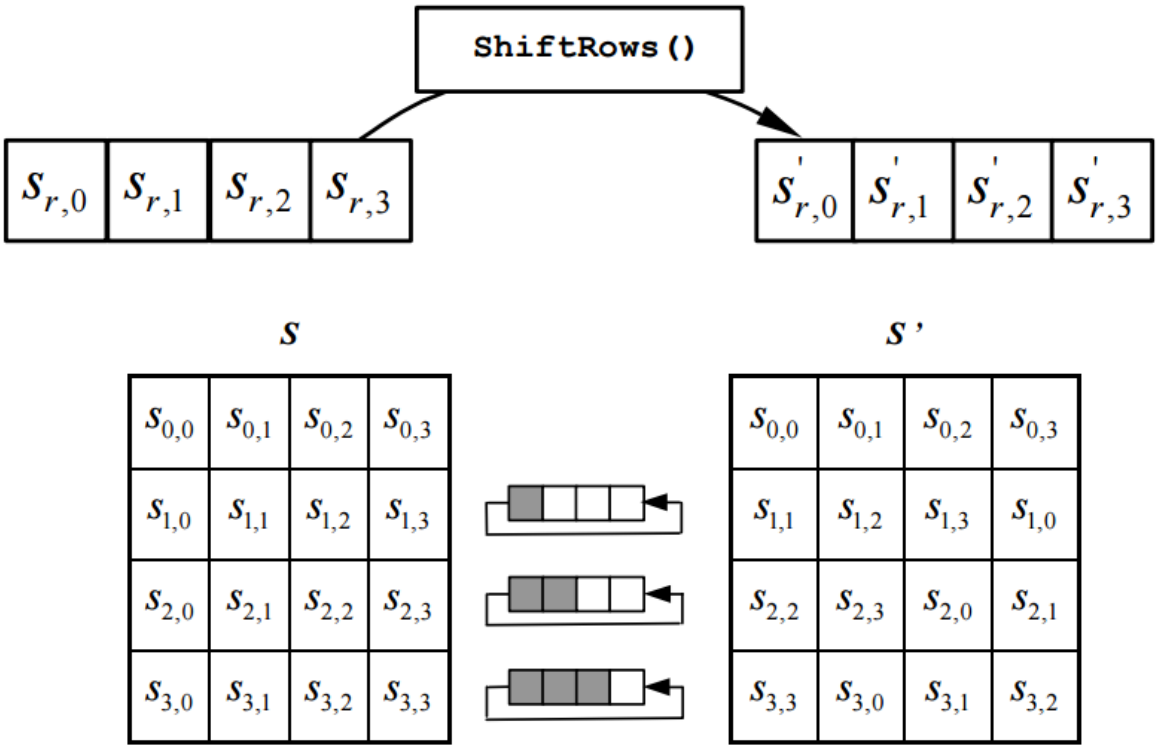

Figure 1.3: AES ShiftRows operation [**14**].

For purpose of creating a WB implementation for AES, we can view the algorithm in a different manner. One can notice that SubBytes and ShiftRows operations can be safely switched without any change in the output. Also because the ShiftRows is a linear transformation,

$\mathrm{AddRoundKey}(K_i)$ followed by ShiftRows is identical to ShiftRows followed by $\mathrm{AddRoundKey}(K_i')$, where K_i' is the result of applying ShiftRows operations on K_i. These allow us to change the AES structure to the one in Figure 1.5.

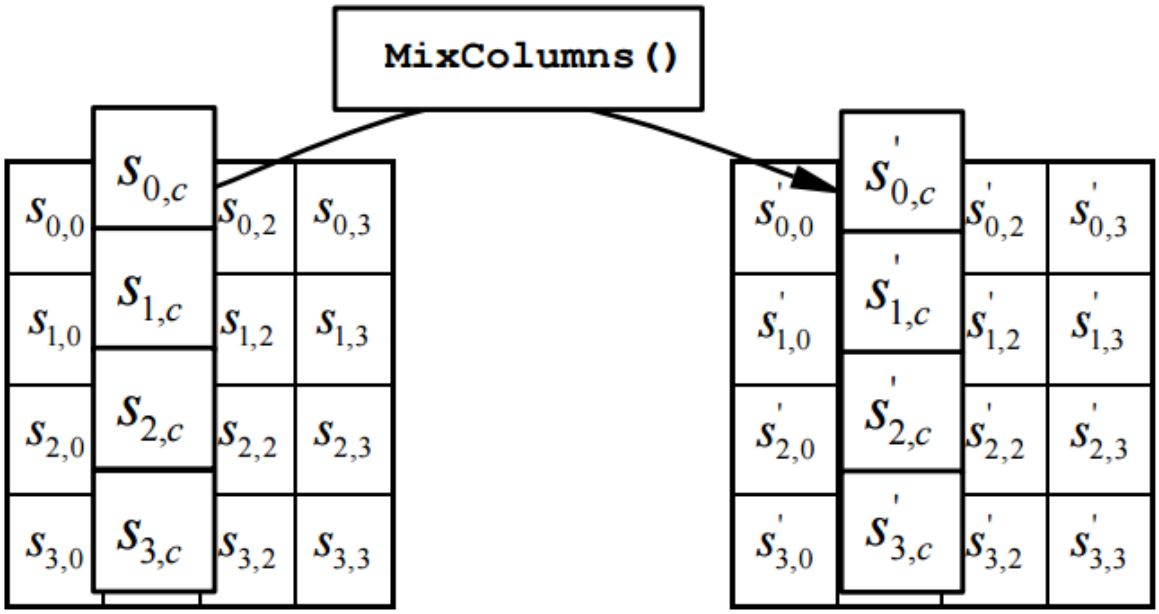

Figure 1.4: AES MixColumns operation [**14**].

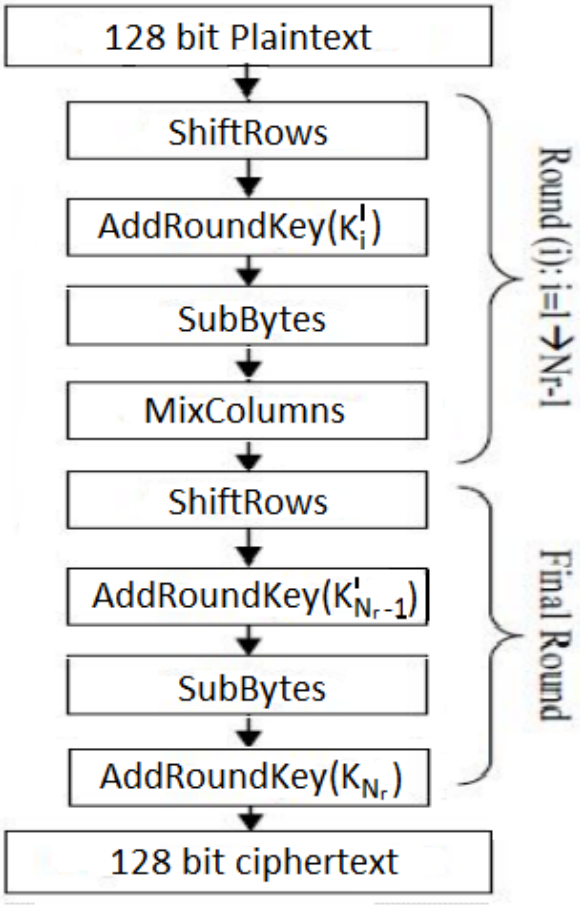

Figure 1.5: Modified structure of AES algorithm [**14**].

1.1.2 CHOW'S AES WHITE-BOX IMPLEMENTATION

In 2002, Chow, Eisen, Johnson and van Oorschot proposed the first WB implementation of AES-128 [**15**]. Instead of calculating the function E_K on the plaintext, another function $E'_K = IDE * E_K * ODE^{-1}$ is computed, where IDE and ODE are input and output data encodings, which are randomly generated independent of K. Since in some cases it's infeasible to have input and output encodings of the desired bit length, some encodings can be represented as a concatenation of smaller bijections.

A bijection F of size $n = n_1 + n_2 + \cdots + n_k$ can be built from a list of smaller bijections F_i, where F_i has size n_i, and for any n-bit vector $b = (b_1, b_2, \dots, b_n)$,

$F(b) = F_1(b_1, .., b_{n_1})\, F_2(b_{n_1+1}, .., b_{n_1+n_2}) \dots F_k\left(b_{n_1+\cdots+n_{k_1+1}}, .., b_n\right)$.

In this case we say $F = F_1 \,||\, F_2 || \dots || F_k$, and call F a concatenated encoding.

The encryption algorithm is implemented as a sequence of look-ups from different tables. The output encoding of any table matches the input encoding of the table following it, so the encodings can cancel each other. After the encryption routine is over, the value of $E'_K = G * E_K * F^{-1}$ is properly computed, as any intermediate encodings are cancelled out. We will present an unprotected implementation first, to describe the tables we'll need, and then describe the modified protected version of the tables.

1.1.2.1 UNPROTECTED IMPLEMENTATION

For each round, AddRoundKey and SubBytes transformations can be made using 16 look-up tables that map 1 byte to 1 byte for each round. These tables are called T-Boxes, and are defined as follows:

$$T_i^r(x) = S\left(x \oplus \hat{k}_{r-1}[i]\right), \quad \text{for } i = \overline{0\dots15} \text{ and } r = \overline{1\dots9},$$

$$T_i^{10}(x) = S\left(x \oplus \hat{k}_9[i]\right) \oplus \hat{k}_{10}[i], \quad \text{for } i = \overline{0\dots15}.$$

It's obvious, that T-boxes have no security and the attacker can easily extract the keys from the T-boxes, if those were provided. So additional encoding is applied on the T-boxes, before they can be used.

After the state vector passes through the T-boxes, MixColumns operations must be applied on it. MixColumns operation multiplies each 4 bytes of the state vector $[x_1, x_2, x_3, x_4]$ with the $MC = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$ matrix. This can be accomplished by the so called T_{y_i} tables, where

$$T_{y_0}(x) = x \cdot [02\ 01\ 01\ 03]^T,$$
$$T_{y_1}(x) = x \cdot [03\ 02\ 01\ 01]^T,$$
$$T_{y_2}(x) = x \cdot [01\ 03\ 02\ 01]^T,$$
$$T_{y_4}(x) = x \cdot [01\ 01\ 03\ 02]^T.$$

So T_{y_i}-boxes are 1 byte to 4 byte tables. The outputs of each 4 consecutive T_{y_i} tables must be XOR-ed to get the output of the round, i.e. $T_{y_0}(x) \oplus T_{y_1}(x) \oplus T_{y_2}(x) \oplus T_{y_3}(x)$.

XOR tables are used for this purpose:

$$XOR(x, y) = x \oplus y.$$

XOR tables operate on 2 nibbles (4-bit values) each, so they are 1 byte to 4 bits mapping tables. The XOR of two 32-bit values can be computed using 8 copies of these XOR tables as shown in Figure 1.6.

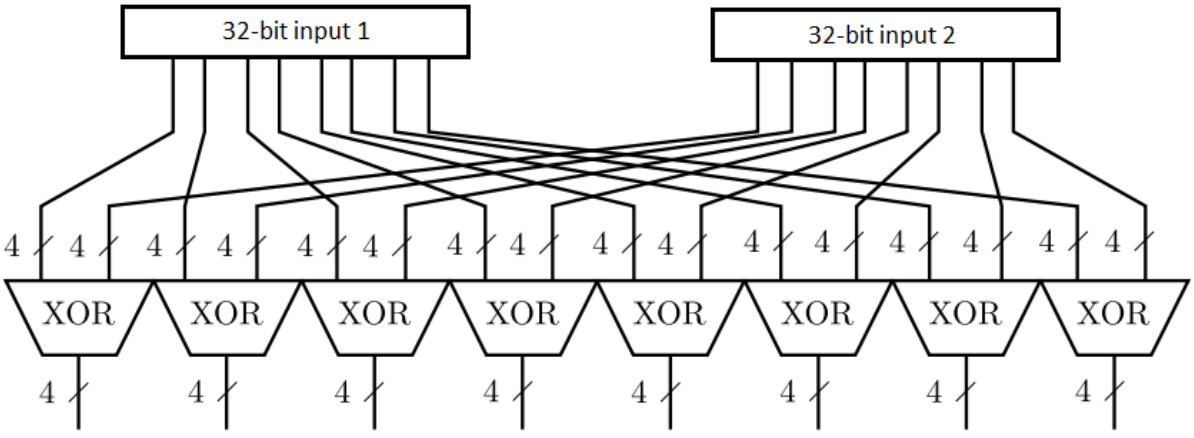

Figure 1.6: Applying XOR operation on 2 32-bit values using 8 4-bit XOR tables.

One can notice that T-boxes and T_{y_i} boxes can be combined into a single table. These tables will look as follows:

$$T_{y_0}\left(T_i^r(x)\right) = S\left(x + \hat{k}_{r-1}[i]\right) \cdot [02\ 01\ 01\ 03]^T, \quad \textbf{(1.1.2.1)}$$

$$T_{y_1}\left(T_i^r(x)\right) = S\left(x + \hat{k}_{r-1}[i]\right) \cdot [03\ 02\ 01\ 01]^T, \quad \textbf{(1.1.2.2)}$$

$$T_{y_2}\left(T_i^r(x)\right) = S\left(x + \hat{k}_{r-1}[i]\right) \cdot [01\ 03\ 02\ 01]^T, \quad \textbf{(1.1.2.3)}$$

$$T_{y_3}\left(T_i^r(x)\right) = S\left(x + \hat{k}_{r-1}[i]\right) \cdot [01\ 01\ 03\ 02]^T. \quad \textbf{(1.1.2.4)}$$

Then the outputs of these tables must be passed through XOR boxes to get the round's output.

So there will be 144 composed $\mathrm{T-box}/T_{y_i}$ tables, 864 XOR tables and 16 T-boxes total for all the rounds of AES together. These boxes are not secure against key extraction. The next section will change these tables to apply some defense mechanisms against different attacks.

1.1.2.2 PROTECTED IMPLEMENTATION

As there are just 256 possible values for $k^{r-1}[i]$, one can build a $\mathrm{T-box\ /\ T_{y_i}}$ table for each of these values, and check if the table we provide matches any of those. This will allow an attacker to extract the key from $T - box/Ty_i$ tables.

In order to prevent this, input and output encodings are applied to all the tables. The encodings are applied in a networked fashion, so all the encodings except the input encoding of the first round F and the output encoding of the last round G neutralize each other, and the function $E'_K = G \circ E_K \circ F^{-1}$ is calculated by using these tables, where $\circ$ stands for function composition. Here G and F are concatenated encodings of 128 bits, made of 16 bijections of 8 bits each. These encodings dramatically increase the total number of possible tables. There are $16!^2$ possible input encodings and $16!^8$ output encodings per table. It's obvious that for a fixed input key, the tables constructed for different output encodings are all different. This means that there are at least $16!^8$ possible tables, which makes it impossible to the attacker to enumerate.

As the input/output encodings provide the confusion step for the tables, linear transformations are applied to produce the diffusion step. The table input/outputs are multiplied with randomly generated matrices over $GF(2)$ which are called mixing bijections (MB).

16 8-bit to 8-bit MBs are randomly generated and applied at the inputs of each round except the first. Let's denote mixing bijection for byte i in round r as L_i^r. Another 4 32-bit to 32-bit mixing bijections MB_i^r, $\mathrm{i} = \overline{1,4}$, $\mathrm{r} = \overline{1,9}$ are applied to all the rounds outputs except the last one. So after these changes, the results that we get after applying XOR tables will need to be multiplied with the inverse of the mixing bijection $(MB_i^r)^{-1}$ and $(L_i^{r+1})^{-1}$,so the effect of MB is cancelled, and the inverse of L_i^{r+1} is applied so it can get cancelled in the next round. This is done with the same technique as the MixColumns step, and additional XOR tables are created for this.

4 types of tables are used in the WB, shown in Figures 1.7-1.10. Type I tables compute the strips in input permutation F and ODE , and represent a pair of 4-bit decodings, followed by an 128x8 matrix, then thirty-two 4-bit encodings. These tables are used in round 1 and 10 only, 32 tables are required, $2^8 * 128\ bits = 4096\ bytes$ each.

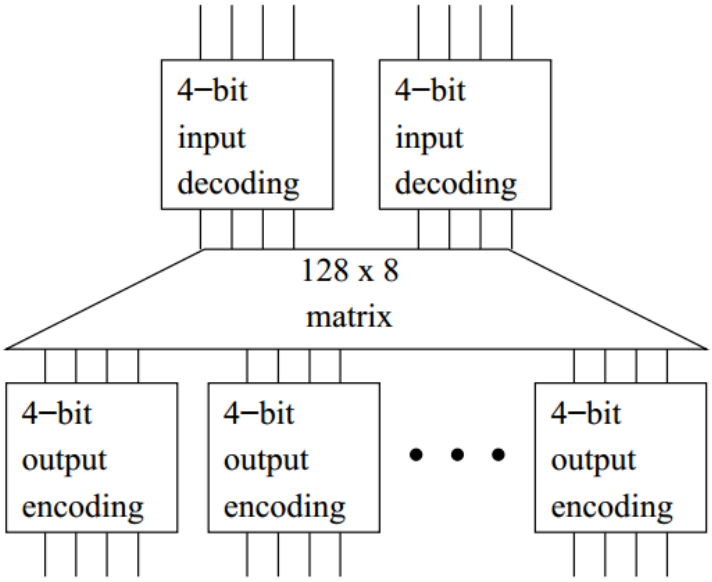

Figure 1.7: "**extern_encode**" table for input and output, type 1 [**15**].

Type II tables compute the strips in the first half of a round transformation. These represent two 4-bit decodings, then an 8x8 mixing bijection, followed by a T-box, then an 32x8 matrix, and finally eight 4-bit encodings. These tables are used in rounds 1-9, so 9 * 4 * 4 = 144 tables are required, $2^8 * 32\ bits = 1024\ bytes$ each.

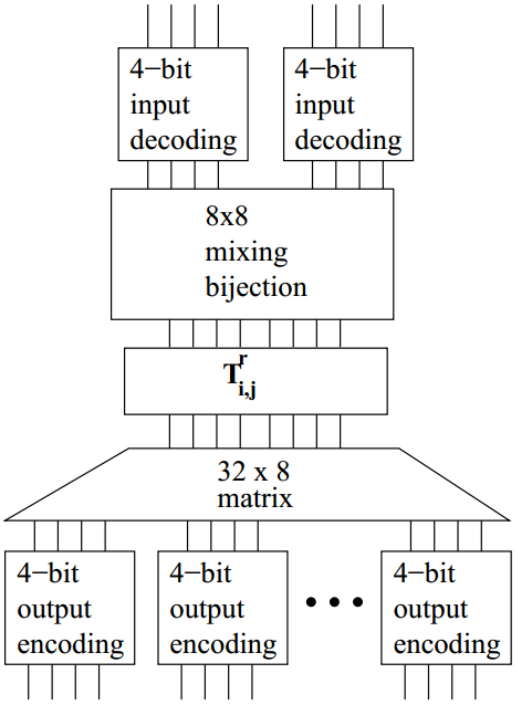

Figure 1.8: "**Sub**" table, type 2 [**15**].

Type III tables compute the strips in the second half of a round transformation, representing two 4-bit decodings, followed by a 32x8 matrix, and then eight 4-bit encodings. These tables are used in rounds 1-9, so 9 * 4 * 4 = 144 tables are required, $2^8 * 32\ bits = 1024\ bytes$ each.

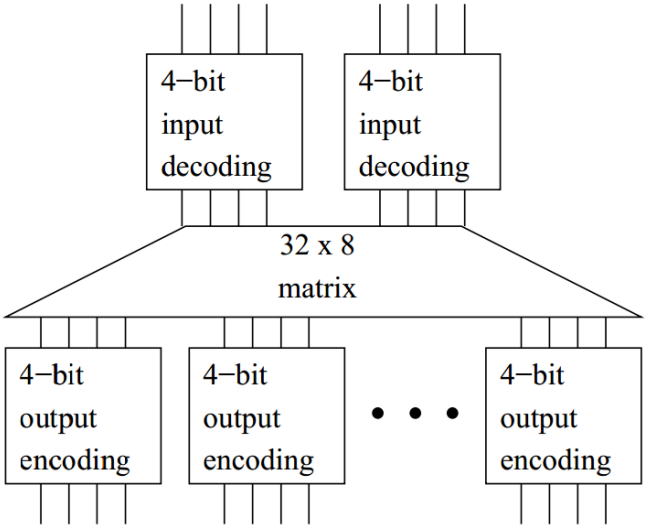

Figure 1.9: "**Untwist**" table, type 3 [**15**].

Type IV tables compute the $GF(2)$ additions, representing two 4-bit decodings, the known XOR operation, and a 4-bit output encodings. These tables are used in groups for applying XOR operation on values of higher bit-length. In each of rounds 1-9 twelve 32-bit XORs are

required and as 8 XOR tables are required to compute a single 32-bit XOR, a total of 9 * 2 * 4 * 8 * 3 = 1728 type 4 tables are required adding up the outputs of type 2 and 3 boxes, and another 2 * 32 * 15 = 960 tables are required for adding up the outputs of type 1 boxes. So 2688 type 4 boxes are required in total, $2^8 * 4\ bits = 128\ bytes$ each.

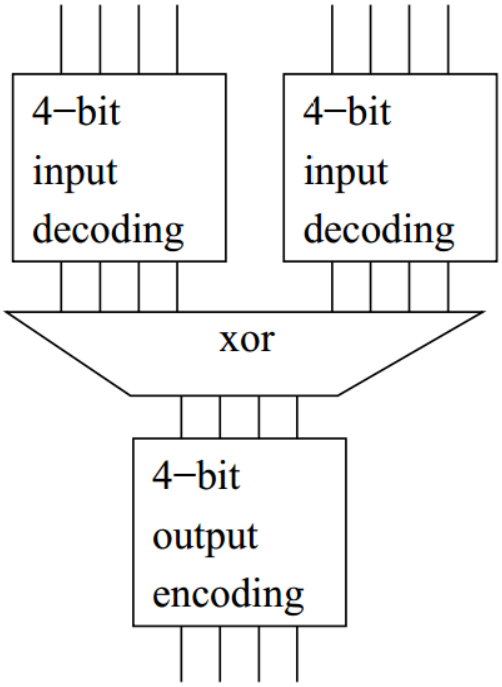

Figure 1.10: "**XOR**" table, type 4 [**15**].

The structure of a complete AES round 2 for first 4 bytes of state of Chow's WB is shown in Figure 1.11. AES 16-byte state is divided into four 4-byte parts, and the encryption is procedure is executed separately on them. Similar boxes are applied on the other bytes of the 16-byte state. Rounds 3-9 are similar. For the first round, the input mixing bijections (MBs) are skipped and type 1 tables are used.

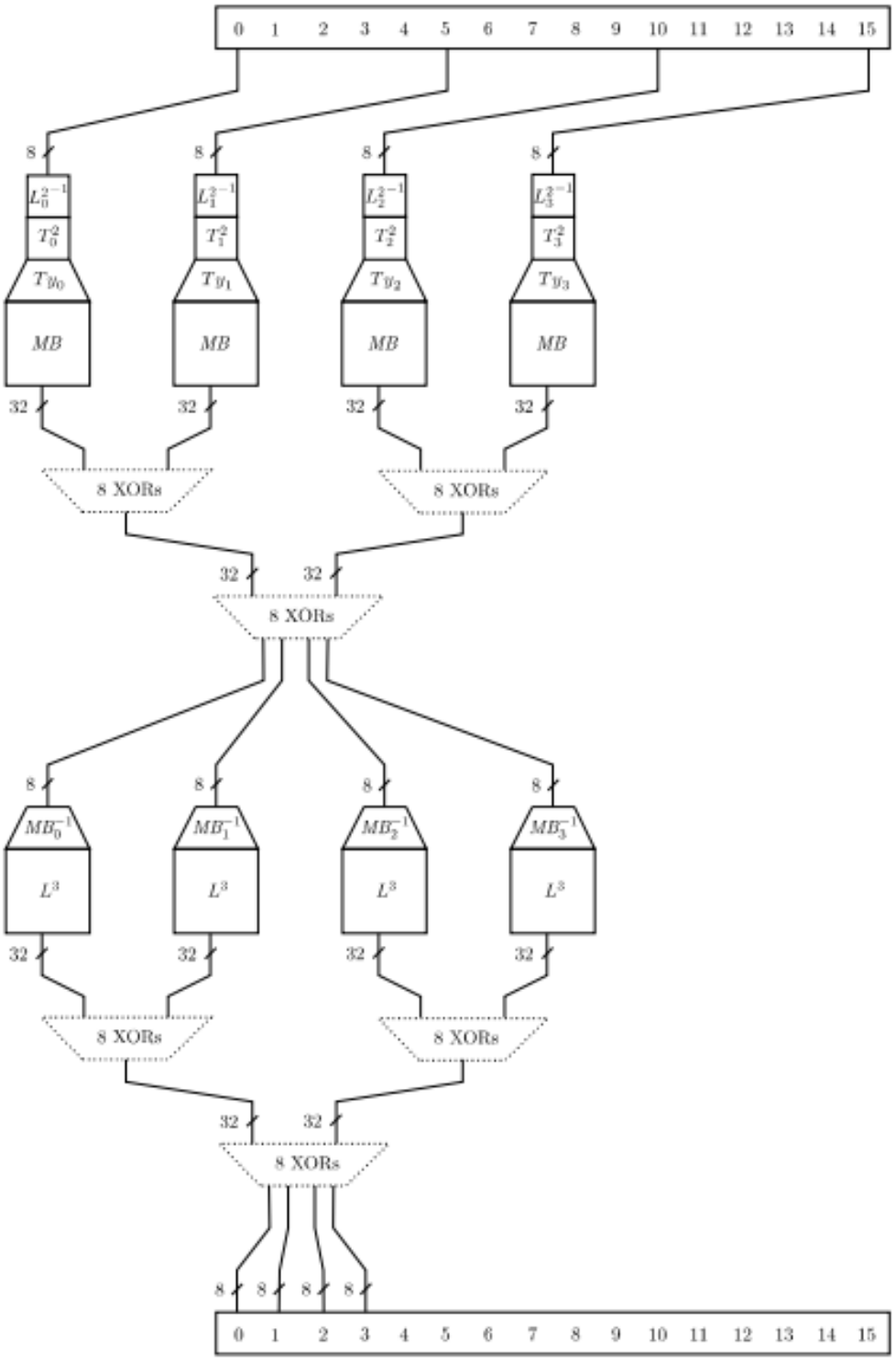

Figure 1.11: Structure of a complete AES WB round 2. Rounds 3-9 are similar. For round 1 the only difference is the absence of input mixing bijections on the T-boxes [**49**].

As we see in Figure 1.11 the encryption on the first 4 bytes is done with the following steps:

- All 4 bytes pass through the $T - boxes$ and MBs using Type 2 tables.

- The four 32-bit outputs get XOR-ed using 24 type 4 tables.
- The inverse MBs are applied using type 3 tables.
- The four 32-bit outputs get XOR-ed using 24 type 4 tables.

The total size of the lookup tables of this implementation is $32 * 4096 + 144 * 1024 + 144 * 1024 + 2688 * 128 = 770{,}048$ bytes. There are 3104 lookups during each execution [**15**]. For a more detailed description of Chow's WB AES refer to [**15**] and [**49**].

1.1.3 MIXING BIJECTIONS ARE NECESSARY FOR CHOW'S IMPLEMENTATION

In this chapter we briefly describe an attack on Chow's WB tables which could possibly work if MBs were not used, and the tables were protected only with input/output encodings. It is shown in [**15**] how to extract input/output encodings and secure key bytes having $T - box/Ty_i$ tables of round 2 defined by formulas **1.1.2.1-1.1.2.4** using frequency analyses. So we have 3 different values to attack:

$$01 \cdot S(x + \hat{k}_{r-1}[i]), 02 \cdot S(x + \hat{k}_{r-1}[i]), 03 \cdot S(x + \hat{k}_{r-1}[i]).$$

Each value is given by two 4-bit nibble pair, with an output permutation applied on it. The first 4 rows of AES S-box are provided in the table 1.1.3.1.

Table 1.1.3.1: The first 4 rows of AES S-box [**49**].

63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75

The number of times each left nibble value occurs in the first row is shown in the following table.

Table 1.1.3.2: The frequencies of each value as a left nibble in the first row of S-box [**49**].

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	0	1	1	0	0	4	4	0	0	1	0	1	1	0	2

By sorting the entries in the table 1.1.3.2 in descending order we get 4421111110000000.
So if we take the outputs of $01 \cdot S(x + \hat{k}_{r-1}[i])$ and apply this operation of both left and right nibble of its outputs, the value we'll get is called frequency signature [**15**].
One can notice, that frequency signatures do not depend on the 4-bit input-output encodings used, so no matter which encoding you use, the resulting signatures are the same, and depend only on the key bytes. Similar to the row signatures column signatures can be built. So two sets R and C are built, which contain 16 row and column signatures for each byte. It can be programmatically checked, that the values of R and C are distinct for each key. Using the values of R and C for the given $T - box/Ty_i$ tables Chow et al. presented an attack for key extraction [**15**], [**49**]. After the application of MBs these frequency signatures are dependent not only on the key, but also on the chosen random bijection, which makes this attack impossible.

1.1.4 BGE ATTACK

BGE attack, introduced in 2004 [**16**], targets not a single table from Chow's implementation, but a group of tables, which form the AES round. As each table has good confusion and diffusion properties, it's hard to extract the key from a single table, but attacking a group of tables is more reasonable. The AES round is viewed as four 32-bit to 32-bit mappings R_i^r, where the structure of R_i^r is shown in Figure 1.6. P_i^rs are a combination of input encodings and mixing bijections(MBs), Q_i^rs are a combination of MBs and output encodings. As MBs are already canceled out after these 2 steps are performed, they have no effect any more.
The BGE attack consists of 3 phases:

- The values of P_i^rs and Q_i^rs are recovered up to an unknown affine transformation.

This allows us to replace transformations P_i^r and Q_i^r with affine transformations.

- The values of P_i^r and Q_i^r are recovered completely.
- Having P_i^r and Q_i^r, the AES-128 private key is extracted from the WB tables.

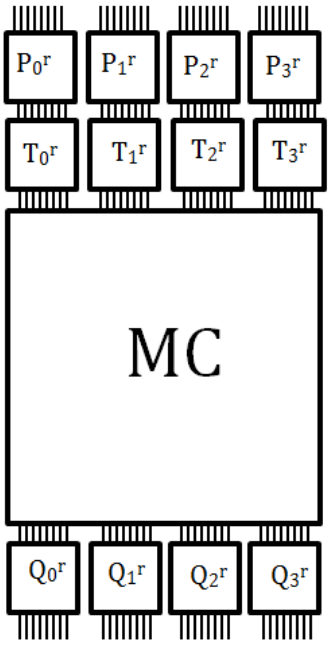

Figure 1.12: R_0^r mapping, 1 of the 4 mappings that make the AES round.

Let's denote the inputs of R_0^r as x_0, x_1, x_2, x_3 and the outputs as y_0, y_1, y_2, y_3, where

$$y_0 = Q_0(02 \cdot T_0^r(P_0^r(x_0)) \oplus 03 \cdot T_1^r(P_1^r(x_1)) \oplus 01 \cdot T_2^r(P_2^r(x_2)) \oplus 01 \cdot T_3^r(P_3^r(x_3))),$$
$$y_1 = Q_1(01 \cdot T_0^r(P_0^r(x_0)) \oplus 02 \cdot T_1^r(P_1^r(x_1)) \oplus 03 \cdot T_2^r(P_2^r(x_2)) \oplus 01 \cdot T_3^r(P_3^r(x_3))),$$
$$y_2 = Q_2(01 \cdot T_0^r(P_0^r(x_0)) \oplus 01 \cdot T_1^r(P_1^r(x_1)) \oplus 02 \cdot T_2^r(P_2^r(x_2)) \oplus 03 \cdot T_3^r(P_3^r(x_3))),$$
$$y_3 = Q_3(03 \cdot T_0^r(P_0^r(x_0)) \oplus 01 \cdot T_1^r(P_1^r(x_1)) \oplus 01 \cdot T_2^r(P_2^r(x_2)) \oplus 02 \cdot T_3^r(P_3^r(x_3)));$$

Now, having these tables, we must find a transformation $\tilde{Q}_i^r$, such that

$$\tilde{Q}_i^r = Q_i^r \circ A_i^r,$$

i.e. $\tilde{Q}_i^r$ differs from Q_i^r by an unknown affine transformation A_i^r.

So y_0 is a function of 4 parameters x_0, x_1, x_2, x_3. Let's fix the values of x_1 and x_2 to c_1 and c_2, respectively, and give 2 different values to x_3, namely c_3 and c_3'. We will get 2 functions:

$$y_0(x_0, c_1, c_2, c_3) = Q_0\left(02 * T_r(P_r(x_0))\right) + B_{c_1,c_2,c_3},$$
$$y_0(x_0, c_1, c_2, c_3) = Q_0\left(02 * T_r(P_r(x_0))\right) + B_{c_1,c_2,c_3'};$$

From these 2 equations we get:

$$y_0(x_0, c_1, c_2, c_3) \circ y_0(x_0, c_1, c_2, c_3)^{-1} = Q_0 \circ \oplus_B \circ {Q_0}^{-1},$$
$$\text{where } B = B_{c_1,c_2,c_3} + B_{c_1,c_2,c_3'}.$$

If we vary the value of c_3' so it can take all the possible 256 values in $GF(2^8)$, the value of B will also take all the 256 different values. This gives us a set of 256 functions $S =$

$\{Q_0 \circ \oplus_B \circ Q_0^{-1}\}_{B \epsilon GF(2^8)}$, where Q_0 is a permutation of $GF(2^8)$. This set of bijections forms a commutative group under composition. Billet et al provide a technique to recover the value of Q_0 up to an unknown affine transformation, given this group of functions S. Once this is done, we can change all the tables to use A_i^r instead of Q_i^r, which significantly weakens the confusion properties of the tables. As the output encodings Q match the input encodings of the next round P then the values of P can also be changed to affine.

Now we will shortly review the algorithm of recovering the input and output encodings completely, given they are already changed to be affine. For any round we can write a set of equations, describing the mappings R_i^r as

$$y_0 = Q_0\big(02 \cdot T_0'(x_0) \oplus 03 \cdot T_1'(x_1) \oplus 01 \cdot T_2'(x_2) \oplus 01 \cdot T_3'(x_3)\big),$$
$$y_1 = Q_1\big(01 \cdot T_0'(x_0) \oplus 02 \cdot T_1'(x_1) \oplus 03 \cdot T_2'(x_2) \oplus 01 \cdot T_3'(x_3)\big),$$
$$y_2 = Q_2\big(01 \cdot T_0'(x_0) \oplus 01 \cdot T_1'(x_1) \oplus 02 \cdot T_2'(x_2) \oplus 03 \cdot T_3'(x_3)\big),$$
$$y_3 = Q_3\big(03 \cdot T_0'(x_0) \oplus 01 \cdot T_1'(x_1) \oplus 01 \cdot T_2'(x_2) \oplus 02 \cdot T_3'(x_3)\big),$$

where $T_i' = T_i \circ P_i$, and the round number r was dropped, since the equations are the same for all the rounds.

The algorithm is provided in [**16**], by which for each pair (y_i, y_j) we can find a linear mapping L and a constant c such that

$$y_i(x_0, 0, 0, 0) = L\Big(y_j(x_0, 0, 0, 0)\Big) \oplus c.$$

The algorithm requires less than 2^{16} work steps to compute the values of L and c, i.e. the mapping between $y_i \; and \; y_j$can be found. This means, that having the affine parasite A_i^r for one value of i we can compute the values of the other 3 affine parasites A_j^r, spending less than 2^{16} work steps on each. So now all we need to do is find the affine transformation A_i^r for just one value of i.

Having affine transformations as input and output encodings and the algorithm to find the values of L and c for each pair of (y_i, y_j) the values of Q_i and P_i are determined, which is described in detail in [**16**]. Also the fact, that $T_i(x) = S(x \oplus k_i)$, i.e. the mapping $x \mapsto S^{-1} \circ T_i \circ P_i(x) = P_i(x) \oplus k_i$ is affine, is used for recovering the affine parasites up to the key addition. Then having these values and using the simple structure of AES key schedule, the secure key is recovered.

1.1.5 PERTURBED AES WHITE-BOX

In 2006, perturbed AES WB implementation was presented [**43**]. This WB scheme relies on the Isomorphism of Polynomials (IP) problem [**50**]. In order to tightly link the rounds together and prevent any attack targeted at a single round, specific terms are added to the first round, which are carried over up to the last round, where they are cancelled out. An AES WB implementation is presented, where the S-boxes are secret and unknown to the attacker – called Advanced Encryption without standard S-boxes (AEw/oS).

1.1.5.1 BRINGER ET AL.'S PERTURBATIONS

Let $\tilde{0}$ be a polynomial which often equals to 0, and denote $\tilde{P} = P + \tilde{0}$ for any polynomial P. The idea of [**51**] is to replace the original systems of polynomials $G_{i,j}$ by $\tilde{G}_{i,j}$ and insert random linear transformations between each pair of rounds to hide this modification [**43**]. The algorithm of building several correlated polynomials $\tilde{0}$ is provided in [**51**]. This allows creation of a set of white-boxes with modified S-boxes in each of them. As $\tilde{0}$ equals 0 often enough, the actual result can be obtained by majority vote, if enough white-boxes are present. This modifies the set of equations the attacker must analyze to break the cipher.

1.1.5.2 MODIFIED AES WHITE-BOX

The operations in a single round of AES can be defined as systems of n = 16 equations in $GF(2^8)$ with 4 variables each. The last round gives a system 16 equations, each having around 255 terms of a degree less than 254. Four correlated polynomials $\tilde{0}$ are selected in a special way, such that for any input, 2 polynomials are equal to zero while the other 2 give different values. This way having 4 white-boxes the real result will be distinguishable by the majority vote.

Several modifications were made to the initial AES WB:

- Random variables are added in every round except the last one to dissimulate the information.

- Specific terms are added to the first round, that are carried round after round, up to the last one, where they are canceled out.
- Corresponding polynomials are added to the last round, such that they vanish when the correct value is reached. In this way, intermediate results of each round are 'false' with a high probability as the specific terms added in the first round perturb all the intermediate values until they are cancelled in the last round [**43**].

The size of the described WB is 142 MB, and as the authors state that the presented system has to be improved to be practical for execution on a general purpose processor. Huge size of the WB tables and slow performance make this implementation practically infeasible.

1.1.6 XIAO-LAI WHITE-BOX AES ALGORITHM

In 2009 another attempt was made to build a secure AES WB by Yaying Xiao and Xuejia Lai [**17**]. The main idea is to add a mixing bojection MB before MixColumn matrix and cancel it in the next round. For that, let's define a mixing bijection MI as

$$MI^r = T^r \circ MC \circ (MB^{r-1})^{-1} \circ MC^{-1} \circ (T^r)^{-1} \text{ , for } r = 2,..,9.$$

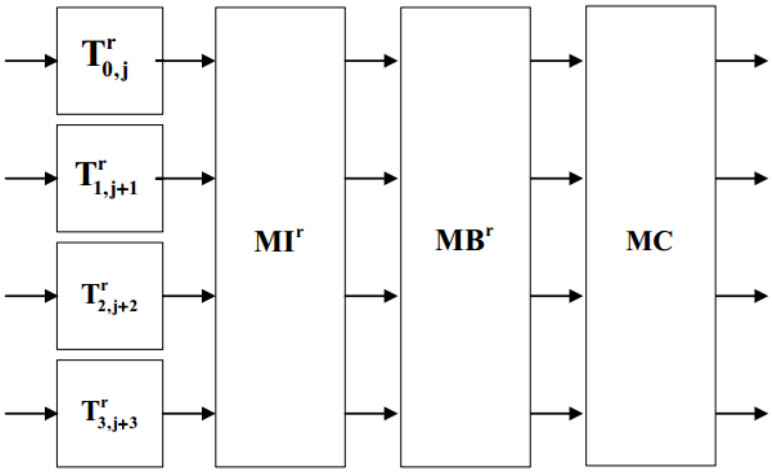

Figure 1.13: AES WB round using MI.

A single AES round for 4 bytes is shows in Figure 1.13. In order to improve diffusion properties of the table, additional 8x8 mixing bijections are applied on each byte of the output and canceled out in the next round, so the final structure of the round is depicted in

Figure 1.14. In order to encrypt a message using structure defined in Figure 1.14, modifications to Type 2 tables of Chow's implementation are required.

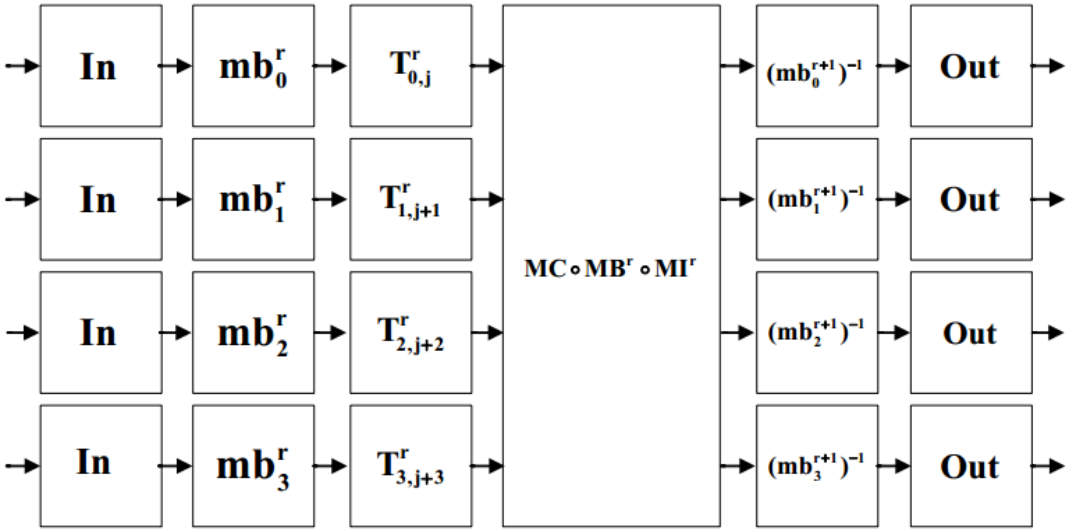

Figure 1.14: AES WB round structure of Xiao-Lai implementation.

Also several modifications were made to type 1 tables. Input and output encoding F and G are used, where is composed of a non-linear input encodings and a linear bijection $U.G$ is composed of a linear bijection V and non-linear output encodings. Type 1 (a) tables, shows in Figure 1.15, are applied on the inputs of round 1 to apply the input encoding F and the inverse of MB^1, so the MB applied in round 2 will be canceled out. V is integrated into the last round, so table type 1(b), shown in Figure 1.16, is used [**17**]. The total size of the WB tables is 500 KB.

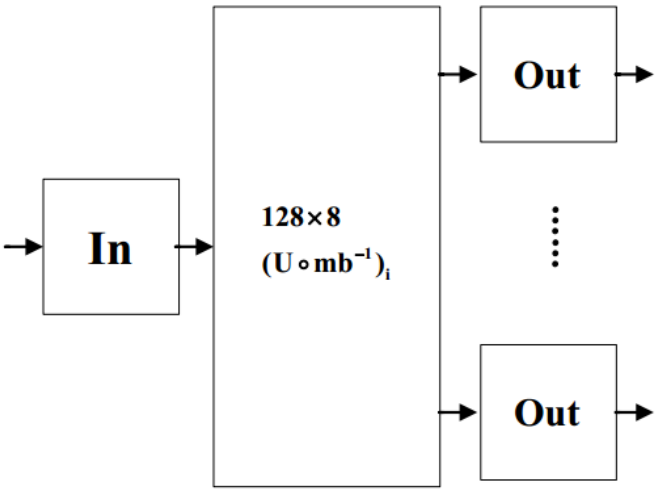

Figure 1.15: type 1(a) table.

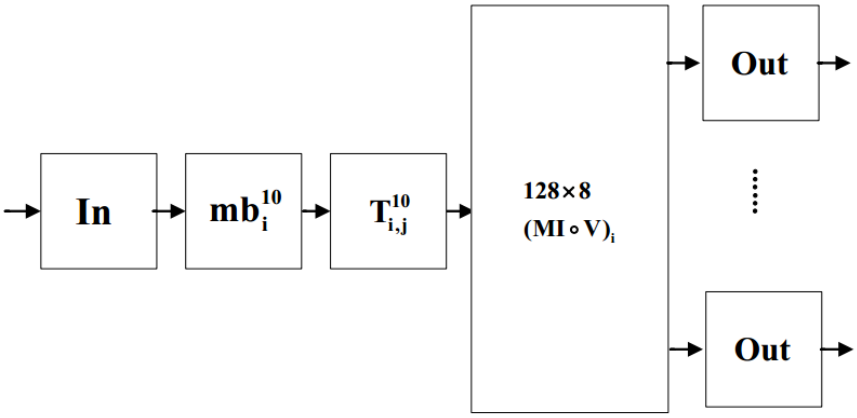

Figure 1.16: type 1(b) table.

Authors state that this implementation will resist the BGE attack, as the mixing bijection MB is never completely canceled out. The BGE attack attacks a group of Chow's WB tables, such that MB is already cancelled out and has no effect any more. In the described implementation this is not possible any more, as MB is removed only in the next round.

1.1.7 KARROUMI'S AES WHITE-BOX IMPLEMENTATION

In 2010 Mohamed Karroumi presented a modification of Chow's AES WB algorithm, which was supposed to withstand the BGE attack [**19**]. The algorithm is based on usage of AES dual ciphers. AES dual ciphers were first presented in [**20**] with a list of 240 ciphers. This list was further expanded to 61,200 ciphers that are dual to AES in [**21**]. For each of these dual ciphers, there exists an affine transformation *Δ that maps AES plaintext P, ciphertext C*

and key K into plaintext P_{dual}*, ciphertext* C_{dual} *and key* K_{dual} *of a dual AES, i.e.* $P_{dual} = \Delta(P)$, $C_{dual} = \Delta(C)$ and $K_{dual} = \Delta(K)$.

In Karroumi's WB implementation, for each of 10 rounds of AES a dual-AES is selected randomly. $SubBytes$ constants, $MixColumns$ matrix and the key of the corresponding dual-AES cipher are used for building the $T - box/T_{y_i}$ tables. In order to get the same output values for the same input values as Chow's AES algorithm, affine transformation Δr must be applied at the input of each round, and Δr^{-1} at the output of the round. The outputs of each round of this implementation will be different from Chow's outputs as different $SubBytes$ and $MixColumns$ values are used, but the final round outputs will match.

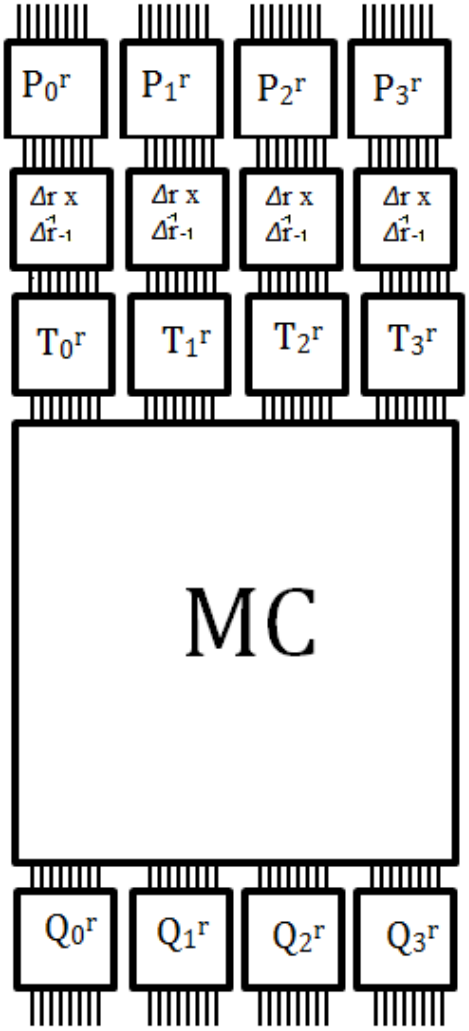

Figure 1.17: Dual AES round structure.

One can notice, that as each 4 output bytes of an AES round depend on only 4 input bytes, 4 different dual ciphers may be used in each round, so employing 40 different randomly chosen dual ciphers in total. These changes will not affect the general structure of the tables, so the speed and memory requirement will be identical to Chow's implementation. Karroumi argues that the attacker will need to brute force these randomly chosen dual ciphers, which will increase the security to 2^{91}. The detailed description of this implementation can be found in [**19**].

1.1.8 ATTACK OF LEPOINT AND RIVAIN FOR BOTH CHOW'S AND KARROUMI'S IMPLEMENTATIONS

In 2013, another successful attack was presented by Lepoint and Rivain [**23**]. The attack requires 2^{22} work steps, is simpler to implement than the BGE attack and can be applied on both Chow's and Karroumi's white-boxes. For this attack, the AES round is viewed as application of function f on 4 bytes of the state, where f is defined as

$$f : (x_0, x_5, x_{10}, x_{15}) \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \oplus \begin{pmatrix} S(x_0 \oplus k_0) \\ S(x_5 \oplus k_5) \\ S(x_{10} \oplus k_{10}) \\ S(x_{15} \oplus k_{15}) \end{pmatrix} \quad (\mathbf{1.1.7.1}).$$

Let's denote by $E_i = \left(P_{i,0} \middle|\middle| P_{i,1}\right) \circ L_i$ and $E_i' = \left(P_{i,0}' \middle|\middle| P_{i,1}'\right) \circ L_i'$, where $P_{i,j}$ are input encodings and L_i are mixing bijections. Chow's WB algorithm can then be expressed by a function f', where

$$f' = (E_0' || E_1' || E_2' || E_3') \circ f \circ (E_0^{-1} \,||\, E_5^{-1} \,||\, E_{10}^{-1} \,||\, E_{15}^{-1}) \text{ [23].}$$

Function f' can be computed using Chow's WB tables, by putting together all the tables that build a single AES round. Let's denote by f_i' the coordinate functions of f', such that

$$f' = (f_0', f_1', f_2', f_3').$$

Then let us denote by S_j the function

$$S_j(x) = S\left(k_j \oplus E_j^{-1}(x)\right).$$

Then the attack recovers the functions S_j by looking at the collisions of the function f'. Particularly, for recovering S_0 and S_5, we look at the collisions of form

$$f_0'(\alpha, 0,0,0) = f_0'(0, \beta, 0,0),$$

which can be written as

$$\mathrm{E}_0'(02 \cdot S_0(\alpha) \oplus 03 \cdot S_5(0) \oplus S_{10}(0) \oplus S_{15}(0)) = \mathrm{E}_0'(02 \cdot S_0(0) \oplus 03 \cdot S_5(\beta) \oplus S_{10}(0) \oplus S_{15}(0)),$$

from which we get

$$02 \cdot S_0(\alpha) \oplus 03 \cdot S_5(0) = 02 \cdot S_0(0) \oplus 03 \cdot S_5(\beta).$$

There are always enough collisions in the outputs of function f_0' to construct enough such equations and solve them for the values of S_0 and S_5. Actually functions $\alpha \mapsto f_0'(\alpha, 0,0,0)$ and $\beta \mapsto f_0'(0, \beta, 0,0)$ are bijections, so there will be exactly 256 collisions for all 256 possible values of α and β, giving us 256 linear equations. If we look at the other collisions of form $f_i'(\alpha, 0,0,0) = f_i'(0, \beta, 0,0)$, we will get 256 linear equations from each of $i = 0..3$, so 4*256 linear equations will be built. Some of these will be irrelevant or repetitive, but this system of 4*256 linear equations with just 2*256 unknowns $S_0(\alpha)$ and $S_5(\alpha)$ can always be solved.

In a similar way the rest of the values of S_i can be recovered.

After recovering the values of S_i, the next step is the key extraction. It's easy to notice that

$$f_0'(\alpha, 0,0,0) = E_0'\left(\left(02 \cdot S_0(\alpha) \oplus 03 \cdot S_5(0) \oplus S_{10}(0) \oplus S_{15}(0)\right)\right),$$

which means having the values of f_0' and S_i, we can easily compute the value of function E_0'. Other values of E_i' are recovered similarly.

Having the values of E_i', the key can be extracted by checking all 256 possible values for the key byte. It's easy to check that function

$$\varphi'(x) = \sum_{\alpha=0}^{15} f_0'\left(E_0\left(S^{-1}(x) \oplus \widehat{k_0}\right), 0,0,0\right)$$

equals to 0 when the value of $\widehat{k_0}$ equals to the value of the secret key, used in the construction of the white-boxes [**23**]. So one can easily run over the 256 possible values of $\widehat{k_0}$ and check which one is the real key byte. Then the same is done for the other key bytes. The bottleneck of this attack is the recovery of the values S_0, S_5, S_{10}, S_{15}, which requires $\sim 2^{22}$ operations.

1.2 APPLICATIONS OF WHITE-BOX CRYPTOGRAPHY

Initially WB cryptography schemes were created to be used in DRM systems and for software protection of cryptographic keys for the cases, when the software runs in unprotected environment. Later WB algorithms found applications in Oblivious Transfer (OT) protocols, Secure Pattern Search, Secure Multi-party computations and SE algorithms, as more computational-efficient replacement of traditional PK cryptography. In this section, we describe these applications of WB cryptography in more details.

1.2.1 DRM AND SOFTWARE PROTECTION

Cryptography is increasingly deployed in applications which run on unprotected devices (such as PCs, smartphones or tablets). On this kind of devices, gets full access to all the intermediate values generated during the software execution and can change the execution at will. This means that an attacker can use different emulators, disassemblers and debuggers such as IDA Pro system and others to analyze binary code of the software, see values of variables during execution and change the execution routine. Any cryptographic software running on these kind of devices is operating in the WBAC. This makes WB algorithms an essential part of any encryption routine running on such unprotected devices, as usage of conventional black-box algorithms will allow the attacker to directly read the cryptographic key from device's memory during the execution.

WB cryptography is a crucial part of several security-centric commercial products, such as Digital Rights Management (later DRM) applications. It is used to effectively protect cryptographic keys, which has often been the Achilles heel in DRM systems in the past. Other benefits of using a WB algorithm are software traceability, tamper resistance, renewability, and so on. This makes WB cryptography an essential part of any software security strategy for cryptographic software running on unprotected devices. This technique enables digital media distribution from Amazon Prime Video, iTunes, and others, which actively use software DRM protection. The initial versions of iTunes used black-box cryptography and were famous for being broken by Jon Lech Johansen (DVD John) in a matter of days. In 2005, Apple switched to using WB encryption algorithms, which resulted to the increased level of difficulty in breaking "FairPlay", breaches to the DRM were

stopped, allowing the company to sell movies on iTunes. WB algorithms decrypt movie on the fly, allowing the user to watch it, but without letting the attacker observe any portion of the secure key. This is achieved by making the encryption algorithm run using specially built look-up tables. The attacker must be unable to derive the secret master key based on the tables, yet the encryption must be performed correctly.

1.2.2 EFFICIENT OBLIVIOUS TRANSFER PROTOCOLS

"1-out-of-2" Oblivious transfer (OT) protocol can be described in the following way. Alice (the Server) possesses two data items in a database: k1 and k2. Bob (the Receiver) secretly requests a single item from Alice. OT is a protocol which describes the communication between Bob and Alice, such that Bob requests Alice to send him one of the items, yet in such way that Alice does not learn which item was sent, and Bob does not learn the value of the other item. There are also many generalizations of the basic OT protocol and one of the most important generalization is 1 out of n OT. 1-out-of-n OT means Alice (the server) owns n secrets, namely $s_1, s_2, .., s_n$, and Bob(the client) wants to get the values of s_i. Alice must return the value of s_i to Bob, without learning the value of i, and Bob must get the value of s_i without learning any other information about the secrets. OT protocol plays a key role in secure multiparty computation protocols.

Most existing basic and generalized OT protocols are based on slow PK operations, which makes them infeasible to be used in most of their applications. In the recent result [**13**] a novel approach for designing OT protocols is introduced, which replaces PK operations with WB cryptography techniques. This results to significant performance improvement and reduction of necessary communication bandwidth for OT protocol. This makes it practical to use OT protocols as a part of secure function evaluation (SFE).

1.2.3 SECURE PATTERN SEARCH ALGORITHMS

Secure pattern matching (SPM) algorithms considers the following problem. The first party (client) has a Deterministic Finite Automata (DFA) (or regular expression) Γ as long as the second party (server) has a string X. Their objective is to check whether the string X

belongs to the language generated by the Γ allowing both parties to learn the answer so that neither the client nor the server learned any additional information about the input of each other.

This problem and its variations have numerous application: lots of problems which are associated with two-party secure and private communication result to this problem. One such example is searching the number of appearances of some given DNA pattern in the server's DNA storage, without client revealing to the server the pattern searched, and without the server providing the DNA of any person to the client. Another popular application is checking the client's credit history by the bank, to see if the client is trustworthy, without getting the full credit history of the client. Secure pattern matching problem and its applications have been actively discussed during recent years [**52**], [**53**], [**54**], [**55**], [**56**], [**57**], [**58**], [**59**].

SPMs are compared to each other based on the the number of client-server communication rounds, the required bandwidth and computational resources. The recent result [**60**] presents an SPM algorithm similar to the "Efficient Protocol for Oblivious DFA Evaluation" construction described in [**52**], but unlike it, it does not use any PK operations in the algorithm and is totally based on WB based 1-out-of-2 OT protocol [**32**], which results to significant performance improvements for the secure search pattern algorithm.

1.2.4 DELEGATABLE SEARCHABLE ENCRYPTION SCHEMES

Nowadays online storage providers like Dropbox [**2**], Google Drive [**3**], Box [**4**] and Microsoft OneDrive [**5**] are widely used. These cloud storages provide data storage and sharing functionality to their users. However, the third-party storages often are not trusted, which requires users to outsource their data in encrypted form. Users of such systems store their files in the cloud in unencrypted form, so the provider has access to each and every file of the user. This gave birth to cloud encryption gateways like Sookasa [**9**], nCryptedCloud [**10**], Boxcryptor [**11**] and SkyCryptor [**12**], startups that provide encryption services to the users who want to keep their files secure, while still storing them in any of the clouds. Data encryption preserves the data security but on the other hand prevents the

user to search and selectively access the parts of the data. Searchable encryption (SE) is a cryptographic technique which allows clients store their documents on a server in encrypted form without sacrificing the user experience of accessing the data. In recent years many practical schemes have been developed for performing SE with optimal search time and efficient memory consumption with reasonable security leakage. The most practical schemes are based on the secure index approach.

In an index-based SSE scheme [**61**], [**62**], [**63**], [**64**], [**65**], [**66**], [**67**], [**68**], [**69**], [**70**] the encryption algorithm takes as input an index and a sequence of n files f = (f_1,…. , f_n) and outputs an encrypted index and a sequence of n ciphertexts c = (c_1,…, c_n). All known index constructions encrypt the files f using a symmetric encryption scheme such as AES. To search for a keyword w, the client generates a search token t_w and given t_w and c, the server can find the identifiers I_w of the files that contain w.

Most of the schemes consider only the SE domain where the owner of the data performs both storage and retrieval of data, i.e. there is no file sharing between the users and each user has access to only his own files. The natural extension of symmetric SE was the multi-user setting, where each user has an encrypted storage for his documents, but many entities can be granted access to search among his files.

In [**32**] a development of a novel method of constructing delegable (multi-user) SE schemes is presented which can transform any existing symmetric SE schemes into multi-party SE scheme. This allows the third-parties search on the user's encrypted database as well as update the data by adding (and/or deleting) new documents without compromising the user's key security or violating the practical aspects of scheme usage. Access to the files shared between users is set up by each user and enforced by the server, where the server is considered to be honest-but-curious. The third-party added files can only be decrypted using the secure master key. The underlying idea of the proposed approach is to employ WB cryptography techniques to delegate the encryption functionality to third-parties without revealing the secret key. The proposed implementation is based on the scheme called "Dynamic searchable symmetric encryption" [**62**], where the data owner can grant permission to third-parties for searching on his/her encrypted database, as well as update

the database without violating the master key security or adding extra communication and/or computation efforts for each search.

CHAPTER 2

WHITE-BOX ENCRYPTION ALGORITHM BASED ON SAFER+ BLOCK CIPHER

In this chapter we introduce a novel WB encryption algorithm based on SAFER+ block cipher. The black-box algorithm is briefly described. The original key schedule was modified to make it irreversible, i.e. if the attacker somehow gets the key for some round, he/she must be unable to derive the keys for the other rounds.

The encryption/decryption and table generation algorithms are presented for the proposed WB algorithm. Security analyses are discussed and resistance to BGE attack [**16**] is shown.

A C++ library was implemented to demonstrate the cipher. Implementation aspects are discussed and performance test results are provided.

2.1 SAFER+ BLACK BOX ENCRYPTION AND DECRYPTION

The underlying symmetric encryption algorithm for the WB encryption is based on Safer+ block cipher [**26**] with 128-bit key running 6 rounds plus and extra key addition at the end of 6-th round. The cipher was nominated as a candidate for Advanced Encryption Standard(AES). Though the official document [**26**] requires 8 rounds of encryption, differential analyses show, that 7 rounds of encryption provide enough security. So in this paper we are using 128-bit Safer+ with 6 full rounds encryption and an extra key addition at the end. Similar WB can be built for 192 and 256-bit settings.

The round encryption procedure is graphically shown in the Figure 2.1. We can see, that a single round of Safer+ consists of non-liner layer built with key addition and logarithmic and exponent functions, followed by a linear layer which consists of 4 layers of $2-PHT$ boxes followed by Armenian shuffles.

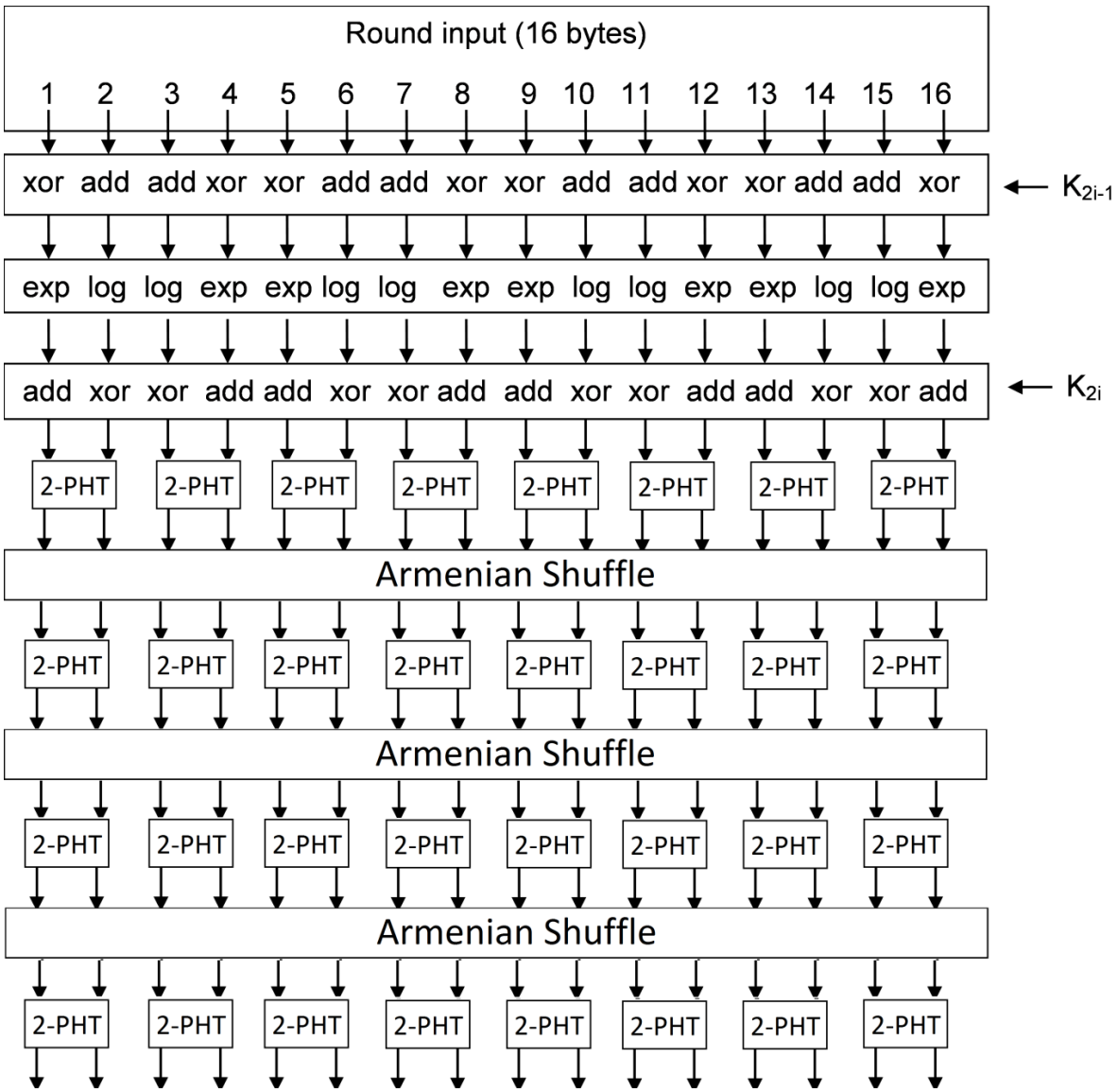

Figure 2.1: Round structure of SAFER+

Below is the meaning of the functions used in the SAFER+ round.

- **exp** stands for the exponential function, which converts the input value x into $45^{x} \bmod 257$.
- **log** stands for the logarithmic function, which converts the input values x into $\log_{45}(x) \bmod 257$.
- **xor** stands for bitwise exclusive OR of two input values.
- **add** stands for the sum of the two input value modulo 256.

$2-PHT$ stands for Pseudo-Hadamard Transform operation, i.e. it multiplies 2 input values with matrix $\mathrm{H} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$, i.e. given a, $\mathrm{b} \in Z_{256}$ on inputs, $2-PHT$ operation will produce values of $2 * a + b$ and $a + b$ on the outputs.

Armenian shuffle is $[9, 12, 13, 16, 3, 2, 7, 6, 11, 10, 15, 14, 1, 8, 5, 4]$. Four layers of $2-PHT$ boxes with Armenian shuffles between each 2 layers makes up the linear layer of the cipher and is identical to multiplication of the input 16-byte vector with the matrix M shown below in Figure 2.2.

$$\begin{bmatrix}
2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 4 & 4 \\
1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 \\
1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 2 & 1 & 1 \\
1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
2 & 1 & 16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 \\
2 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 \\
4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 \\
2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 1 & 16 & 8 \\
4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 8 & 4 \\
16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 \\
8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2
\end{bmatrix}.$$

Figure 2.2: Diffusion matrix M of SAFER+ algorithm [**26**].

2.2 KEY GENERATION SCHEDULE FOR SAFER+ WHITE-BOX ENCRYPTION AND DECRYPTION

By the key schedule for SAFER+ WB encryption we mean the key schedule for generating keys for corresponding regular black-box encryption. The key schedule starts with a 16-byte master key, and generates 32-bytes of sub-keys for each round. First it XORs 16 bytes of the master key to get the 17th byte value. Then for each round each byte of the previous round's key is rotated by 3 bits, and then the whole vector of 17 bytes is rotated by 1 byte, so the 17th byte is moved to the start of the vector. Then a fixed bias vector is XOR-ed with the first 16 bytes of the generated vector and the result is used as the next sub-key. The

algorithm of key schedule is shown in Figure 2.3. SAFER+ key schedule details can be found in [**26**]. It's obvious that given any sub-key for any round from the described key schedule one can easily compute all the keys used for all the rounds.

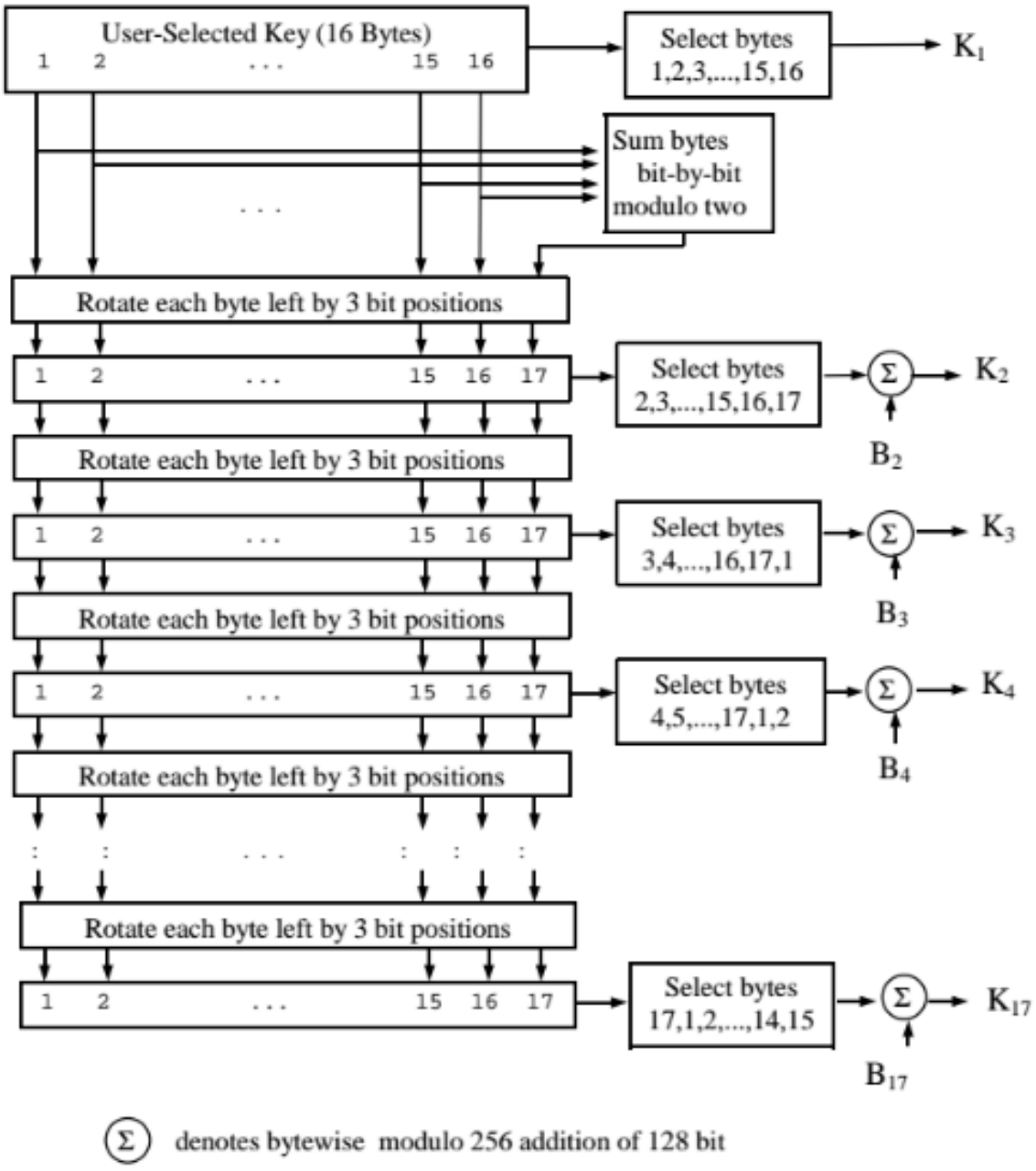

Figure 2.3: Structure of SAFER+ key schedule [**26**].

We modify the key schedule in order to eliminate any possibility of using the key schedule algorithm in key recovery attacks against the described WB implementation. We use the same key schedule algorithm for generating the 2nd key of the first round, after which SHA-

256 hash is applied over the last 32 bytes of sub-keys to generate the next 32 bytes. This is continued before R pairs of 16-byte sub-keys are generated, where R is the number of rounds in the SAFER+ (Figure 2.4). Then for each round r, keys (r and $2 * R - r$) are used. This modified key schedule algorithm prevents the attacker from getting the keys of any other round, if the key of some round is somehow compromised. SHA-256 is actually used as a Pseudo-Random Function (PRF) with a fixed seed of K_1, K_2.

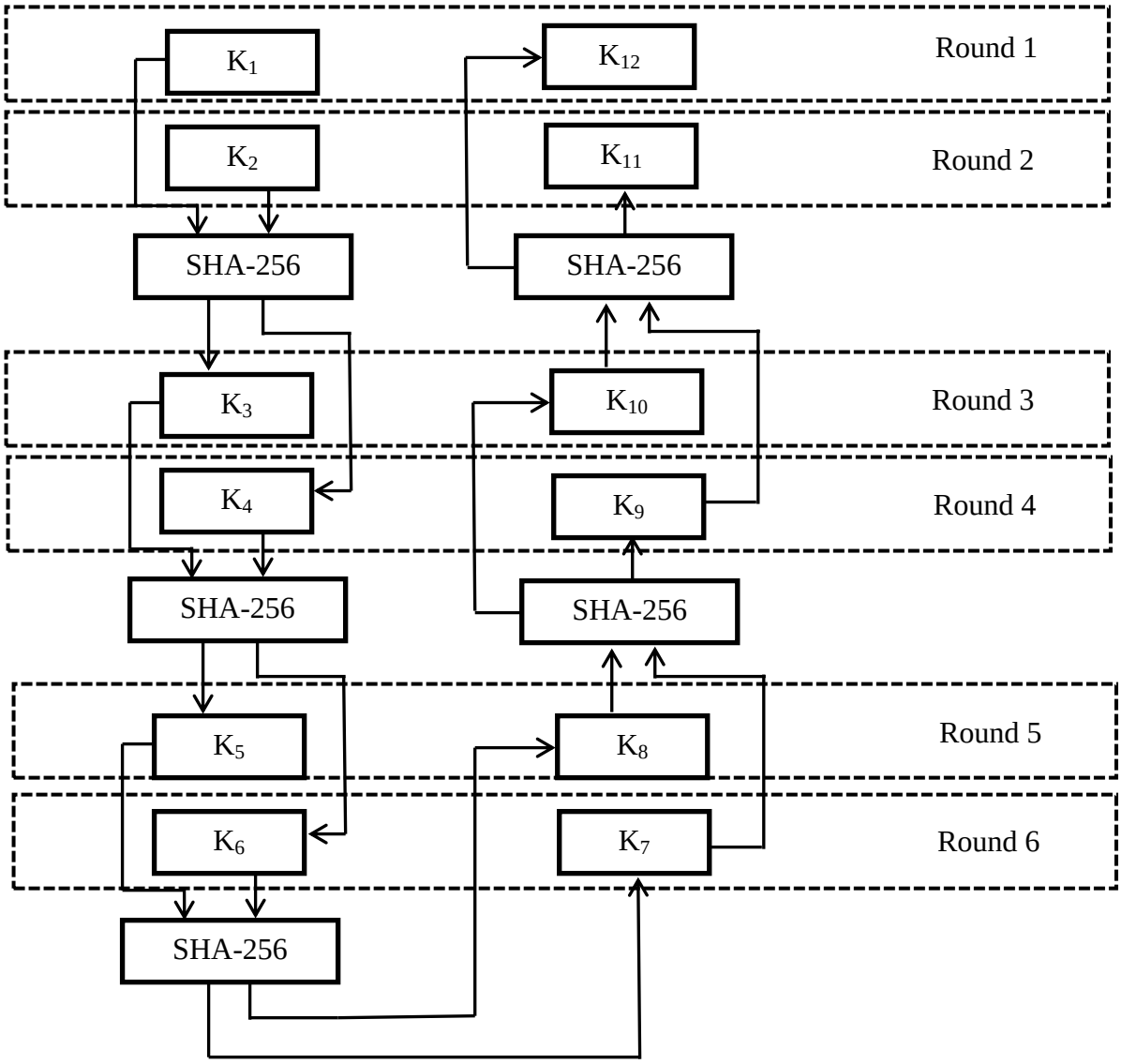

Figure 2.4: Structure of the modified key schedule for 6 round SAFER+

2.3 WHITE-BOX ENCRYPTION DESIGN RATIONALE

Our WB encryption is based on SAFER+ algorithm described in [**26**]. We're merging several steps of encryption rounds into look-up tables and obfuscating the table's inputs and outputs with input/output encodings (permutations) similar to those first introduced by Chow in [**15**]. We split the 16 bytes indices for each round into two sets $A = \{1,4,5,8,9,12,13,16\}$ and $B = \{2,3,6,7,10,11,14,15\}$ according to the order in which exponential (**exp**) and logarithmic (**log**) functions are applied. 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 $r = 1$

$$T_i^1(x) := exp\left(x \oplus k_{i1}^1\right) + k_{i2}^1 \quad \text{for } i \in A, \quad \textbf{(2.3.1)}$$

$$T_i^1(x) := log\left(x + k_{i1}^1\right) \oplus k_{i2}^1 \quad \text{for } i \in B, \quad \textbf{(2.3.2)}$$

for $2 \leq r \leq 6$

$$T_i^r(x) := exp(x \oplus k_{i1}^r) + k_{i2}^r \quad \text{for } i \in A, \quad \textbf{(2.3.3)}$$

$$T_i^r(x) := log(x + k_{i1}^r) \oplus k_{i2}^r \quad \text{for } i \in B, \quad \textbf{(2.3.4)}$$

for $r = 7$

$$T_i^7(x) := x \oplus k_i^{13} \quad \text{for } 1 \leq i \leq 16; \quad \textbf{(2.3.5)}$$

For security reasons, we randomly divide the output value into 2 parts that sum up to the original value. Given that, we need 16 tables that map 1 byte to 2 bytes for round 1, 2 bytes to 2 bytes for rounds 2 to 6, and 2 bytes to 1 byte for round 7 in the following manner.

Let's denote the i[th] mapping for input value x for round $r = 1$ as $\mathrm{T}_{i1}^1(\mathrm{x})$ and $\mathrm{T}_{i2}^1(\mathrm{x})$, i^{th} pair of mappings of the r^{th} round, for rounds $r = 2..6$, for input values of x_1 and x_2 by $\mathrm{T}_{i1}^r(x_1, x_2)$ and $\mathrm{T}_{i2}^r(x_1, x_2)$ and for round $r = 7$, i^{th} mapping for input values x_1 and x_2 by $\mathrm{T}_i^7(x_1, x_2)$.

Taking into account also the input and output permutations **IP** and **OP**, we can make the following definitions:

for $r = 1$

$$T_{i,1}^{1}(x) := exp\left(\mathbf{IP}_i(x) \oplus k_{i1}^{1}\right) + k_{i2}^{1} + R_i^1(x) \quad \text{for } i \in A, \quad \textbf{(2.3.6)}$$

$$T_{i,1}^{1}(x) := log\left(\mathbf{IP}_i(x) + k_{i1}^{1}\right) \oplus k_{i2}^{1} + R_i^1(x) \quad \text{for } i \in B, \quad \textbf{(2.3.7)}$$

$$T_{i,2}^{1}(x) := -R_{i1}^{1}(x) \quad \text{for } 1 \le i \le 16, \quad \textbf{(2.3.8)}$$

for $2 \le r \le 6$

$$T_{i,1}^{r}(x_1, x_2) := exp((x_1 + x_2 - S_i) \oplus k_{i1}^{r}) + k_{i2}^{r} + R_i^r(x_1, x_2) \quad \text{for } i \in A, \quad \textbf{(2.3.9)}$$

$$T_{i,1}^{r}(x_1, x_2) := log(x_1 + x_2 - S_i + k_{i1}^{r}) \oplus k_{i2}^{r} + R_i^r(x_1, x_2) \quad \text{for } i \in B, \quad \textbf{(2.3.10)}$$

$$T_{i,2}^{r}(x_1, x_2) := -R_i^r(x_1, x_2) \quad \text{for } 1 \le i \le 16, \quad \textbf{(2.3.11)}$$

for $r = 7$

$$T_i^7(x_1, x_2) := \mathbf{OP}_i((x_1 + x_2 - S_i) \oplus k_i^{13}) \quad \text{for } 1 \le i \le 16, \quad \textbf{(2.3.12)}$$

where $R_i^r(x_1, x_2)$ is a random function for each round, byte and input values and S_i is a compensation for random values added in the $2 - PHT$ boxes described later.

2.3.1 STRUCTURE OF SAFER+ WHITE-BOX ALGORITHM ROUND

The structure of SAFER+ round using encryption boxes is shown in Figure 2.5. Two types of boxes are used: $E - Boxes$ and $2 - PHT$ boxes. $E - boxes$ are accounting for the round confusion operations, while $2 - PHT$ boxes are similar to $2 - PHT$ boxes of regular SAFER+ algorithm described in [**26**], except they operate on 2 pairs of bytes. Armenian shuffles also operate on byte pairs, i.e. the permutation is applied on consecutive byte pairs, instead of single bytes.

For each round $1 \le r \le 6$ the confusion step is followed by layers of $2 - PHT$ boxes and Armenian shuffles. Tables will be provided for applying $2 - PHT$ operation on each pair of outputs.

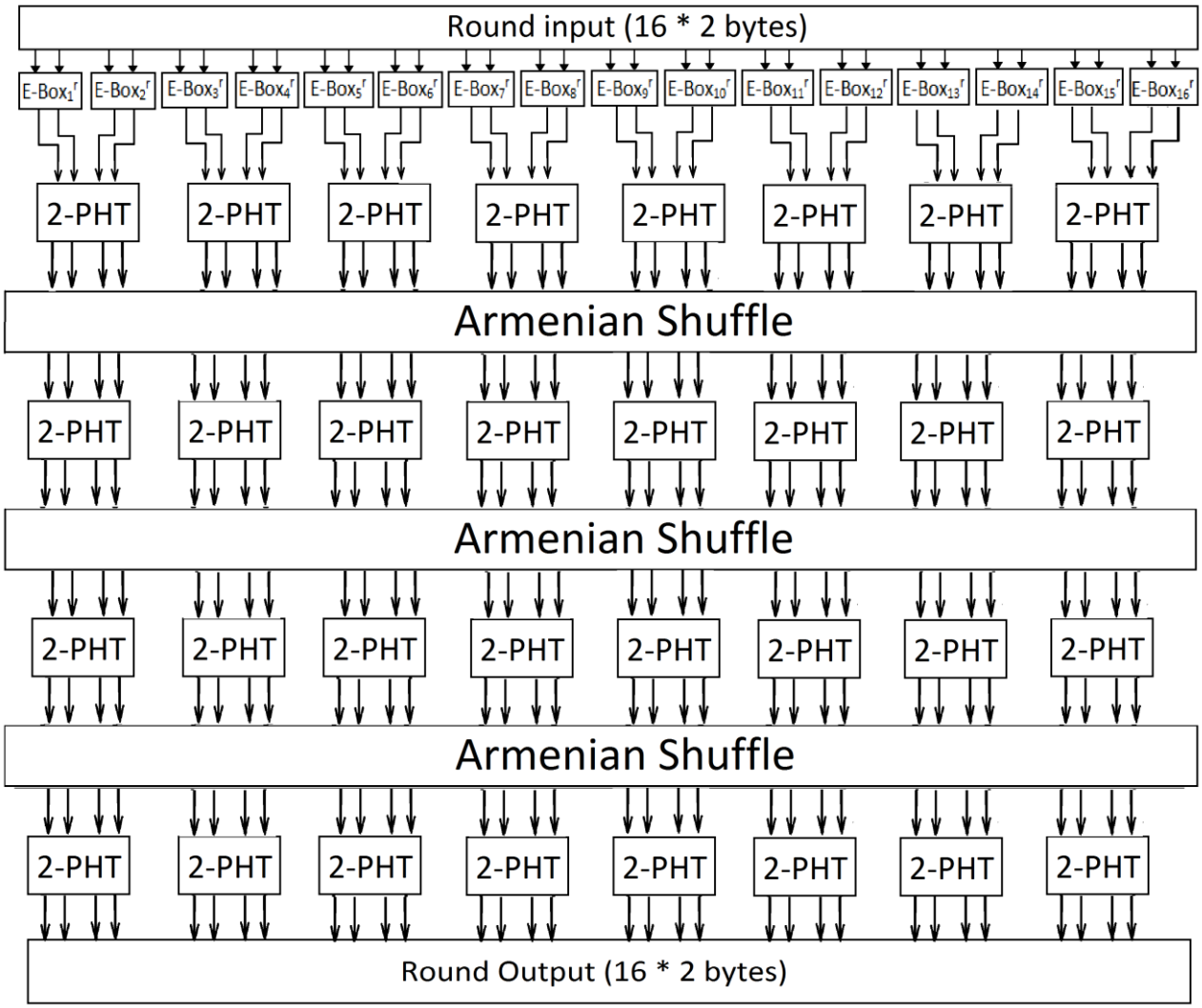

Figure 2.5: Round structure of SAFER+ White-box encryption.

2.3.2 E-BOXES

E-Boxes are look-up tables for getting values of $T_{i1}^{1}(x)\ and\ T_{i2}^{1}(x)$ for round r=1, values of $T_{i1}^{r}(x_1, x_2)\ and\ T_{i2}^{r}(x_1, x_2)$ for all $x_1\ and\ x_2$ for rounds $2 \le r \le 6$ and values of $T_{i}^{7}(x_1, x_2)$ for round r=7. We apply random encodings to the $E - box$ inputs and outputs for all the rounds. Thirty two random output encodings (permutations) $\mathrm{f}_1^r, \mathrm{f}_2^r, \ldots, \mathrm{f}_{32}^r$ are generated at WB table generation phase per round for rounds $1 \le r \le 6$, where $\mathrm{f}_i^r : Z_{256} \to Z_{256}$, and thirty two input encodings $\mathrm{g}_1^r, \mathrm{g}_2^r, \ldots, \mathrm{g}_{32}^r$ are generated for rounds $2 \le r \le 6$ $\mathrm{g}_i^r : Z_{256} \to Z_{256}$. We apply inverse permutations of $\mathrm{g}_{2*i-1}^r\ and\ \mathrm{g}_{2*i}^r$ on the inputs of the $\mathrm{E} - \mathrm{Box}_i^r$ and

output encodings f_{2*i-1}^{r} *and* f_{2*i}^{r} on the outputs. So we get the following formulas for the outputs of E-boxes:

for $r = 1$

$$E_{i1}^{1}(x) \stackrel{\text{def}}{=} \mathrm{f}_{2*i-1}^{r}\left(T_{i1}^{1}(x)\right), \qquad \textbf{(2.3.2.1)}$$

$$E_{i2}^{1}(x_1, x_2) \stackrel{\text{def}}{=} \mathrm{f}_{2*i}^{r}\left(T_{i2}^{1}(x)\right), \qquad \textbf{(2.3.2.2)}$$

for $2 \le r \le 6$

$$E_{i1}^{r}(x_1, x_2) \stackrel{\text{def}}{=} \mathrm{f}_{2*i-1}^{r}\left(T_{i1}^{r}\left({\mathrm{g}_{2*i-1}^{r}}^{-1}(x_1), {\mathrm{g}_{2*i}^{r}}^{-1}(x_2)\right)\right), \qquad \textbf{(2.3.2.3)}$$

$$E_{i2}^{r}(x_1, x_2) \stackrel{\text{def}}{=} \mathrm{f}_{2*i}^{r}\left(T_{i2}^{r}\left({\mathrm{g}_{2*i-1}^{r}}^{-1}(x_1), {\mathrm{g}_{2*i}^{r}}^{-1}(x_2)\right)\right), \qquad \textbf{(2.3.2.4)}$$

for $r = 7$

$$E_{i}^{1}(x_1, x_2) = T_{i}^{7}\left({\mathrm{g}_{2*i-1}^{r}}^{-1}(x_1), {\mathrm{g}_{2*i}^{r}}^{-1}(x_2)\right). \qquad \textbf{(2.3.2.5)}$$

So from this formulas we can see, that E-boxes essentially provide access to the values of T_{i1}^{r} and T_{i2}^{r} given by formulas 2.3.6-2.3.12, but also apply input and output encodings. These encodings make the tables secure, because if the values of T_{i1}^{r} and T_{i2}^{r} were given out by value without external encodings, the attacker might easily extract the values of secure keys k_{i1}^{r} and k_{i2}^{r} by simply computing the values of

$$T_{i,1}^{r}(x_1, x_2) + T_{i,2}^{r}(x_1, x_2) = exp((x_1 + x_2 - S_i) \oplus k_{i1}^{r}) + k_{i2}^{r} \quad \text{for } i \in A,$$

$$T_{i,1}^{r}(x_1, x_2) + T_{i,2}^{r}(x_1, x_2) = log(x_1 + x_2 - S_i + k_{i1}^{r}) \oplus k_{i2}^{r} \quad \text{for } i \in B;$$

As there are just 256 possible values for each of S_i, k_{i1}^{r} and k_{i2}^{r}, the attacker can simply brute-force all 256^3 possible values, build tables for each of these values and check which one matches the given tables.

2.3.3 2-PHT boxes

$2 - \mathrm{PHT}$ boxes are look-up tables that multiply the 2-byte input with matrix H = $\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$, i.e. given a, b $\in Z_{256}$ on inputs, the box will produce values of $2 * a + b$ and $a + b$ on the outputs. In our WB implementation $2 - \mathrm{PHT}$ boxes must operate on 4 bytes of input, because each input is split in 2 parts and encoded with a permutation. One can easily notice that this kind of a box can be built with 2 standard (2-byte input, 2-byte output) $2 - \mathrm{PHT}$

boxes as shown in Figure 2.6 and that the sum of pairs of the outputs $2a_1 + b_1 + 2a_2 + b_2 = 2(a_1 + a_2) + (b_1 + b_2)$ and $a_1 + b_1 + a_2 + b_2 = (a_1 + a_2) + (b_1 + b_2)$ match the required outputs. This means that $2 - \mathrm{PHT}$ operation can be successfully implemented on 2 pairs of numbers using two $2 - \mathrm{PHT}$ boxes of size $256 * 256 * 2\,bytes = 128\,KB$ each. Random input and output encodings (permutations) are applied on all the inputs and outputs for all the $2 - \mathrm{PHT}$ boxes. Input encodings of the first layer of $2 - \mathrm{PHT}$ boxes match the output encodings of the corresponding E-boxes of the previous round and the output encoding of the last layer of $2 - \mathrm{PHT}$ boxes must match the input encodings of the corresponding E-boxes of the same round. This lets the input/output encodings successfully neutralize each other, resulting to the proper execution of the algorithm.

If input encodings $\mathrm{f}_1, \mathrm{f}_2$ and output encodings $\mathrm{g}_1, \mathrm{g}_2$ are used for the 2-byte $2 - \mathrm{PHT}$ box of round r, layer $l(l\epsilon[1..4])$, byte $B(B\epsilon[1..16])$, the outputs for input values of x_1, x_2 will be

$$2 - \mathrm{PHT}_1(x_1, x_2) = \mathrm{g}_1\left(2 * f_1^{-1}(x_1) + f_2^{-1}(x_2) + S_B^{rl}\right), \quad \textbf{(2.2.3.1)}$$

$$2 - \mathrm{PHT}_2(x_1, x_2) = \mathrm{g}_2\left(f_1^{-1}(x_1) + f_2^{-1}(x_2) + S_{B+1}^{rl}\right). \quad \textbf{(2.2.3.2)}$$

The values of S_B^{rl} are randomly generated for each $2 - \mathrm{PHT}$ box for each round, layer and byte. Later these values are compensated in the subsequent $E - boxes$ of the next round, taking into account the coefficients which apply to each value of S_B^{rl}.

For the given inputs a_1, a_2, b_1, b_2, given that permutations f_1, f_2, f_3, f_4 and g_1, g_2, g_3, g_4 are applied on the inputs and outputs of a $2 - PHT$ box at round r, layer l and byte B, the left $2 - \mathrm{PHT}$ box in the Figure 2.6 will output values of $\mathrm{g}_1\left(2 * f_1^{-1}(a_1) + f_2^{-1}(b_1) + S_B^{rl}\right)$ and $\mathrm{g}_2\left(f_1^{-1}(a_1) + f_2^{-1}(b_1) + S_{B+1}^{rl}\right)$, and the right $2 - \mathrm{PHT}$ box will output the values of $\mathrm{g}_3\left(2 * f_3^{-1}(a_2) + f_4^{-1}(b_2) + S_{B+2}^{rl}\right)$ and $\mathrm{g}_4\left(f_3^{-1}(a_2) + f_4^{-1}(b_2) + S_{B+4}^{rl}\right)$.

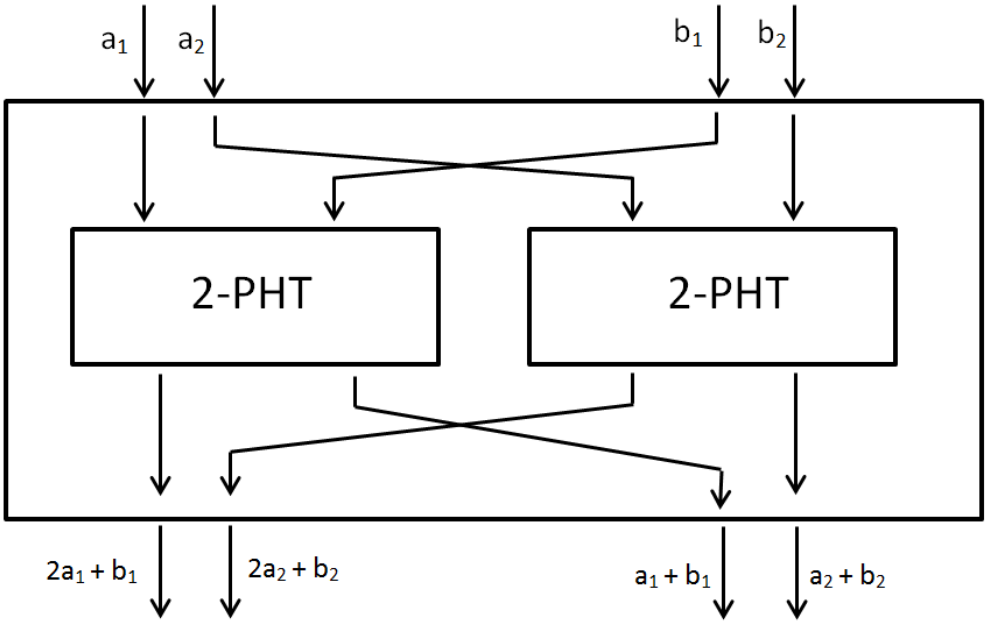

Figure 2.6: $2-\mathrm{PHT}$ operation for 2 pairs of bytes

Proposition 1: The result of the WB encryption presented above is equivalent to the black-box encryption result of SAFER+ accurate to the input and output permutations $\mathbf{IP}_i$ *and* $\mathbf{OP}_i$ applied on the WB input and output states.

Proof: The proof is straightforward. We have to compare two functions the one which makes a black-box encryption based on the formulas 2.3.1-2.3.5 and formulas implementing WB algorithm described in by formulas 3.2.1 and 3.2.5, 2.2.3.1 and 2.2.3.2. One can easily notice that

$$T_{i1}^{r}(x_1, x_2) + T_{i2}^{r}(x) = exp((x_1 + x_2) \oplus k_{i1}^{r}) + k_{i2}^{r} = T_i^r((x_1 + x_2) \text{ for } i \in A,$$

$$T_{i1}^{r}(x_1, x_2) + T_{i2}^{r}(x) = log(x_1 + x_2 + k_{i1}^{r}) \oplus k_{i2}^{r} = T_i^r((x_1 + x_2) \text{ for } i \in B.$$

So E-boxes correctly apply the SAFER+ confusion step on the input value $(x_1 + x_2)$. All the output permutations of E-boxes match the input permutations of the corresponding $2-\mathrm{PHT}$ boxes, and the output encodings of the last layer of $2-\mathrm{PHT}$ boxes match the input encodings of E-boxes, so it's obvious that all the encodings cancel each other. So the only difference between ordinary black-box SAFER+ encryption procedure and WB encryption procedure described above are the input and output encodings $\mathbf{IP}_i$ *and* $\mathbf{OP}_i$ applied

accordingly to the input of the E-boxes of round 1 and the output of $\mathrm{E}-\mathrm{boxes}$ of the final round.

2.4 WHITE-BOX DECRYPTION ALGORITHM

Decryption white-boxes can be built in a similar way to the encryption boxes. This chapter is devoted to the details of how to build and use decryption white-boxes. Similar to the encryption, the decryption 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 $r = 1$

$$DT_i^1(x) := log(x - k_{i2}^1) \oplus k_{i1}^1 \quad \text{for } i \in A, \tag{2.3.1}$$

$$DT_i^1(x) := exp(x \oplus k_{i2}^1) - k_{i1}^1 \quad \text{for } i \in B, \tag{2.3.2}$$

for $2 \leq r \leq 6$

$$DT_i^r(x) := log(x - k_{i2}^r) \oplus k_{i1}^r \quad \text{for } i \in A, \tag{2.3.3}$$

$$DT_i^r(x) := exp(x \oplus k_{i2}^r) - k_{i1}^r \quad \text{for } i \in B, \tag{2.3.4}$$

for $r = 7$

$$DT_i^7(x) := x \oplus k_i^{13} \quad \text{for } 1 \leq i \leq 16; \tag{2.3.5}$$

It is easily to check, that equations 2.3.1-2.3.5 apply the inverse operation of equations 2.2.1-2.2.5, I.E. $DT_i^r\left(T_i^r(x)\right) = x$, for any value of i, r and x. For $2 \leq r \leq 6$ and $i \in A$

$$DT_i^r\left(T_i^r(x)\right) = log(T_i^r(x) - k_{i2}^r) \oplus k_{i1}^r = log(exp(x \oplus k_{i1}^r) + k_{i2}^r - k_{i2}^r) \oplus k_{i1}^r$$
$$= log\left(exp(x \oplus k_{i1}^r)\right) \oplus k_{i1}^r = x \oplus k_{i1}^r \oplus k_{i1}^r = x.$$

The same check can be applied for the other values of r and i.

In a similar way to the encryption algorithm, these values of $DT_i^r(x)$ are actually randomly divided into 2 values, which sum up to the original value.

Let's denote the i[th] mapping for input value x for round r=1 as $\mathrm{DT}_{i1}^1(\mathrm{x})$ and $\mathrm{DT}_{i2}^1(\mathrm{x})$, i-th pair of mappings of the r^{th} round, for rounds $r = 2..6$, for input values of x_1 and x_2 by

$DT_{i1}^{r}(x_1, x_2)$ and $DT_{i2}^{r}(x_1, x_2)$ and for round $r = 7$, i^{th} mapping for input values $x_1$and x_2 by $T_i^7(x_1, x_2)$. We split the 16 bytes indices for each round into two sets $A = \{1,4,5,8,9,12,13,16\}$ and $B = \{2,3,6,7,10,11,14,15\}$ according to the order in which exponential (**exp**) and logarithmic (**log**) functions are applied. Taking into account also the input and output permutations **IP** and **OP**, we can make the following definitions:

for $r = 1$

$$DT_{i1}^{1}(x) := log\left(\mathbf{IP}_i(x) - k_{i2}^{1}\right) \oplus k_{i1}^{1} + R_i^1(x) \quad \text{for } i \in A,$$

$$DT_{i1}^{1}(x) := exp\left(\mathbf{IP}_i(x) \oplus k_{i2}^{1}\right) - k_{i1}^{1} + R_i^1(x) \quad \text{for } i \in B,$$

$$DT_{i2}^{1}(x) := -R_{i1}^{1}(x) \quad \text{for } 1 \le i \le 16,$$

for $2 \le r \le 6$

$$DT_{i1}^{r}(x_1, x_2) := log((x_1 + x_2 - k_{i2}^{r} - S_i)) \oplus k_{i1}^{r} + R_i^r(x_1, x_2) \quad \text{for } i \in A,$$

$$DT_{i1}^{r}(x_1, x_2) := exp((x_1 + x_2) \oplus k_{i2}^{r} - S_i) - k_{i1}^{r} + R_i^r(x_1, x_2) \quad \text{for } i \in B,$$

$$DT_{i2}^{r}(x_1, x_2) := -R_i^r(x_1, x_2) \quad \text{for } 1 \le i \le 16,$$

for $r = 7$

$$DT_i^7(x_1, x_2) := \mathbf{OP}_i((x_1 + x_2 - S_i) \oplus k_i^{13}) \quad \text{for } 1 \le i \le 16;$$

where $R_i^r(x_1, x_2)$ is a random function for each round, byte and input values and S_i is a compensation for random values added in the 2-PHT boxes described later.

2.4.1 STRUCTURE OF SAFER+ WHITE-BOX DECRYPTION ROUND

The structure of WB decryption round is similar to the encryption structure. Two types of boxes are used: E-Boxes and 2-PHT boxes. Figure 2.7 illustrates the structure of a single intermediate round.

$D - boxes$ are responsible for the confusion step, i.e. for undoing the effect of the encryption E-boxes, and 2-PHT boxes are the opposite of the encryption $2 - PHT$ boxes. Both boxes are described in detail in the following chapters.

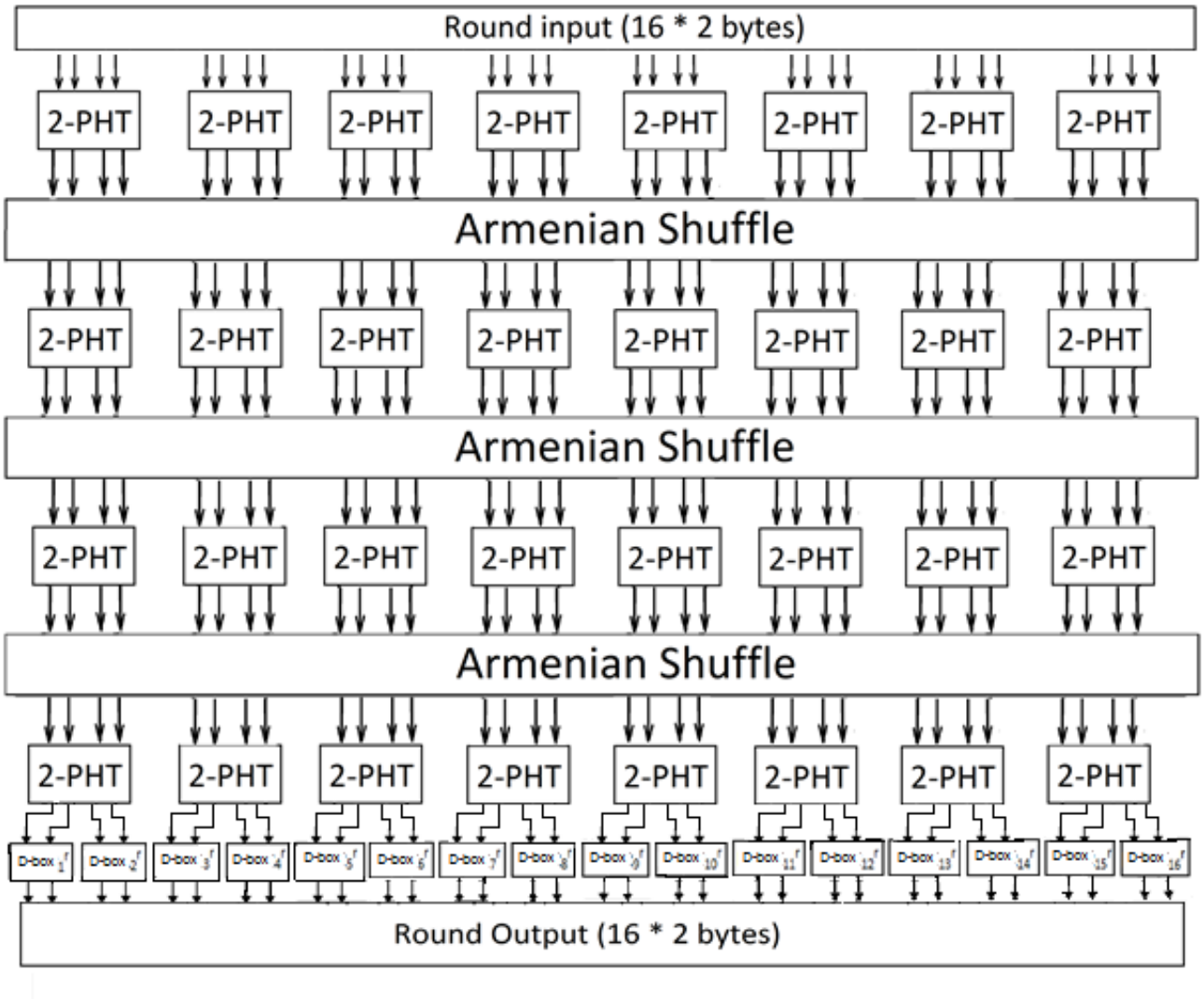

Figure 2.7: Round structure of SAFER+ WB decryption

2.4.2 D-BOXES

D-Boxes are look-up tables for getting values of $DT_{i1}^{1}(x)\ and\ DT_{i2}^{1}(x)$ for round r=1, values of $DT_{i1}^{r}(x_1, x_2)\ and\ DT_{i2}^{r}(x_1, x_2)$ for all $x_1\ and\ x_2$ for rounds $2 \le r \le 6$ and values of $DT_{i}^{7}(x_1, x_2)$ for round $r = 7$. Similar to the encryption $E - boxes$, we apply random encodings to the D-boxes outputs, and the inputs for all the rounds are also encoded. Thirty two random output encodings (permutations) $\mathrm{f}_1^r, \mathrm{f}_2^r, \ldots, \mathrm{f}_{32}^r$ are generated at WB table generation phase per round for rounds $1 \le r \le 6$, where f_i^r: $Z_{256} \to Z_{256}$, and thirty two $input\ encodinds$ $\mathrm{g}_1^r, \mathrm{g}_2^r, \ldots, \mathrm{g}_{32}^r$ are generated for rounds $2 \le r \le 6$ g_i^r: $Z_{256} \to Z_{256}$. We apply reverse permutations of $\mathrm{g}_{2*i-1}^r\ and\ \mathrm{g}_{2*i}^r$ on the inputs of the $\mathrm{D} - \mathrm{Box}_i^r$ and output

encodings $\mathrm{f}^r_{2*i-1}\ and\ \mathrm{f}^r_{2*i}$ on the outputs. So we get the following formulas for the outputs of E-boxes:

for $r = 1$

$$D^1_{i1}(x) \stackrel{\text{def}}{=} \mathrm{f}^r_{2*i-1}\left(DT^1_{i1}(x)\right), \quad \textbf{(2.4.2.1)}$$

$$D^1_{i2}(x) \stackrel{\text{def}}{=} \mathrm{f}^r_{2*i}\left(DT^1_{i2}(x)\right), \quad \textbf{(2.4.2.2)}$$

for $2 \le r \le 6$

$$D^r_{i1}(x_1, x_2) \stackrel{\text{def}}{=} \mathrm{f}^r_{2*i-1}\left(DT^r_{i1}\left({\mathrm{g}^r_{2*i-1}}^{-1}(x_1), {\mathrm{g}^r_{2*i}}^{-1}(x_2)\right)\right), \quad \textbf{(2.4.2.3)}$$

$$D^r_{i2}(x_1, x_2) \stackrel{\text{def}}{=} \mathrm{f}^r_{2*i}\left(DT^r_{i1}\left({\mathrm{g}^r_{2*i-1}}^{-1}(x_1), {\mathrm{g}^r_{2*i}}^{-1}(x_2)\right)\right), \quad \textbf{(2.4.2.4)}$$

for $r = 7$

$$D^1_i(x_1, x_2) = DT^7_i\left({\mathrm{g}^r_{2*i-1}}^{-1}(x_1), {\mathrm{g}^r_{2*i}}^{-1}(x_2)\right). \quad \textbf{(2.4.2.5)}$$

Equations 2.4.2.1-2.4.2.5 are similar to equations 2.3.2.1-2.3.2.5 used for encryption, though the input and output encodings do not require to be the same. If both encryption and decryption are implemented as white-boxes, any $E-box$ of round r will use different input/output encodings and random values than the corresponding $D-box$ of round r, so if one encrypts some data using $E-box$ at round r, then decrypts it with the $D-box$ of round r, the result will not be equivalent to the initial data.

2.4.3 Decryption 2-PHT boxes

Decryption $2-\mathrm{PHT}$ boxes multiply the input with matrix

$$\mathrm{H}^{-1} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix},$$

i.e. given a, b $\in Z_{256}$ on inputs, will produce values of $a-b$ and $2*\mathrm{b}-a$ on the outputs. In our WB implementation $2-\mathrm{PHT}$ boxes must operate on 4 bytes of input, because each input is divided into 2 values and encoded. One can easily notice that this kind of a box can be built with 2 standard (2-byte input, 2-byte output) $2-\mathrm{PHT}$ boxes as shown in Figure 2.8 and that the sum of pairs of the outputs $a_1 - b_1 + a_2 - b_2 = (a_1 + a_2) - (b_1 + b_2)$ and $2b_1 - a_1 + 2b_2 - a_2 = 2(b_1 + b_2) - (a_1 + a_2)$ match the required outputs. Random input

and output encodings (permutations) are applied on all the inputs and outputs for all the $2-\mathrm{PHT}$ boxes. Input encodings of the first layer of $2-\mathrm{PHT}$ boxes match the output encodings of the corresponding D-boxes of the same round and the output encoding of the last layer of $2-\mathrm{PHT}$ boxes must match the input encodings of the corresponding D-boxes of the next round. This lets the input/output encodings successfully neutralize each other, resulting to the proper execution of the algorithm.

If input encodings $\mathrm{f}_1, \mathrm{f}_2$ and output encodings $\mathrm{g}_1, \mathrm{g}_2$ are used for the 2-byte decryption $2-\mathrm{PHT}$ box of round r, layer $l\epsilon[1..4]$, byte $B\epsilon[1..16]$, the outputs for input values of x_1, x_2 will be

$$2-\mathrm{PHT}_1(x_1, x_2) = \mathrm{g}_1\left(f_1^{-1}(x_1) - f_2^{-1}(x_2) + S_B^{rl}\right) \quad (2.2.3.1)$$

$$2-\mathrm{PHT}_2(x_1, x_2) = \mathrm{g}_2\left(2 * f_2^{-1}(x_2) - f_1^{-1}(x_1) + S_{B+1}^{rl}\right) \quad (2.2.3.2)$$

The values of S_B^{rl} are randomly generated for each $2-\mathrm{PHT}$ box for each round, layer and byte. Later these values are compensated in the subsequent $D-boxes$ of the next round, taking into account the coefficients which apply to each value of S_B^{rl}.

For the given inputs a_1, a_2, b_1, b_2, given that permutations f_1, f_2, f_3, f_4 and g_1, g_2, g_3, g_4 are applied on the inputs and outputs of a $2-PHT$ box at round r, layer l and byte B, the left $2-\mathrm{PHT}$ box in the Figure 2.8 will output values of $\mathrm{g}_1\left(f_1^{-1}(a_1) - f_2^{-1}(b_1) + S_B^{rl}\right)$ and $\mathrm{g}_2\left(2 * f_2^{-1}(b_1) - f_1^{-1}(a_1) + S_{B+1}^{rl}\right)$, and the right side decryption $2-\mathrm{PHT}$ box will output the values of $\mathrm{g}_3\left(f_3^{-1}(a_2) - f_4^{-1}(b_2) + S_{B+2}^{rl}\right)$ and $\mathrm{g}_4\left(2 * f_4^{-1}(b_2) - f_3^{-1}(a_2) + S_{B+4}^{rl}\right)$.

One can easily notice, that decryption $2-PHT$ boxes apply the inverse operation of encryption $2-PHT$ boxes given by formulas **(2.2.3.1)** and **(2.2.3.2)**, yet this does not mean, that if white-boxes are used for both encryption and decryption operations a single decryption 2-PHT table for some round, layer and byte will perform the exact opposite operation of a single encryption 2-PHT table of the same round, layer and byte. This is because the values of S_B^{rl} and input/output permutations used for protection of the tables are different for encryption and decryption. This is crucial, because usage of the same values of S_B^{rl} and input/output permutations might result to weaker system, and the attack having both encryption and decryption tables built with those values can learn some information regarding the permutations used.

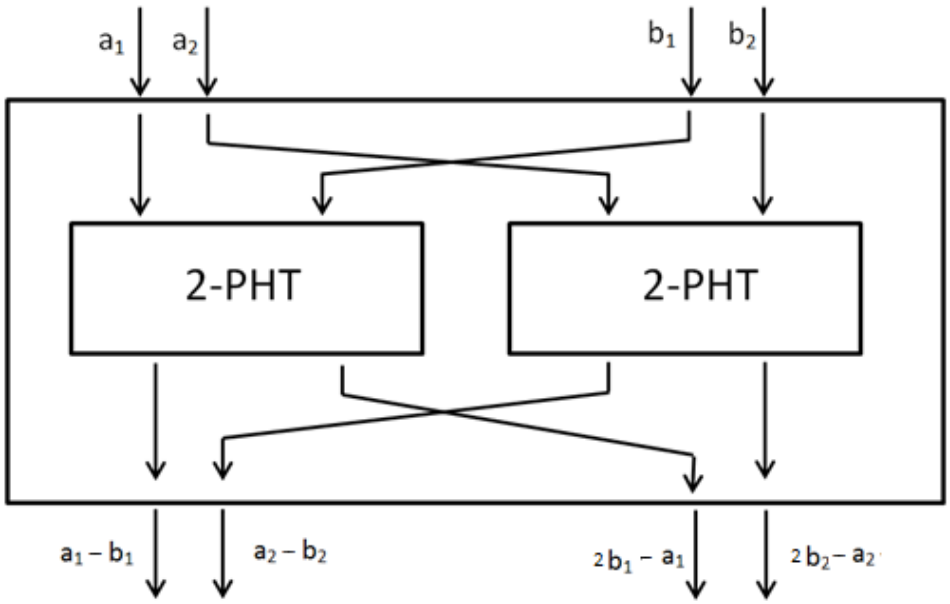

Figure 2.8: decryption $2-\mathrm{PHT}$ operation for 2 pairs of bytes

2.5 SECURITY ANALYSIS

Definition 1.1: A WB encryption scheme $WBES \stackrel{\text{def}}{=} (\mathrm{WBGenEnc},\ \mathrm{WBEnc})$ is *secure against key recovery attacks* if given $T_{enc} \leftarrow \mathrm{WBGenEnc}\,(k)$ it is computationally infeasible to extract the secret key k.

Definition 1.2: A WB encryption scheme $WBES \stackrel{\text{def}}{=} (\mathrm{WBGenEnc},\ \mathrm{WBEnc})$ is *secure against reverse-engineering attacks* if given $T \leftarrow \mathrm{WBGen}\,(k)$ and any ciphertext $c = \mathrm{Enc}_k(m)$ it is computationally infeasible to compute m.

Definition 1.3: A WB encryption scheme $WBES \stackrel{\text{def}}{=} (\mathrm{WBGenEnc},\ \mathrm{WBEnc})$ is *computationally secure* if for every $m_0, m_1 \in \mathrm{M}$ the values $\mathrm{WBEnc_k}(m_0)$ and $\mathrm{WBEnc_k}(m_1)$ are computationally indistinguishable.

Definition 1.4: A WB scheme $WBES \stackrel{\text{def}}{=} (\mathrm{WBGenEnc},\ \mathrm{WBEnc})$ is considered *secure* if it is secure *against key recovery and reverse-engineering attacks, is computationally secure and* the corresponding black-box scheme is secure in the black-box attack context.

It was shown in [**26**] that SAFER+ black-box symmetric encryption algorithm is secure against differential and linear cryptanalysis attacks after 6 rounds of encryption. Black-box

encryption described in current document is similar to SAFER+, except it applied input and output encodings for round 1 input and last round's output. Clearly these encodings will not reduce the security of black-box encryption from the point of view of differential and linear cryptanalysis.

There is no common way to proof the security of a WB algorithm yet the known algorithms were successfully attacked by the BGE attack [**16**], so the security against this particular attack will be addressed. Chow defined two properties for estimating the security of WB tables, namely WB diversity and ambiguity. WB diversity is the number of possible construction of any given table, while ambiguity is the number of constructions which could lead to the given exact table [**15**]. Having good diversity and ambiguity properties is a pre-requisite for a possibly good white-box algorithm. These metric are discussed later.

2.5.1 DIVERSITY AND AMBIGUITY

Two types of boxes are used in our WB implementation of SAFER+: $E-boxes$ and $2-PHT$ boxes. $E-boxes$ of round r depend on the input and output encodings of 2 bytes of input and output, round key k_{i1}^{r} , random parameter S_i and random function $R_i^r(x_1, x_2)$. There are $256!$ possible bijections for 1 byte, so there are $(256!)^4$ possibilities for input and output encodings for a single $E-box$. There are 256 possible values of k_{i1}^{r}, 256 values for S_i and $256^{256*256}$ possible functions $R_i^r(x_1, x_2)$. So the WB diversity of a single $E-box$ is $(256!)^4 * 256 * 256 * 256^{256*256} \approx 2^{4*1684+8+8+256*256} = 2^{72288}$. For $2-PHT$ tables, there are $(256!)^4$ possible input and output encodings and 256 values for S_B^{rl}, so WB diversity is $(256!)^4 * 256 \approx 2^{1692}$. One can notice that this value is significantly smaller compared to the diversity of $E-boxes$, which implies that these boxes must be less secure. But compared to the diversities of boxes in Chow's WB, this value is acceptable.

There is no known way of computing the WB ambiguity for a given table. So instead of directly counting the value, we must somehow estimate it or provide a lower bound. In our case, for the $E-box$, it's easy to notice that one can manipulate the output by the output encodings only, i.e. for any given $E-box$, for any fixed input encoding, values of k_{i1}^{r}, S_i and $R_i^r(x_1, x_2)$, we can change the output encodings in such a way, that the resulting

$E-box$ will exactly match the given one. This means that the lower bound for ambiguity of the $E-boxes$ is $(256!)^2 * 256 * 256 * 256^{256*256} \approx 2^{2*1684+8+8+256*256} = 2^{68920}$. The same is true for $2-PHT$ boxes, i.e. for any fixed $2-PHT$ box and values of S_B^{rl} and input encodings, we can find output encodings in such a way, that the resulting tables matches the given one. This means that the lower bound for $2-PHT$ table is $(256!)^2 * 256 \approx 2^{2*1684+8} = 2^{3376}$.

Table 2.5.1: Estimated WB diversity and ambiguity for our and Chow's WB tables.

	Diversity	Ambiguity
$E-box$	2^{72288}	2^{68920}
$2-PHT$	2^{1692}	2^{3376}
Chow's type 1	2^{2420}	2^{546}
Chow's type 2	2^{769}	2^{129}
Chow's type 3	2^{699}	2^{117}
Chow's type 4	2^{133}	2^{48}

From Table 2.5.1 we can see, that both diversity and ambiguity of our tables are better that those of Chow. This is a good sign, that the WB must be secure. We can see, that the "weakest" tables in Chow's implementation are Type 4 tables, and the $2-PHT$ tables are much "weaker" compared to $E-boxes$. We can also see that the "weak" $2-PHT$ tables have stronger diversity and ambiguity values compared to Chow's tables. This is a good sign but does not guarantee absolute security.

2.5.2 SECURITY AGAINST BGE ATTACK

An algebraic attack against WB AES implementation was developed in 2004 called BGE attack [**16**]. It is shown in [**22**] that BGE attack can be applied to all substitution-linear transformation network (later SLT) ciphers which match the following definition.

Definition 2.1: A cipher is called SLT cipher if it can be specified as follows: It consists of R rounds where R>0. A single round r is a bijective function $F_{SLT}^r(x_1, x_2, .., x_s)$ on $GF(2^n)$

where $n = m \cdot s$ and $x_i \in GF(2^m)$. This function starts with the XOR-ing an n-bit length round key $k^r = (k_1^r, k_2^r, \dots, k_s^r)$ with the input$(x_1, x_2, .., x_s)$. That is, a value $y_i = x_i \oplus k_i$ is computed. Next the round computes $z_i = S_i^r(y_i)$ for all y_i where the (non-linear) invertible S-boxes $(S_1^r, S_2^r, .., S_s^r)$ are part of cipher. These two steps achieve confusion. The diffusion is realized by multiplying the outcome of the S-boxes with an invertible matrix M(r) over $GF(2^m)$.

The BGE attack proceeds in three steps:

- Transform the non-linear output encodings $Q_{i,j}^r$ to unknown $GF(2)$-affine transformation. (i.e. recover the non-linear part up to some unknown affine transformation)
- Fully determine the values of $Q_{i,j}^r$ using algebraic analysis.
- Obtain round keys for two consecutive rounds and recover the symmetric secret key using the reversibility of underlying key-schedule.

In our case the first step of BGE attack cannot be applied on the $E - boxes$, as the outputs of the $E - boxes$ are randomly divided into 2 parts that sum up to the real result. This significantly complicates the equations in the tables, so it's impossible to build a commutative group under composition. This deems the first step of BGE attack on the $E - boxes$ impossible. Any attack targeted on only one output of an E-box will fail because of totally random values of $R_i^1(x)$, so the attacker must try to analyze both outputs $T_{i1}^r(x_1, x_2)\ and\ T_{i2}^r(x_1, x_2)$ together, which makes it impossible to apply the BGE attack.

Though the first step of the attack cannot be applied on the $E - boxes$ to recover the random input-output permutations of E-boxes $\mathrm{f}_i^r\ and\ \mathrm{g}_i^r$ up to affine transformations, that can still be achieved by attacking the $2 - PHT$ boxes, which have much simpler structure.

However, the techniques for completely recovering the input and output encodings will not work in our case. The 2-byte input/output structure of our E-boxes is significantly different from SLT ciphers and makes it impossible for the attacker to recover the input/output encodings using methods of the BGE attack. Step 3 relies on the fact that

$T_i(x) = S(x \oplus k_i)$, i.e. the mapping $x \mapsto S^{-1} \circ T_i \circ P_i(x) = P_i(x) \oplus k_i$ is affine, which is not true in the case of SAFER+, because from formulas **(2.3.3)** and **(2.3.4)** we can see, that

the key is applied in two different ways, either by modulo 2 addition, or by modulo 256 addition. This stops the recovery of affine parasites of BGE attack, even if the rest of the steps succeed. Also the modified key schedule is irreversible, so the structure of key schedule cannot be used by the attacker.

2.5.3 USING DIFFERENT 2-PHT TABLES IN DIFFERENT LAYERS IS NECESSARY

In this section we will consider an attack on a weakened variant of the presented SAFER+ WB. Let's assume our WB uses the same $2-PHT$ tables for 2 different layers of the same round. Using the same tables looks like secure from the first sight, and could save up some space, resulting to lower memory requirements for the WB tables in total.

So let us assume that the same $2-PHT$ table is used for layer 1 and 4 for some intermediate round. He same conclusions could be made if the same tables are used for any other 2 layers. Let's denote the input encodings of the first 2 bytes as f_1 and f_2. As we are using the same $2-PHT$ boxes for the 4th layer, then the same input encodings must be used there. We know that the input for the first byte of the 4th layer can be expressed by the input of the first layer in the following way:

$$f_1{}^{-1}(x_3) + f_2{}^{-1}(x_4) = 2 * \left(f_1{}^{-1}(x_1) + f_2{}^{-1}(x_2)\right) + c, \qquad \textbf{(2.5.3.1)}$$

where x_1 and x_2 are the first 2 inputs of the round, x_3 and x_4 are the first 2 inputs of the 4th layer of $2-PHT$ and c is the contribution of the other values on the inputs of the 2-PHT table's inputs. So if we keep the other input values constant(let's say 0), for each value of x_1 and x_2 we will get one equation of $(\mathbf{2.5.3.1})$, where c will be a constant. This set of 256^2 equations will have 256*2+1 unknowns, i.e. $f_1{}^{-1}(x)$ and $f_2{}^{-1}(x)$ for all 256 values of x and c, which means that we have more than enough linear equations to solve them for the values of $f_1{}^{-1}$ and $f_2{}^{-1}$. In a similar way the other values of $f_i{}^{-1}$can be recovered. Having these values one can easily attack the $E-boxes$ for key extraction.

Let's show how the secure keys can be extracted from E-boxes, given input and output permutations. Having the values of input and output permutations, means we can get the values of functions $T_{i,1}^r(x_1, x_2)$ and $T_{i,2}^r(x_1, x_2)$ for each $x_1, x_2 \epsilon Z_{256}$ given by formulas **(2.3.9) - (2.3.11)**. So by summing up these 2 values, we get the values of

$$T_{i,1}^r(x_1, x_2) + T_{i,2}^r(x_1, x_2) = exp((x_1 + x_2 - S_i) \oplus k_{i1}^r) + k_{i2}^r \text{ for } i \in A,$$

$$T_{i,1}^r(x_1, x_2) + T_{i,2}^r(x_1, x_2) = log(x_1 + x_2 - S_i + k_{i1}^r) \oplus k_{i2}^r \text{ for } i \in B.$$

This means we've got access to functions

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

$$F_i^r(x) := log(x - S_i + k_{i1}^r) \oplus k_{i2}^r \qquad \text{for } i \in B,$$

which are very similar to formulas **(2.3.1)- (2.3.4)**, except a random values S_i is subtracted. These values of S_i can be computed from 2-PHT boxes, as they are also not protected by input/output permutations any more. Then the attacker can brute-force all 256^2 possible values to extract the values of k_{i1}^r and k_{i2}^r. In cases when more than 1 pair of k_{i1}^r and k_{i2}^r satisfies the formulas, the relations between keys of different rounds can be used to filter the proper keys. Despite the fact that the changed key schedule does not allow computation of round keys based on some other round's key pair, it still allows checking if a given set of keys for all rounds results to a proper key schedule.

We proved that using the same $2 - PHT$ boxes in different layers of the same round creates security issues. Similar techniques can be applied for proving that using the same $2 - PHT$ boxes in the different rounds is also unsafe.

2.5.4 COLLISION ATTACK ON E-BOXES

In this chapter we describe a possible attack on our E-boxes, which could work, but was programmatically checked to be impossible. The main idea of this attack is to gain information about the output encodings of the $E - boxes$ by analyzing the collisions between different outputs. $E - boxes$ are built with equations **(2.3.2.1) - (2.3.2.5),** while the values of T_i^r are computed by formulas **(2.3.6) - (2.3.12).** Particularly from the formulas **(2.3.9)** and **(2.3.10)** one can notice, that there are 256^2 possible inputs for x_1, x_2 pair, but just 256 values for $x_1 + x_2$, i.e. the values of $T_{i1,}^r(x_1, x_2)$ will have collisions.

Let's say $T_{i,1}^r(x_1, x_2) = T_{i,1}^r(x_3, x_4)$ for some values x_1, x_2, x_3, x_4 and $i \in A$, i.e.

$$exp((x_1 + x_2 - S_i) \oplus k_{i1}^r) + k_{i2}^r + R_i^r(x_1, x_2) = exp((x_3 + x_4 - S_i) \oplus k_{i1}^r) + k_{i,2}^r + R_i^r(x_3, x_4).$$

This can be achieved if $x_1 + x_2 - S_i = x_3 + x_4 - S_i$ and $R_i^r(x_1, x_2) = R_i^r(x_3, x_4)$, or randomly. The first statement $x_1 + x_2 - S_i = x_3 + x_4 - S_i$ happens pretty often, because there are just 256 values for $x_1 + x_2$ while there are 256^2 values for x_1, x_2 pair. $R_i^r(x_1, x_2) = R_i^r(x_3, x_4)$ happens less frequently.

So for each collision we can say, not certainly but with a high probability that $x_1 + x_2 = x_3 + x_4$. Having this, the attacker may use collisions of values of the $E - box$ outputs to construct a set of linear equations like

$$f_1^{-1}(x_1) + f_2^{-1}(x_2) = f_1^{-1}(x_3) + f_2^{-1}(x_4).$$

Yet the attacker is not sure if all the equations for all the collisions are certainly true, but he/she knows that each equation is true with some probability. We have programmatically checked that the attacker never gets enough equations to solve them, and knowing that some of them are not true makes solutions of these set of equations, or even finding possible candidate solutions for it impossible.

2.5.5 COLLISION ATTACK ON 2-PHT BOXES

Another possible attack could be trying to user the collisions of the outputs of 2-PHT boxes to gain some information about the input/output permutations used. The attacker could analyze the tables given by formula **(2.2.3.2)**. One can notice that there are 256^2 possible input values for these boxes and just 256 possible outputs, which means that collisions here are pretty common.

So if

$$2 - \mathrm{PHT}_2(x_1, x_2) = 2 - \mathrm{PHT}_2(x_3, x_4)$$

for some values of x_1, x_2, x_3, x_4, then we get

$$\mathrm{g}_2\left(f_1^{-1}(x_1) + f_2^{-1}(x_2) + S_{B+1}^{rl}\right) = \mathrm{g}_2\left(f_1^{-1}(x_3) + f_2^{-1}(x_4) + S_{B+1}^{rl}\right).$$

From this we get

$$f_1^{-1}(x_1) + f_2^{-1}(x_2) = f_1^{-1}(x_3) + f_2^{-1}(x_4).$$

This way the attacker can build a set of linear equations and try to solve them to get the values of functions f_i.Yet this attack fails because this set of equations has many possible solutions, and the functions f_i are only used in these tables and the corresponding

$E-boxes$. This does not allow the attacker to gain enough information about these permutation functions f_i to recover them.

2.5.6 FREQUENCY ANALYSES ARE NOT POSSIBLE

In this section we describe why frequency analyses are not an option for the presented WB algorithm, meaning there is no need for Mixing Bijections(MBs) in our implementation. One key difference between Chow's AES WB and the presented Safer+ WB is that all our tables are 8-bit tables, and are not operating on nibbles. This means, that the frequency signatures must operate on all 8 bits. Yet one can easily notice, that in our case, for each given master key and $E-boxes$, one can choose a set of input and output permutations and functions $R_i^r(x_1,x_2)$, such that the $E-boxes$ constructed with given key, permutations and $R_i^r(x_1,x_2)$ match the given $E-boxes$. This can be achieved by using the same $R_i^r(x_1,x_2)$ that were used for construction of the initial $E-boxes$, because functions $T_i^r(x)$ are bijections. The input permutations can be chosen arbitrarily and the output permutations can simply be chosen as

$$\mathrm{f}_{2*i-1}^r\left(T_{i1}^r\left({\mathrm{g}_{2*i-1}^r}^{-1}(x_1),{\mathrm{g}_{2*i}^r}^{-1}(x_2)\right)\right)=E_{i1}^r(x_1,x_2),$$

$$\mathrm{f}_{2*i}^r\left(T_{i2}^r\left({\mathrm{g}_{2*i-1}^r}^{-1}(x_1),{\mathrm{g}_{2*i}^r}^{-1}(x_2)\right)\right)=E_{i2}^r(x_1,x_2),$$

where T_{i1}^r are built using the new master key, and E_{i1}^r and E_{i2}^r are the $E-boxes$ provided, built with the initial master key.

Because function $T_i^r(x)$ is a bijection of Z_{256}, the frequency signature of function $E_{i1}^r(x_1,x_2)$ or $E_{i2}^r(x_1,x_2)$ is always identical to frequency signature of function $R_i^r(x_1,x_2)$, and the attacker can learn limited amount of information about $R_i^r(x_1,x_2)$, but nothing about the secure key. This shows that MBs are not necessary for our WB implementation.

2.6 C++ IMPLEMENTATION

To demonstrate the algorithm and do the performance testing, a C++ library was implemented which has features of building encryption or decryption tables and applying encrypt/decrypt operations on a single block of data of 16 bytes.

Class **Safer_key_schedule** (Figure 2.9) is used for generating the keys for all rounds of SAFER+ from the given 16-byte master key using the modified key schedule algorithm described in section 3.2. The constructor of this class requires a 16-byte master key and function "get_keys" returns a pair of 16-byte round keys for any given round. SHA-256 implementation of CryptoPP library is used by the class.

Figure 2.10 shows a UML diagram with the classes used for generation of encryption and decryption WB tables from a given secret master key and input/output permutations.

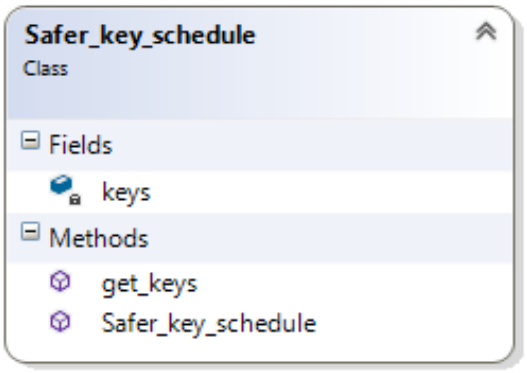

Figure 2.9: Class Safer_key_schedule.

The following are the main classes in the library, providing an API to the WB algorithm:

- **WB_enc_tables** - responsible for generation, access, storage and retrieval of encryption WB tables.
- **WB_dec_tables** - responsible for generation, access, storage and retrieval of decryption WB tables.
- **WB_tables** - contains shared data and methods for both tables.

Constructors of both **WB_enc_tables** and **WB_dec_tables** require the 16-byte master key and input/output permutations to generate the key schedule and build the WB tables for encryption and decryption, respectively. This includes the generation of $E-boxes$ and $2-PHT$ boxes for all the rounds.

The main functions for accessing the WB tables are:

- **get_round_1_ebox_output** - returns the 2-byte output of an $\mathrm{E}-\mathrm{box}$ of round 1 for any given 1-byte input. Looks up the value in local array "table_round_1", works in $O(1)$ time.
- **get_intermediate_round_ebox_output** - returns the 2-bytes of output for any given 2-byte input for all the $\mathrm{E}-\mathrm{boxes}$ for each round and byte. Looks up the value in local array "tables", works in $O(1)$ time.
- **get_2_pht_output** – returns 2-byte output for 2-byte input for a given $2-\mathrm{PHT}$ table for all rounds, layers and bytes. Looks up the value in local array "two_pht", works in $O(1)$ time.
- **constructor** – builds the WB tables based on the given master key and input/output permutations. The generation of white-boxes tables ~200ms, as all the $E-boxes$ and $2-PHT$ tables must be built.
- **tables_from_file** – loads WB tables from a given file. The tables must be created by providing the master key to constructor, and stored in the file using function tables_to_file.
- **tables_to_file** – stores the created WB tables into the given file in binary format. Later function "tables_from_file" can be used for loading the tables back into the class.

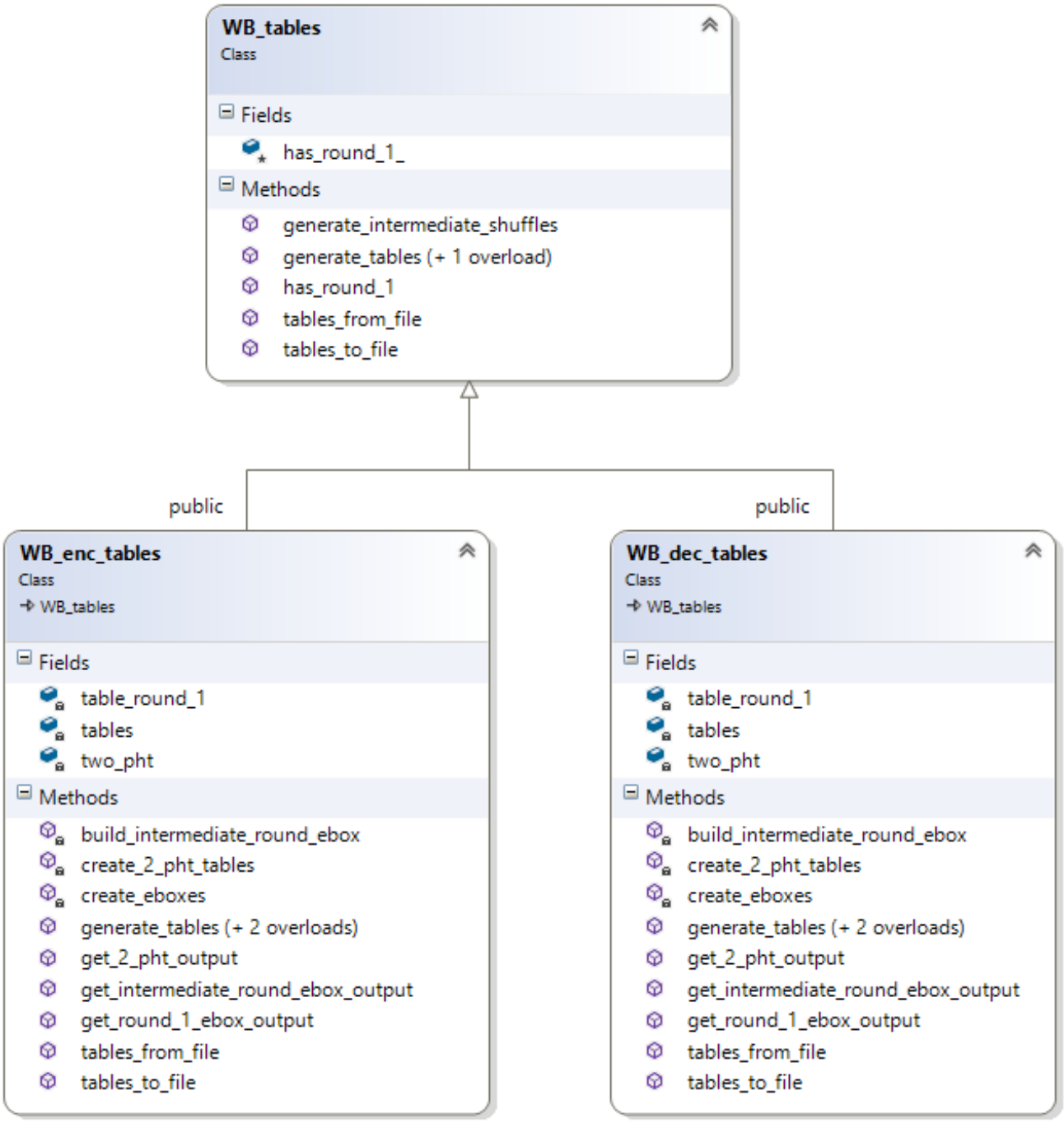

Figure 2.10: Classes for generation and access to white-box tables for encryption and decryption.

Class **WB_cryptor** shown in Figure 2.11 is the main class for applying encrypt/decrypt operations. It holds encryption and/or decryption WB tables and applies the encryption/decryption operation on a single integer, string or a blob of 16-byte binary data. Encrypted integers and strings are returned as a *base64* encoded string.

The API functions of class **WB_Cryptor** are:

- **encrypt** – encrypts the given 16-byte blob of binary data.
- **decrypt** – decrypts the given 16-byte blob of binary data.
- **encrypt_integer** – encrypts a single integer using encryption WB tables and base64 encodes the resulting ciphertext.
- **decrypt_integer** – decrypts a single base64 encoded integer, using the decryption white-boxes if present, the reverse of function encrypt_integer.
- **encrypt_string** – encrypts the given data encoded as a base64 string.
- **decrypt_string** – decrypts the given data encoded as a base64 string, the reverse operation of encrypt_string.

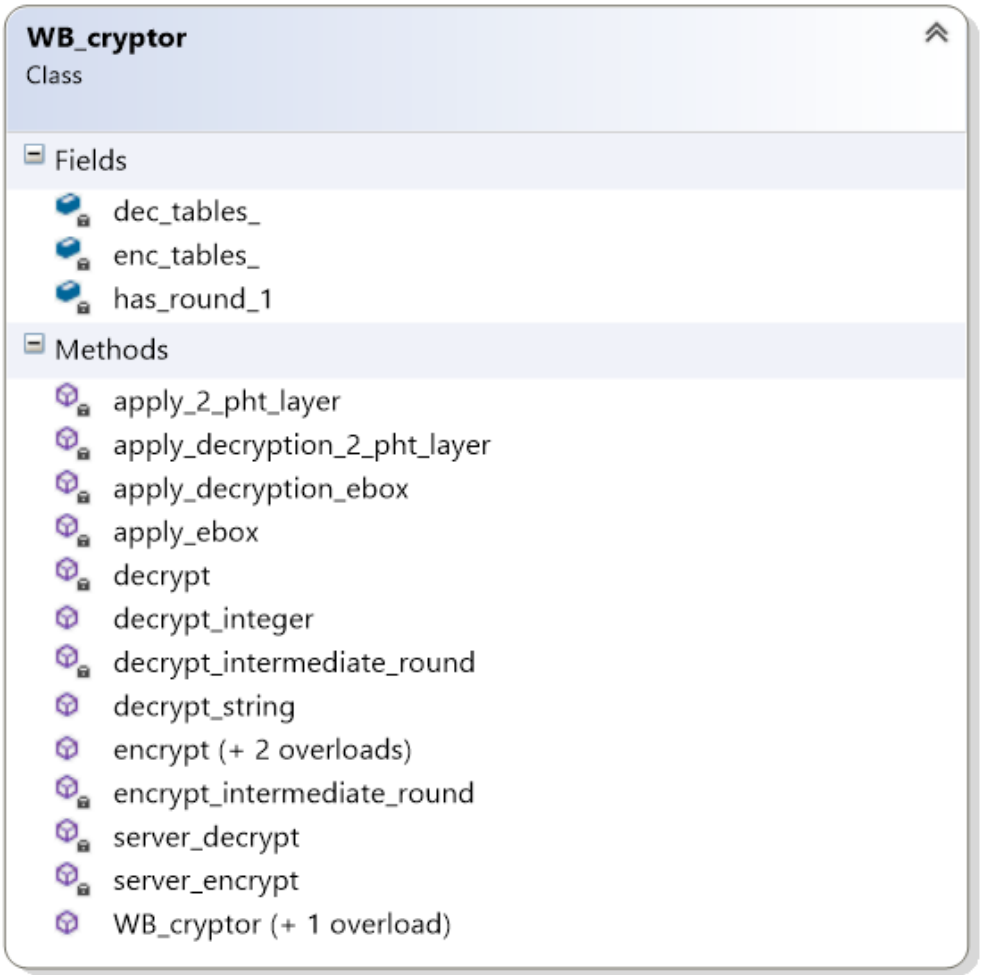

Figure 2.11: Class WB_cryptor.

2.7 PERFORMANCE TESTING

Performance tests of the presented WB encryption were made on Intel Core I5 CPU 1.6 GHZ processor. A single encryption operation on a 16-byte plaintext block took $6.5 * 10^{-5}$ seconds, while a single decryption took $6.7 * 10^{-5}$ seconds. This means the current implementation encrypts at ~5.07 Mbit/sec and decrypts at ~5.23 Mbit/sec. These performance values can be improved programmatically by writing a faster but less readable software code. Comparison of the WB implementation performance with its black-box and with RSA is shown in the table 2.7.1. As we can see WB encryption is about 6.5 times slower compared to its black-box variant. Yet for those applications, where using the black-box algorithm is not an option, the speed of WB presented is sufficient. As in some cases white-box can replace PK cryptography, the comparison with RSA-2048 and RSA-4096 is shown. As we can see that the algorithm is ~1800 times faster on encryption and ~80000 times faster on decryption. This means that replacement of PK cryptography with the presented WB algorithm will result to significant performance improvements.

Table 2.7.1: Time spent on a single encryption/decryption operation.

	Encryption speed (ms)	Decryption speed (ms)
White-box Safer+	$6.5 * 10^{-5}$	$6.7 * 10^{-5}$
Black-box Safer+	$1.02 * 10^{-5}$	$1.07 * 10^{-5}$
RSA-2048	0.12	5.46
RSA-4096	0.64	16.1

2.8 MEMORY REQUIREMENTS

Presented WB implementation of Safer+ requires 16 1-byte to 2-byte $E - boxes$ for round 1, 16 2-byte to 2-byte $E - boxes$ for each of rounds 2 to 6, 2-byte to 1 byte $E - box$ for round 7 and four layers of sixteen $2 - PHT$ tables for each of rounds 1 to 6.

Table 2.8.1: Size and quantity of white-box tables required.

Table name	Size of each table	Number of tables required	Total memory required
Round 1 $E-box$	512 byte	16	8 KB
Round 2-6 $E-box$	128 KB	16*5 = 80	10 MB
Round 7 $E-box$	64 KB	16	1 MB
$2-PHT$	128 KB	16*4*5=320	40 MB

So the total size of WB tables required for our implementation is around 51 MB. These boxes are significantly larger compared to their AES alternatives, yet this large size is required for security reasons. As in some applications WB tables are created and delivered to each user very rarely or just once, this huge size of the tables must not be an issue. Yet we understand that the size of WB tables can be an obstacle for using it in some memory-critical applications.

2.9 STATIC AND DYNAMIC TABLES FOR SECURE KEY UPDATING

In some applications, server needs to update WB tables with each file download, or several times during a given operation. Particularly, this is the case for DRM systems, which need to send the WB tables for each multimedia file, so the end user can decrypt and watch the content. In some cases, multiple WB tables can be used for watching different parts of the multimedia file, making the files harder to steal. Yet some of WB tables are not dependent on the secure key, and optimizations can be made to save the network traffic. Those tables which do not depend on the secure key, so do not need to get updated are called static tables, and those which contain the key are called dynamic tables. It is easy to notice, that in our implementation $E-boxes$ are dynamic tables and $2-PHT$ boxes are static. This lets the server update only $E-boxes$ with each file to be decrypted, and store $2-PHT$ boxes locally on the client device. In case the attacker tries to analyze the stored $2-PHT$ boxes separately from $E-boxes$, this kind of analyses will give no result, because the secure key is not stored in them. This way the server is able to frequently update the $E-boxes$ allowing shorter timeframe for the attacker to break them, also resulting to smaller network

bandwidth. This way, the content stream provider may create a separate white-box for each small fragment of the multimedia file, such that the attacker will gain access to only a small fragment of the whole file if one of the user white-boxes is broken.

So for our white-boxes, the total size of $E-boxes$ required is around 11 MB, and the total size of $2-PHT$ boxes is 40 MB. So in order to update the secure key, the server will be required to send the new $E-boxes$ only, which is sufficiently less that the total size of white-boxes.

2.10 CONCLUSION

A novel WB encryption scheme was introduced based on SAFER+ [**26**] algorithm. The structure of SAFER+ was used to build a WB algorithm with better security, compared to AES white-boxes. SAFER+ round operates on all 16 bytes of the input state and uses both bitwise addition and addition modulo 256 operations when adding the key to the state vector in the non-linear layer of the cipher. These give us additional advantages while trying to build a secure WB. The modification of the original key schedule, which complicates the key schedule algorithm making it irreversible, adds security to the described implementation, making it impossible for the attacker to easily compute round keys for another rounds, given the round key of some round. BGE attack is using the reversibility of AES key schedule, and the fact that the AES key is just added to the state, making the key addition an Affine Transformation. This is not true in our case, thus preventing application of the BGE attack for key extraction. Also the 16 byte WB algorithm runs with a 32-byte state vector, which is a novel approach in WB cryptography. This increases the size of WB tables and slows down the encryption, but lets us build a secure WB.

Security analyses were provided describing several possible attacks which do not work on the described cipher, including BGE attack, collision attacks, frequency analyses and other algebraic attacks.

Performance tests show that the provided C++ library encrypts at ~5.07 Mbit/sec and decrypts at ~5.23 Mbit/sec, which is slower compared to its black-box, but is order of magnitudes faster compared to modern PK cryptography algorithms. As WB cryptography

can be an alternative to PK cryptosystems in most of their applications, this creates a huge domain for the presented algorithm to be applied. Also the presented algorithm can be used in security critical software applications, which are running on unprotected devices, on which the black-box algorithms cannot be safely used.

CHAPTER 3

PUBLIC KEY ENCRYPTION ALGORITHM BASED ON PERMUTATION POLYNOMIALS

Standard PK algorithms are nowadays widely used for key sharing and many other popular applications, despite being known for slow execution speeds. A pioneering work describing DH key exchange by Diffie and Hellman [**27**] was presented in 1976, which is based on the discrete logarithm problem (DLP). In 1978 the RSA cryptosystem [**28**] was invented by Rivest, Shamir and Adleman, which is the standard public key (PK) encryption to use for most applications and is based on integer factorization problem. Another important development for public key cryptosystems was the invention of Elliptic Curve(later EC) cryptosystems [**71**].

An effective PK encryption system was presented in [**30**], which uses permutation polynomials and runs as a WB algorithm. The presented cryptosystem can replace the standard public key cryptosystems in all of their applications, resulting to significant speed improvements. Later several modifications were made to the algorithm to improve security and increase performance in [**33**]. In this chapter we will briefly review the cryptosystem [**30**] and discuss the modifications made.

3.1 BRIEF DESCRIPTION OF THE INITIAL ALGORITHM

Let $GF(q)$ be a finite field of characteristic p. For every permutation polynomial $f(x)$ over $GF(q)$, there exists a unique polynomial, $f^{-1}(x)$ over $GF(q)$ such that $f\big(f^{-1}(x)\big) = f^{-1}\big(f(x)\big) = x$ called the compositional inverse of $f(x)$.

Let $F(x) = \sum_{u=0}^{n} a_u x^u \in GF(2)$ be a primitive polynomial and $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ be its linearized 2-associate. Elements of the $GF(2^n)$ will be represented through a primitive polynomial $g(x)$ over $GF(2)$. Then $P(x)$ is a permutation polynomial and an algorithm for finding it's compositional inverse $P^{-1}(x)$ was presented in [**30**].

A primitive polynomial $g(x)$ of degree n over $GF(2)$ is used as the PK encryption private (secret) parameter and a permutation polynomial $P(x)$ as the public parameter.

a) Public key encryption: any message $m(x)$ of the length n as an input (plaintext) and evaluation of the polynomial

$$P\big(m(x)\big) = c(x)\ mod\ g(x)$$

as an output (ciphertext). The evaluation operation will be implemented via the WB evaluation without revealing the value of polynomial $g(x)$. The output of PK encryption will be $c'(x)$ based on WB tables described later.

b) Private key decryption: Given a ciphertext $c'(x)$ calculate $c(x)$. Compute

$$P^{-1}\big(c(x)\big) = P^{-1}\Big(P\big(m(x)\big)\Big) = m(x)\ mod\ g(x)\ \textbf{[30]}.$$

The WB evaluation is used to calculate the value of $c(x)$ without revealing the value of modulo reduction polynomial $g(x)$ in such a way, that the "owner" of the system can calculate $c(x)$ from it.

The evaluation procedure will be as follows: all possible residues

$$x^N = R_N(x)\ mod\ g(x)\,, \qquad (3.1.1)$$

$$where\ N = 2^i * (2k+1), i = \overline{1..128}, r = 2k+1\ for\ k = \overline{0..63}$$

are calculated. Then these values are biased by using random 64 secret polynomials $L_0, L_1, .., L_{63}$ based on another secret polynomial $L(x)$ which are only known to the "owner" of the system. All biased values for residues modulo polynomial $g(x)$ are provided to the public in the following manner:

$$B_N(x) = \big(R_N(x) \cdot L_0(x)\ mod\ L(x)\big) \oplus L_{k+1}(x)\,, \qquad (3.1.2)$$

$$for\ any\ N = 2^i * (2k+1),\ i = \overline{1,128},\ k = \overline{0,63}$$

Based on the above explanation an encoding procedure will be as follows: The user calculates an evaluation result of the polynomial $P\big(m(x)\big)$ without any reduction, takes the polynomial $R(x)$ that contains all terms of evaluation for the degrees not exceeding 127, and calculates the modulo two sum of nonzero terms $B_N(x)$ denoted by $\sum B_N(x)$ corresponding to nonzero terms of evaluation result exceeding $\mathrm{N} = 127$.

An encrypted message $c'(x)$ then contains two 16-byte vectors $R(x)$ and $\sum B_N(x)$ and another 8-byte vector $B = (b_0, b_1, .., b_{63})$, where $b_k = 0$ if the number of nonzero terms with the same value k in evaluation result is even and $b_k = 1$ otherwise for

$N = 2^i * (2k+1),\ k = \overline{0,63}$.

Decoding procedure by the "owner" of the system will be as follows: based on the value $B = (b_0, b_1, .., b_{63},)$ and vector $\sum B_N(x)$ the "owner" calculates:

$$R(x) \oplus \left(\sum B_N(x) \oplus \sum_{i=1}^{128} b_i * L_i(x) \right) \cdot L_0(x)^{-1} = c(x) \, mod \, L(x). \qquad (\mathbf{3.1.3})$$

Then having the values of $c(x)$ the value of $m(x)$ can be calculated:

$$P^{-1}\big(c(x)\big) = P^{-1}\Big(P\big(m(x)\big)\Big) = m(x) \, mod \, g(x) \, [\mathbf{30}].$$

3.2 IMPROVEMENTS MADE ON THE PUBLIC KEY ENCRYPTION ALGORITHM

A public polynomial $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ is used in the encryption algorithm described in [**30**].

Our analyses showed, that if $a_u = 0$ for all but one value v in $u = \overline{2..7}$ then the attacker may gain some information about the plain-text message $m(x)$ having the value of $R(x)$. For example if $P(x) = x^{2^{127}} + x^2 + x$ is used and if $m(x) = \sum_{i=0}^{126} a_i \cdot x_i$ then $R(x) = \sum_{i=0}^{63} a_i x^{2i}$, which means that the attacker will easily get the values of a_i for $i = \overline{0..63}$. A simple solution to this problem is to use values of $P(x)$ for which $a_u \neq 0$ for more than one u from $u = \overline{2..7}$.

A modification of the PK system follows, which will make usage of any value of $P(x)$ secure. We will describe the algorithm for $n = 128$, but it can be easily extended to be used for any value of n.

a) Public key encryption: any message $m(x)$ of the length n as an input (plaintext) and evaluation of the polynomial

$$P(m(x) * x^{128}) = c(x) \, mod \, g(x)\,, \qquad (\mathbf{3.2.1})$$

as an output (ciphertext). The evaluation operation will be implemented via a WB evaluation described later.

b) Private key decryption: Given a ciphertext $c'(x)$ calculate $c(x)$ using formula (3.1.2). Then compute

$$P^{-1}\big(c(x) * x^{2^{128-8}}\big) = (x) \;\; mod \, g(x), \qquad (3.2.2)$$

where $P^{-1}(x)$ is the compositional inverse of the polynomial $P(x)$. The equation holds, because $x^{2^{128-1}} = 1\ mod\ g(x)$.

After the modification there are no terms with degrees of $N < 128$ in the evaluation result of the polynomial $P(m(x) * x^{128})$, so there is no need for having the polynomial $R(x)$ any more.

Polynomials $L_0, L_1, .., L_{127}$ are generated, and values of $B_N(x)$ are provided publicly for any $N = 2^i * (2k + 1)$ where $i = \overline{1,128}$ and $k = \overline{0,127}$ according to formula (3.1.2).

An encrypted message $c'(x)$ then contains a 16-byte vector $\sum B_N(x)$ and another 16-byte vector $B = (b_0, b_1, .., b_{127})$, where $b_k = 0$ if the number of non-zero terms with the same value k in the evaluation result is even and $b_k = 1$ otherwise.

The decoding procedure by the "owner" of the system will be as follows: based on the value $c'(x) = \{\sum B_N(x),\ B = (b_0, b_1, .., b_{127})\}$ the "owner" calculates:

$$\left(\sum B_N(x) \oplus \sum_{i=1}^{128} b_i * L_i(x)\right) \cdot L_0(x)^{-1} = c(x)\ mod\ L(x). \qquad (3.2.3)$$

Then the "owner" of the system can decrypt the message by calculating the value of $m(x)$ using formula (4.3.2).

Now we need to prove the following lemma:

Lemma 3.2.1: The polynomial obtained by applying formula (3.2.2) on polynomial $c(x)$ obtained by using formula (3.2.3) gives a correct decryption result $m(x)$.

Proof: During the encryption, first the value of $c(x)$ is calculated according to formula (3.2.1). The reverse operation of this first step, done during decryption is the computation of $m(x)$ according to formula (3.2.2). It's easy to notice that

$$P^{-1}\left(c(x) * x^{2^{128-8}}\right) = P^{-1}\left(P(m(x) * x^{128}) * x^{2^{128-8}}\right) = m(x) * x^{128} * x^{2^{128-8}}$$
$$= m(x) * x^{2^{128-1}} = m(x)\ mod\ g(x),$$

i.e. formula (3.2.2) successfully applies the reverse operation of formula (3.2.1) on any polynomial.

Now all is left to prove is that the values of $c(x)$ computed with formula (3.2.3) is equal to $P(m(x) * x^{128})\ mod\ g(x)$. So let's say

$$P(m(x) * x^{128}) = c(x) = \sum_{i=1}^{|c(x)|} x^{c_i}.$$

The ciphertext is $\{\sum B_N(x), B\}, where\ B = (b_0, b_1, .., b_{127})$. In our case,

$$\sum B_N(x) = \sum_{i=1}^{|c(x)|} B_{c_i}(x).$$

According to formula (3.1.2):

$$\sum B_N(x) = \sum_{i=1}^{|c(x)|} \left(R_{c_i}(x) \cdot\ L_0(x)\ mod\ L(x)\right) \oplus\ L_{k+1}(x)\,.$$

According to (3.2.3):

$$\left(\sum B_N(x)\ \oplus \sum_{i=1}^{128} b_i *\ L_i(x)\right) \cdot L_0(x)^{-1}\ mod\ L(x) =$$

$$\left(\left(\sum_{i=1}^{|c(x)|} \left(R_{c_i}(x) \cdot\ L_0(x)\ mod\ L(x)\right) \oplus\ L_{k+1}(x)\right) \oplus \sum_{i=1}^{128} b_i *\ L_i(x)\right) \cdot L_0(x)^{-1}\ =$$

$$\left(\sum_{i=1}^{|c(x)|} \left(R_{c_i}(x) \cdot\ L_0(x)\ mod\ L(x)\right)\right) \cdot L_0(x)^{-1} = \sum_{i=1}^{|c(x)|} R_{c_i}(x)\ mod\ L(x)$$

So using formula (3.2.3) during decryption and doing modulo $L(x)$ operations we get the value of $\sum_{i=1}^{|c(x)|} R_{c_i}(x)$ as a result, and according to (3.1.1)

$$\sum_{i=1}^{|c(x)|} R_{c_i}(x) = c(x)\ mod\ g(x).$$

Thus Lemma **3.2.1** is true, and a message encrypted with the described WB algorithm can be correctly decrypted.

3.3 A NUMERICAL EXAMPLE

We will provide an example of size n = 8 for demonstration purposes. This numerical example can be used for testing purposes for other implementations of the described system. In normal settings $n = 128$ must be used, and $n = 8$ setting, obviously, provides no security.

Let $P(x) = x^{2^2} + x^{2^4} + x^{2^5} + x^{2^6} + x^{2^7}$, $g(x) = 1 + x + x^2 + x^7 + x^8$ and $m(x) = x^3 + x^5$.

$$P(m(x)\cdot x^8) = (x^{3+8}+x^{5+8})^{2^2} + (x^{3+8}+x^{5+8})^{2^4} + (x^{3+8}+x^{5+8})^{2^5} +$$
$$(x^{3+8}+x^{5+8})^{2^6} + (x^{3+8}+x^{5+8})^{2^7} = x^{11\cdot 2^2} + x^{13\cdot 2^2} + x^{11\cdot 2^4} + x^{13\cdot 2^4} + x^{11\cdot 2^5} + x^{13\cdot 2^5}$$
$$+x^{11\cdot 2^6} + x^{13\cdot 2^6} + x^{11\cdot 2^7} + x^{13\cdot 2^7}$$

We have that

$$\sum B_N(x) = B_{11\cdot 2^2}(x) \oplus B_{13\cdot 2^2}(x) \oplus B_{11\cdot 2^4}(x) \oplus B_{13\cdot 2^4}(x) \oplus B_{11\cdot 2^5}(x) \oplus B_{13\cdot 2^5}(x) \oplus$$
$$B_{11\cdot 2^6}(x) \oplus B_{13\cdot 2^6}(x) \oplus B_{11\cdot 2^7}(x) \oplus B_{13\cdot 2^7}(x),$$

where $B_N(x)$ are defined according to (3.1.1).

For this example, $L(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^7 + x^8$ was used, and the following random values for $L_i(x)$ were chosen:

$L_0(x)$	$1 + x + x^2 + x^7 + x^8$
$L_1(x)$	$1 + x + x^5 + x^7$
$L_2(x)$	$x^2 + x^3 + x^4 + x^6$
$L_3(x)$	$1 + x + x^2 + x^3 + x^5 + x^7$
$L_4(x)$	$x^2 + x^3 + x^7$
$L_5(x)$	$x^3 + x^4 + x^6 + x^7$
$L_6(x)$	$1 + x^6 + x^7$
$L_7(x)$	$1 + x^2 + x^7$
$L_8(x)$	$1 + x + x^2 + x^3 + x^5 + x^7$

Given these values of $L_i(x)$ and the value of $g(x)$, we get the following values for polynomials $B_N(x), N = 2^i * (2k+1)$ (only the values required in this particular sample encryption are listed):

$B_{11\cdot 2^2}(x)$	$1+x+\ x^2+x^3+x^5+\ x^6$
$B_{13\cdot 2^2}(x)$	$1+\ x^2+\ x^6$
$B_{11\cdot 2^4}(x)$	$1+x+\ x^4+x^5+\ x^6$
$B_{13\cdot 2^4}(x)$	$1+\ x^2+x^6+\ x^7$
$B_{11\cdot 2^5}(x)$	$x^2+x^4+\ x^7$
$B_{13\cdot 2^5}(x)$	$x^4+\ x^5$
$B_{11\cdot 2^6}(x)$	$1+x+\ x^4+x^5+x^6+\ x^7$
$B_{13\cdot 2^6}(x)$	$1+x+\ x^3+x^5+\ x^7$
$B_{11\cdot 2^7}(x)$	$x+\ x^2$
$B_{13\cdot 2^7}(x)$	$1+x^3+x^6+\ x^7$

The ciphertext will consist of the values $\sum B_N(x) = 1+x+x^2+x^3+x^5+x^7$ and vector$B$=(0,0,0,0,0,1,1,0), i.e. $c'(x) = \{1+x+x^2+x^3+x^5+x^7, (0,0,0,0,0,1,1,0)\}$.

For the decryption, we need to calculate the value of

$$\left(\sum B_N(x)\ \oplus L_6(x) \oplus L_7(x)\right) \cdot L_0(x)^{-1} = c(x)\ mod\ L(x).$$

$$\left(\sum B_N(x)\ \oplus L_6(x) \oplus L_7(x)\right) = \ 1+x+x^2+x^3+x^5+x^7+1+x^6+x^7+1+x^2+x^7$$

$$= 1+x+x^3\ +x^5+x^6\ +x^7\ .$$

It's easy to check that $L_0(x)^{-1} = \ 1+x^2+x^6\ +x^7$ and $c(x) = 1+x+x^2\ +x^3+x^6$.

Using the method described in [**30**], by solving a system of linear equations, we get the value of reverse permutation polynomial of $P(x)$,

$$P^{-1}(x) = \ 1+x+x^3+x^4\ +x^8.$$

The last step of decryption is calculation of the value of $m(x)$ using the following formula:

$$P^{-1}\left(c(x) *\ x^{2^{8-3}}\right) = m(x)\ mod\ g(x)$$

We have $x^{2^{8-3}}$ here, because $n = 8$ in our case, instead of the usual $n = 128$, and $8\ =\ 2^3$, so $x^{2^{8-3}}$ is the multiplicative inverse of $x^8\ mod\ g(x)$.

3.4 IMPLEMENTATION ASPECTS AND PERFORMANCE OPTIMIZATIONS

Encryption algorithm of the proposed system requires evaluation of the polynomial $P(m(x) \cdot x^{128})$ and then a modular reduction using WB implementation. The evaluation of $P(m(x) \cdot x^{128})$ will require to calculate the values of $m(x)^{2^i}$ for all $i = \overline{0,n}$, where 2^n is the order of $P(x)$. This will require n polynomial squaring operations.

In the rest of this dissertation let's agree to denote by $|\boldsymbol{P}(\boldsymbol{x})|$ the weight of polynomial $P(x)$. Then the evaluation of $P(m(x) \cdot x^{128})$ will have $|P(x)| * |m(x)|$ terms, so $|P(x)| * |m(x)|$ polynomials additions will be required to compute $\sum B_N(x)$. Also another $|P(x)| * |m(x)|$ operations will be required to compute the value of vector B.

Let's denote by $|B|$ the number of 1s in the binary vector B. During the decryption procedure, the calculation of $c(x)$ from $c'(x)$ will require $|B|$ modulo two additions to add the values of $L_i(x)$ to $\sum B_N(x)$ and one multiplication of polynomials to multiply with $L_0(x)^{-1}$ to compute the polynomial $c(x)$ according to formula **(3.2.3)**, if the value of $L_0(x)^{-1}$ is pre-computed once for all the messages and stored.

Final decryption operation will require to compute $P^{-1}\big(c(x)\big) \cdot x^{2^{120}} = m(x)\ mod\ g(x)$, where $P^{-1}(x)$ is a compositional inverse of polynomial $P(x)$. Calculation of $P^{-1}\big(c(x)\big)$ will require computing the values of $c(x)^{2^i}$ for all $i = \overline{0,127}$, which takes 128 polynomial squaring operations, and adding up the resulting polynomials which will require $|P^{-1}(x)|$ polynomial additions. Again, we assume that the value of $P^{-1}(x)$ is pre-computed and stored.

The memory required for storing the WB tables, i.e. the values of $B_N(x)$ will be 16 bytes per value for each $N = 2^i * (2k+1)$, $i = \overline{1,128}$, $k = \overline{0,63}$, resulting to overall 128*128*16 bytes, which is 256 KB of data.

Several performance optimizations were made to the presented algorithm. One can notice, that for $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ and $m(x) = \sum_{v=0}^{n} a_v x^v$,

$$P\big(m(x)\big) = \sum_{v=0}^{n} a_v P(x^v) \qquad (3.4.1)$$

In a similar way for $c(x) = \sum_{l=0}^{n} a_l x^l$,

$$P^{-1}\big(c(x)\big) = \sum_{l=0}^{n} a_l P^{-1}(x^l) \qquad (3.4.2)$$

Let's denote by $\boldsymbol{B_{P(x^v)}}(\boldsymbol{x})$ the sum of $B_N(x)$ for all degrees N in polynomial $P(x^v)$.

So from **(3.4.1)** one can notice, that given the values $B_{P(x^v)}(x)$, which are independent of the value of $m(x)$, one can easily notice that for a given polynomial $m(x) = \sum_{v=0}^{n} a_v x^v$,

$$\sum B_N(x) = \sum_{v=0}^{n} a_v \boldsymbol{B}_{\boldsymbol{P}(\boldsymbol{x}^v)}(\boldsymbol{x})$$

This means that if we pre-calculate and store the values of $\boldsymbol{B}_{\boldsymbol{P}(\boldsymbol{x}^v)}(\boldsymbol{x})$ for each $v = \overline{0,127}$ and the values of $P^{-1}(x^v)$ for each $l = \overline{0,127}$, this will require 128*16 bytes of additional memory for each of the encryption and decryption operations, but the performance will be increased dramatically.

We also pre-calculate the impact of $P(x^v)$ on array B for each $v = \overline{0,127}$, which requires another 128*16 bytes of memory.

After calculating and storing these values, instead of doing 128 polynomial squaring operations and $|P(x)| * |m(x)|$ modulo 2 additions for computing $\sum B_N(x)$ and B during the encryption, we'll do no squaring operations and just $|m(x)|$ modulo 2 additions to add up the values of $\boldsymbol{B}_{\boldsymbol{P}(\boldsymbol{x}^v)}(\boldsymbol{x})$. After this modification we will require $|P(x)| = 128$ times less operations in average.

In decryption we will skip doing the squaring operations and will just do sum up the values of $P^{-1}(x^l)$, doing just $|c(x)|$ polynomial additions.

Presented optimizations speed up both encryption and decryption operations by about 2.2 times at the cost of 4 KB additional memory and some pre-computations.

3.5 SECURITY ANALYSIS

The secret values in the cipher presented are polynomials $g(x), L(x), L_i(x)$. Public values are the values of $B_N(x)\ for\ all\ N = 2^i * (2k+1),\ i = \overline{1,128},\ k = \overline{0,127}$, computed by formula **(3.1.2)** and polynomial $P(x)$. Having these public polynomials, as well as the encryption result $c'(x)$ for some given plaintext polynomials $m(x)$, the attacker's objective is to obtain the values of $g(x), L(x)\ and\ L_i(x)$ to be able to decrypt any encrypted message $c'(x)$.

In order to decrypt message $c'(x)$, the attacker must be able to perform the following 2 operations:

- Calculate the value of $c(x)$ from $c'(x)$ using formula (3.2.3).
- Compute the value of $P^{-1}(x)$, so the value of $m(x)$ can be calculated using formula (3.2.2).

In order to be able to compute $c(x)$ from $c'(x)$ using formula (3.2.3), the values of $L_i(x)$ as well as the value of $L_0(x)^{-1} \, mod \, L(x)$ and $L(x)$ are required. In order to compute the value of $P^{-1}(x)$ from the value of $P(x)$, polynomial $g(x)$ is required.

So in order to decrypt a message, the attacker must deduce the values of the secret polynomials $g(x), L(x), and \; L_i(x) \; for \; all \; i = \overline{0,127}$.

Let us proof the following lemma:

Lemma 3.5.1: For any given set of $B_N(x) \; for \; any \; N = 2^i(2k+1), \; i = \overline{1,128}, \; k = \overline{0,127}$, it's computationally infeasible to deduce the values of $g(x), \; L(x) \; or \; L_i(x)$. Moreover, for each $g(x)$ there will be a value of $L(x)$ that would result to the same values of $B_N(x)$.

Proof: From formula **(3.1.2)** we get:

$$B_{128}(x) = \big(R_{128}(x) \cdot \; L_0(x) \; mod \; L(x)\big) \oplus \; L_1(x) \, ,$$
$$B_{256}(x) = \big(R_{256}(x) \cdot \; L_0(x) \; mod \; L(x)\big) \oplus \; L_1(x) \, ,$$
$$B_{512}(x) = \big(R_{512}(x) \cdot \; L_0(x) \; mod \; L(x)\big) \oplus \; L_1(x) \, .$$

From these equations we get:

$$B_{128}(x) \oplus \; B_{256}(x) = \big(R_{128}(x) \oplus R_{256}(x)\big) \cdot \; L_0(x) \; mod \; L(x) \, ,$$
$$B_{128}(x) \oplus \; B_{512}(x) = \big(R_{128}(x) \oplus R_{512}(x)\big) \cdot \; L_0(x) \; mod \; L(x).$$

From these two equations we get:

$$\frac{B_{128}(x) \oplus \; B_{256}(x)}{B_{128}(x) \oplus \; B_{512}(x)} = \frac{R_{128}(x) \oplus R_{256}(x)}{R_{128}(x) \oplus R_{512}(x)} \; mod \; L(x). \qquad (\mathbf{3.5.1})$$

Though the division operations here are done modulo $L(x)$, while $R_{128}(x) = x^{128} \; mod \; g(x)$. So if the value of $g(x)$ is given, one can compute the values of $R_i(x)$, then from formula (3.5.1) can be written as

$$\big(B_{128}(x) \oplus \; B_{256}(x)\big) * (R_{128}(x) \oplus R_{512}(x))$$
$$= (B_{128}(x) \oplus \; B_{512}(x)) * (R_{128}(x) \oplus R_{256}(x)) \; mod \; L(x) \, ,$$

Which is the same as

$$\left(B_{128}(x) \oplus B_{256}(x)\right) * \left(R_{128}(x) \oplus R_{512}(x)\right) - \left(B_{128}(x) \oplus B_{512}(x)\right) * \left(R_{128}(x) \oplus R_{256}(x)\right) = 0 \bmod L(x) \quad (3.5.2)$$

So if all the values on the left side of equation are given, one can compute it and knowing the degree of $L(x)$ find a divisor of the required degree. Or, alternatively another similar equation can be written with $N = 1024$, and having 2 polynomials, both divisible by $L(x)$, we can simply compute the GCD of the given two polynomials.

So for any value of $g(x)$ there exists a value of $L(x)$. On the other hand if the value of $L(x)$ is known, one can compute the value of $g(x)$ from formula $(\mathbf{3.5.1})$**.** We can rewrite it as

$$\frac{B_{128}(x) \oplus B_{256}(x)}{B_{128}(x) \oplus B_{512}(x)} \bmod L(x) = \frac{(x^{128} \oplus x^{256}) \bmod g(x)}{(x^{128} \oplus x^{512}) \bmod g(x)} \bmod L(x),$$

From which we get:

$$\frac{B_{128}(x) \oplus B_{256}(x)}{B_{128}(x) \oplus B_{512}(x)} * (x^{128} \oplus x^{512}) \bmod L(x) - (x^{128} \oplus x^{256}) = 0 \bmod g(x).$$

As the polynomial on the left side of the equation can be computed, given the value of $L(x)$, one can get the value of $g(x)$, because $g(x)$ is a divisor of that polynomial.

So we proved that for any given value of $L(x)$, the polynomial $g(x)$ can be computed, and for any given value of $g(x)$, the value of $L(x)$ can be computed. Yet if none of them is known, the equations got from the values of $B_N(x)$ don't provide enough information for getting any of these values.

Lemma 3.5.2: The computation of $P^{-1}(x)$ from $P(x)$ without the knowledge of secret polynomial $g(x)$ is not possible, and for each value of $g(x)$ there is a corresponding value of $P^{-1}(x)$.

Proof: An algorithm for building the compositional inverse of polynomial $P(x)$ for any value of $g(x)$ was presented in [**30**]. A set of linear equation is being solved, and it's shown that this set of equations must always have a solution. So for every value of $g(x)$ the corresponding value of $P^{-1}(x)$ exists. This means, that without the knowledge of $g(x)$ it's impossible to compute $P^{-1}(x)$.

So based on the proved lemmas 3.5.1 and 3.5.2 we can say, that the attacker, having the values of $B_N(x)$ and $P(x)$, which are publicly provided, will be unable to compute the values of $g(x),\ L(x), L_i(x)\ or\ P^{-1}(x)$, which are necessary for decryption. This proves the security of the presented cryptosystem.

3.6 C++ IMPLEMENTATION

A C++ library was implemented for performance testing of the presented algorithm. CryptoPP [**31**] library was used for RSA [**28**] implementation and for mathematical primitive operations. The main class is class **Cryptor** (Figure 3.1), with the following public methods:

- **wbox_encrypt** - encrypts the given message with the WB algorithm, returning the value of $c'(x) = \{\sum B_N(x),\ B = (b_0, b_1, .., b_{127})\}$ for any given message $m(x)$.
- **blackbox_decrypt** - decrypts the message with black-box algorithm, i.e. computes the value of $c(x)$ from $c'(x)$, then uses formula $(\mathbf{3.2.2})$ to compute the value of $m(x)$.

The values of $g(x)$, $L(x)$, $P(x)$ and $L_i(x)\ for\ all\ i = \overline{0,127}$ must be provided through the constructor. Polynomial $P^{-1}(x)$ is computed using Gauss Elimination method in the constructor and is locally stored in the class for all subsequent decryption operations.

Class **Cryptor** can be used to generate and use WB tables, apply encryption and decryption operations. It possesses the values of polynomials $g(x),\ P(x), P^{-1}(x), L(x), L_i(x), L_0^{-1}(x)$.

Class **EuclideanMultiplicativeInverseCalculator** is used for computing the value of polynomial $L_0^{-1}(x)$ using the Euclidean algorithm.

Class **CompositionalInverseCalculator** is used for computing the value of $P^{-1}(x)$ for a given $P(x)$ by solving a system of linear equations using Gaussian Elimination, as described in [**30**].

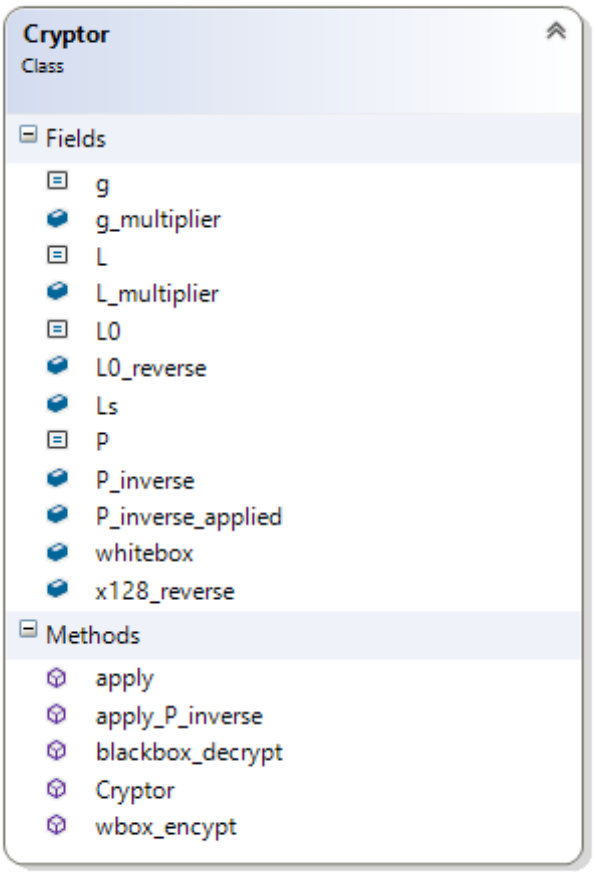

Figure 3.1: Class Cryptor.

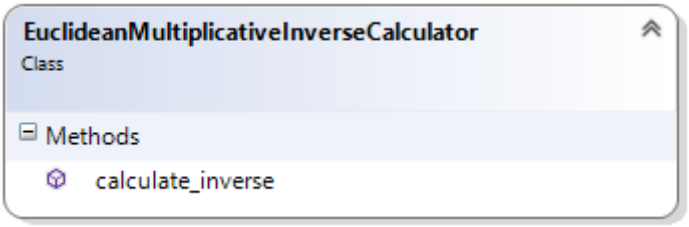

Figure 3.2: Class EuclideanMultiplicativeInverseCalculator.

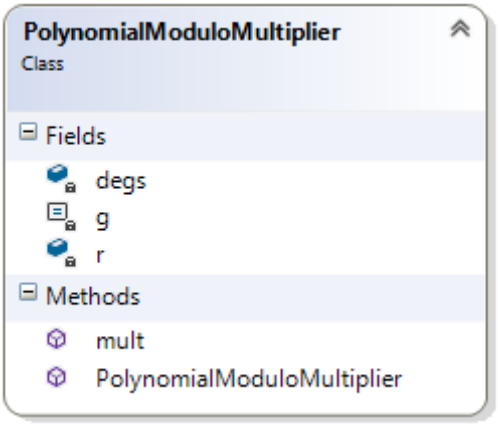

Figure 3.3: Class PolynomialModuloMultiplier.

Class **PolynomialModuloMultiplier** is used for multiplying any 2 polynomials modulo a fixed polynomial $g(x)$. In order to speed up the multiplications, it creates and stores the values of $R_N(x) = x^N \, mod \, g(x) \, for \, N = \overline{0,256}$ during construction, so the modulo reductions of the resulting polynomial can be calculated fast. The construction of these values takes 256 modulo $g(x)$ reductions. Yet the reductions are pretty fast, since the values of $R_N(x)$ are computed with the following pseudo-code:

$R_0(x) = 1$
FOR N := 1 to 256 do
  $R_N(x) := R_{N-1}(x) * x$
  IF degree of $R_N(x)$ = degree of $g(x)$ THEN
    $R_N(x) := R_N(x) \oplus g(x)$
  ENDIF
ENDFOR

This way, if the degree of $g(x)$ is 128, we'll do just 128 polynomial additions to compute all necessary values of $R_N(x)$.

Two instances of this class exist in the **Cryptor** class, for operations modulo $g(x)$ and $L(x)$, respectively. As most of the operations in the cryptosystem are made modulo $g(x)$ or modulo $L(x)$ the total system performs better with these pre-computations.

Class **Polynomial** represents a single polynomial in the system and provides multiplication, addition and division operations. As the most frequent operation is modulo 2 additions, the class is optimized to do it fast. A polynomial of degree <128 is represented as a mask of 128 bits and stored as a vector of four 32-bit integers. So having 2 such polynomials, we can apply XOR operation to these 4 32-bit integers and we'll get the mask representation of modulo two sum of the given pair of polynomials. This makes the speed of one polynomial addition operation equal to 4 integer XOR operations.

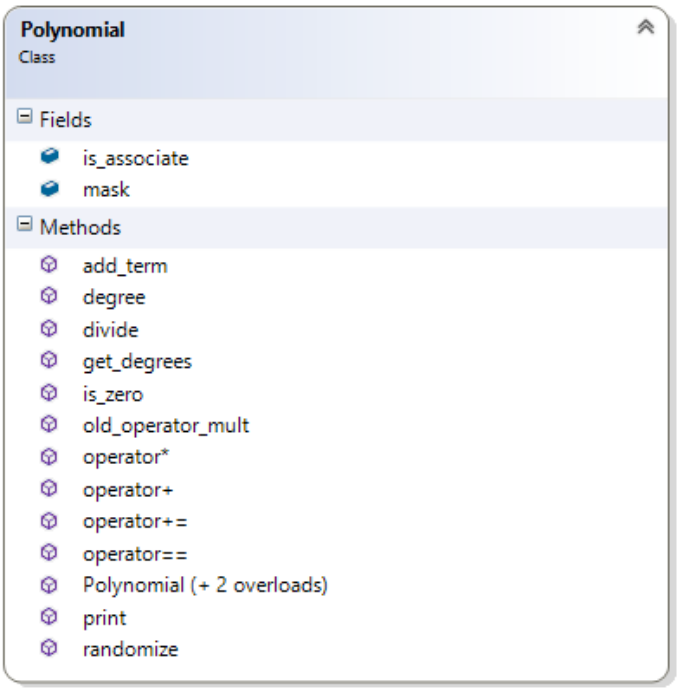

Figure 3.4: Class Polynomial.

3.7 PERFORMANCE TESTING

Performance tests of the presented PK encryption scheme were ran on Intel Core i5 1.6 GHZ processor. The CryptoPP [**31**] library's RSA-2048 and RSA-4096 implementations were used for benchmarking. The results of performance tests are provided in table 3.7.1. As we can see the presented algorithm is ~3.75 times faster on encryption and ~133 times faster on decryption compared to RSA-2048. This is a significant performance improvement, and shows that replacement of RSA with the presented WB algorithm in any of its applications will result to a significant improvement in execution speed.

Table 3.7.1: Time spent on a single encryption/decryption operation in milliseconds.

	Encryption speed	Decryption speed
Initial WB algorithm	0.094	0.12
Improved WB algorithm	0.032	0.041
RSA-2048	0.12	5.46
RSA-4096	0.64	16.1

3.8 CONCLUSION

An improved version of the PK encryption scheme based on permutation polynomials described in [**30**] was presented. The algorithm uses WB reduction during encryption. The improvements were made for security improvement and performance optimization. Security analyses were briefly discussed.

The modified implementation stores several pre-computed values, which results to significant speed improvements at the price of increase in memory requirements by just 4 KB. Performance analyses of the improved system were presented showing that the algorithm runs ~3.75 times faster on encryption and ~133 times faster on decryption compared to RSA-2048. This proves that the algorithm is a good alternative to standard PK algorithms like RSA for all of its applications. Implementation aspects were described, and several implementation optimizations were presented.

CHAPTER 4

INTEGRATION OF THE WHITE-BOX ALGORITHM INTO SKYCRYPTOR'S SECURE SEARCH ENGINE

Nowadays many companies and individual users store their data in cloud storage systems providers like Dropbox [**2**], Google Drive [**3**], Box [**4**] and Microsoft OneDrive [**5**]. These storage providers have full access to the data they store. This is a big security issues for some companies [**72**], so the solution for them is to use third-party tools like Sookasa [**9**], Boxcryptor [**11**], nCryptedCloud [**10**] in order to encrypt their data on the fly before storing it on the cloud storage.

SkyCryptor [**12**] is a cloud encryption service provider which provides zero-knowledge security to users who want to store their files on cloud storages in an encrypted form. SkyCryptor relies on proxy re-encryption scheme [**73**], [**74**], [**75**], [**76**], and acts as a semi-trusted proxy server for key storage and file sharing. Each user has public and private keys used for encrypting the AES key of the file. So each file is first encrypted by a randomly generated key using AES [**14**], then the key itself is encrypted with the user's public key and is stored with the encrypted file on the cloud storage provider's site. SkyCryptor allows file sharing, in which case if user A wants to share a file with user B, he/she creates a special proxy re-encryption key and stores it on the servers of SkyCryptor. Then user B can request the re-encryption key, and using that decrypt the encrypted file key, which can be used to AES decrypt the file itself. The next fundamental advantage of SkyCryptor is the fast SE algorithm used. Both file sharing and search use zero knowledge algorithms.

4.1 USAGE OF WHITE-BOX ALGORITHM IN ENCRYPTED SEARCH IMPLEMENTATION

As SkyCryptor provides zero knowledge security, it must not learn anything about the user's files' contents, even after analyzing the search outcomes. All the communication between SkyCryptor and the user must stay encrypted, and SkyCryptor must not see or possess any information which will allow learning any information about any user's files. For this reason,

all the files are encrypted and decrypted only on the users' devices and SkyCryptor stores only re-encryption keys, which do not provide sufficient information to decrypt the files. Secure file indexes are also built on the user side.

SkyCryptor supports the following operations:

- **Account creation**
  - All users create accounts, RSA key pair and WB tables are generated on creation.
  - WB tables are encrypted using the RSA public key and are stored on the SkyCryptor's servers.
  - Every time a user adds a new device, the tables are downloaded from SkyCryptor servers and decrypted using user's private key.
- **File addition**
  - The new file with id f_{id} is encrypted by a generated 256-bit AES key $K_{f_{id}}$
  - $K_{f_{id}}$ is encrypted by the user's public key K_{pub}
  - User uploads the encrypted file and its AES key to the online storage provider of the his/her choice.
  - A list of keywords stems W_{f_i} is built with a keyword extraction algorithm followed by a stemming algorithm [**77**].
  - Each keyword $w_i \epsilon W_{f_i}$ is encrypted using the WB implementation of SAFER+ block cipher.
  - The list of encrypted keywords $\{E_{WB}(w_i),\ w_i \epsilon W\}$ is uploaded to SkyCryptor
  - SkyCryptor builds forward and inverted indexes using these values. The forward index stores a list of keywords per f_{id}, and the inverted index stores a list of f_{id} which contain the given encrypted keyword $E_{WB}(w)$.

f_{id}	Encrypted keywords
1	$E_{WB}(w_1), E_{WB}(w_8), E_{WB}(w_9)$
2	$E_{WB}(w_1), E_{WB}(w_6), E_{WB}(w_4)$
...	...
n	$E_{WB}(w_1), E_{WB}(w_2)$

(a) Forward index

Encrypted keyword	List of f_{id}
$E_{WB}(w_1)$	$1,2,n$
$E_{WB}(w_2)$	n
...	...
$E_{WB}(w_m)$	$3,15$

(b) Inverted index

Figure 4.1: Forward and reverse indexes.

- **Search**
  - User enters the search query, consisting of a list of keywords.
  - A stemming algorithm [**77**] is ran on each keyword, to extract the stems from each keyword.
  - User uses the same WB tables to encrypt the resulting list of search keyword stems.
  - The list of encrypted keywords is sent to SkyCryptor.
  - SkyCryptor looks up the inverted index for file ids for given encrypted keywords, then intersects the file id lists to compute the list of encrypted file ids matching all the search keywords, then sends this list back to the user.
  - User uses the file ids to download the necessary files from the cloud storage provider where the files were stored. The encrypted AES keys are downloaded with each file.
  - The AES key is decrypted using the user's private key K_{pr}.
  - Decrypted AES key $K_{f_{id}}$ is used to decrypt the file itself.
- **File modification**
  - User edits the file, encrypts it the same way and uploads to the storage provider with the encrypted key.
  - User generates the new list of encrypted keywords, sends it to SkyCryptor server.
  - SkyCryptor uses the forward index to remove the previous keywords from both indexes, then adds the new encrypted keyword list received from the user to both indexes.
- **File deletion**

- User deletes a file from the cloud storage provider.
- User sends a digitally signed deletion request to SkyCryptor.
- SkyCryptor uses the forward index to find a list of encrypted keywords $\{E_{WB}(w_i), w_i \epsilon W_{f_i}\}$ which are present in the file being deleted then removes the corresponding entries from both forward and inverted indexes.

- **File sharing** – User Bob wants to share a file with Alice.
  - Bob downloads the public key of Alice from SkyCryptor.
  - A proxy re-encryption scheme is used to generate a proxy key, which will allow Alice to decrypt a single file owned by Bob. The proxy key is also built in a special way, using our WB algorithm.
  - The proxy re-encryption key is then uploaded to SkyCryptor, providing access to the file for Alice from all of her devices.
  - Alice request the re-encryption key from SkyCryptor, downloads the file she wants to access, and decrypts the file AES key using the re-encryption key and her private key.

As we can see, none of the operations performed by SkyCryptor deals with any unencrypted data of a user. WB algorithm here replaces PK cryptography, and lets the file addition operation run orders of magnitudes faster. This allows SkyCryptor to encrypt big number of keywords on file addition operation in seconds, as well as effectively share files between users and re-build the forward and inverted indexes when necessary.

CONCLUSION

White-box algorithms, unlike their "black-box" alternatives, are designed to run on vulnerable devices (such as PCs, smartphones or tablets), where the attacker can use tools like IDA Pro, debuggers and emulators to gain full access to all the internal values generated during the software execution and full control over the execution routine. We say that software running on such devices operates in the WB attack context and conventional black-box ciphers don't provide the necessary level of security for these devices. WB algorithms use look-up tables for encryption, which are constructed in such clever ways, that the attacker, having full access to all the tables and observing or modifying the encryption routine is unable to extract the secure cryptographic key or perform a decryption operation. This creates a wide range of applications for WB algorithms, from Digital Rights Management systems, to Oblivious Transfer and SE algorithms. WB algorithms are also used as a fast alternative to PK encryption algorithms.

A pioneering work by Chow and others in 2002 [**15**], presenting the first WB implementation of AES block cipher, gave birth to WB cryptography. Yet Chow's implementation were successfully attacked by the BGE attack [**16**]. Later, several attempts were made to improve Chow's white-boxes to protect it from the BGE attack. Yet all the subsequent implementations [**19**], [**43**], [**17**] were successfully broken by the modifications of BGE attack [**22**], [**18**], [**44**]. Later a collision attack was introduced by T.Lepoint and M. Rivain [**23**], which runs much faster and is easier to understand and implement compared to the BGE attack. A review of all the mentioned WB implementations was presented. The BGE attack [**16**] and the collision attack by T.Lepoint and M. Rivain [**23**] were briefly described.

In this dissertation we presented a novel WB encryption scheme based on SAFER+ block cipher [**26**]. The algorithm takes advantage of some details of SAFER+ and uses different techniques to defend from the known attacks such as the BGE attack [**16**]. The key schedule was modified to make it irreversible. The outputs of all the tables were randomly split into 2 parts, which significantly complicates any algebraic attack on the tables. General security analyses were presented, including security against the BGE attack, some collision and

frequency analyses attacks, which work on weakened variants of the presented cipher, but are shown to be not applicable for the presented implementation. A C++ library was implemented as a proof of concept and for performance testing. Performance test results were presented, showing sufficient speed of the algorithm. As all the WB implementations of AES were successfully broken, the presented algorithm is the only WB implementation of a block cipher which can be safely used. This creates large number of applications for the presented algorithm.

The second part of this research presented a modification of a WB encryption algorithm based on permutation polynomials [**30**]. Several modifications were made to improve the security and performance. The algorithm can safely replace conventional PK encryption schemes such as RSA [**28**] and Eliptic Curve Cryptosystems [**29**] resulting to a significant performance improvement. Benchmarks of the implementation were provided proving its high performance compared to rival algorithms. The optimized algorithm is ~3.75 times faster on encryption and ~133 times faster on decryption compared to RSA-2048. This makes it a good alternative to the classic public key cryptosystems and switching to the presented scheme will result to significant performance improvements. Security analyses were provided, describing several attacks which cannot be applied on the presented algorithm.

The presented algorithms were integrated into SkyCryptor's [**12**] secure search engine, to replace all the usages of conventional PK cryptosystems and provide better performance for the search and file sharing operations. SkyCryptor is a cloud encryption service provider which provides zero-knowledge security to users who want to store encrypted files on cloud storages like Box [**4**], Google Drive [**3**], Dropbox [**2**] and Microsoft OneDrive [**5**]. SkyCryptor relies on proxy re-encryption scheme [**73**], [**74**], [**75**] and acts as a semi-trusted proxy server for key storage and file sharing. Two vital features of SkyCryptor are secure keyword search over encrypted user files and file sharing. Without the ability to search, the users would be unable to manage huge amounts of data they want to store on the cloud, and without the ability to share the files, the users would be required to download and re-

encrypted each file which they need to send to another person. Proxy re-encryption schemes allow implementing both operations, but rely on slow PK cryptography algorithms. SkyCryptor uses proxy re-encryption and keyword encryption based on our WB algorithms. The replacement of PK operations with our WB algorithms speeded up the operations of addition and modification of files, as encryption of many keywords was necessary for these operations. The performance of file sharing and re-encryption key generation was also improved. This resulted into a practically usable fast searchable encryption system for users who want to store their files on the cloud in an encrypted form.

REFERENCES

[1] (2006) Amazon S3, Cloud Computing Storage for Files, Images, Videos. [Online]. https://aws.amazon.com/s3/

[2] (2007) Dropbox: file hosting service. [Online]. https://www.dropbox.com/

[3] (2012) Google Drive: Shared disk. [Online]. https://drive.google.com/

[4] (2005) Box - Secure Content & Online File Sharing for Businesses. [Online]. https://www.box.com/

[5] "Microsoft OneDrive: file hosting service," *[Online]. https://onedrive.live.com/*, 2007.

[6] S Rajesh, S Swapna, and P.S. Reddy, "Data as a Service (Daas) in Cloud Computing," *Global Journal of Computer Science and Technology: Cloud & Distributed, vol. 12, no. 11*, 2012.

[7] Iyer B., Mehrotra S. Hacigümüs H., "Providing Database as a Service," *In Proc. of 18th International Conference on Data Engineering (ICDE)*, 2002.

[8] Software & Infromation Industry Association (www.siia.net), "Software as a Service," *Strategic Backgrounder. Washington D. C.*, 2001.

[9] "Sookasa | Fully integrated CASB and cloud security provider," *[Online]. https://www.sookasa.com/*, 2012.

[10] "nCryptedCloud | Enterprise Grade Secure Cloud File Sharing and Collaboaration," *[Online]. https://www.encryptedcloud.com/*, 2012.

[11] "Encryption Software to Secure Files in the Cloud | Boxcryptor," *[Online]. https://www.boxcryptor.com*, 2011.

[12] "SkyCryptor: Cloud Securing Gateway," *[Online]. https://www.skycryptor.com/*, 2016.

[13] Gurgen Khachatrian, Andriy Oliynyk and Mykola Raievskyi Aram Jivanyan, "Efficient Oblivious Transfer Protocols based on White-Box Cryptography," *https://eprint.iacr.org/2016/734.pdf*, 2016.

[14] "Advanced Encryption Standard," *FIPS 197. Federal Information Processing Standards Publication 197, U.S. Department Of Commerce National Institute of*

Standards and Technology. Available from http://www.csrc.nist.gov/publications/fips/, 2001.

[15] Chow S., Eisen P., Johnson H., van Oorschot. P.C., "White-Box Cryptography and an AES Implementation," *In 9th Annual Workshop on Selected Areas in Cryptography (SAC)*, pp. 15-16, 2002.

[16] Henri Gilbert, Charaf Ech-Chatbi Olivier Billet, "Cryptanalysis of a WB AES Implementation," *In Selected Areas in Cryptography(SAC 2004)*, pp. 227-240, 2004.

[17] Yaying Xiao and Xuejia Lai, "A secure implementation of White-Box AES," *In Computer Science and its Applications, 2009. CSA'09. 2ndInternational Conference on. IEEE*, pp. 1–6, 2009.

[18] Peter Roelse, and Bart Preneel Yoni Mulder, "Cryptanalysis of the Xiao – Lai White-box AES implementation," *In Lars R. Knudsen and Huapeng Wu, editors, Selected Areas in Cryptography, volume7707 of Lecture Notes in Computer Science, pages 34–49. SpringerBerlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-35999-6_3*, 2013.

[19] Mohamed Karroumi, "Protecting WB AES with Dual Ciphers," *In Kyung-Hyune Rhee and DaeHun Nyang, editors, Information Security and Cryptology - ICISC 2010, volume 6829 of Lecture Notes in Computer Science, pages 278–291. Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-24209-0_19*, 2011.

[20] E., Biham, E. Barkan, "The book of Rijndaels," *Cryptology ePrint Archive, Report 2002/158, http://eprint.iacr.org/2002/158*, 2002.

[21] A., De Canni` ere, C., Braeken, A., Preneel, B. Biryukov, "A toolbox for cryptanalysis: Linear and affine equivalence algorithms," *In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, Springer, Heidelberg*, pp. 33–50, 2003.

[22] Peter Roelse, and Bart Preneel Yoni De Mulder, "Revisiting the BGE Attack on a WB AES Implementation," *http://eprint.iacr.org/2013/450.pdf*, 2013.

[23] T.Lepoint; M. Rivain, "Another Nail in the coffin of White–box AES implementations," *URL: http://eprint.iacr.org/2013/455*, 2013.

[24] Joppe W. Bos and Charles Hubain and Wil Michiels and Philippe Teuwen, "Differential Computation Analysis: Hiding your White-Box Designs is Not Enough," *Cryptology ePrint Archive: Report 2015/753*, 2015.

[25] E. Biham and A. Shamir, "Power analysis of the key scheduling of the AES candidates.," *In Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, pp. 115–121, 1999.

[26] J.Massey; G.Khachatrian; M.Kuregian, "Nomination of SAFER+ as a Candidate Algorithm for Advanced Encryption Standard (AES)," *Represented at the first AES conference, Ventura, USA, August20-25*, 1998.

[27] W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory, Vol. IT-22*, pp. 644--654, November 1976.

[28] A. Shamir, L. Adleman R. Rivest, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM 21 (2), 120--126*, 1978.

[29] Neal Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation, 48*, pp. 203 – 209, 1987.

[30] M. Kuregian G. Khachatrian, "Permutation polynomials and a new public key encryption," *accepted for publication in Discrete Applied Mathematics journal, 9 pages*, February 2015.

[31] *C++ class library of cryptographic schemes.*: [ONLINE] https://www.cryptopp.com/, 1995.

[32] Khachatryan, Oliynik Jivanyan, "Delegable (Multi-User) Searchable Encryption Scheme: Construction and Security Analysis," *Private Conversation*.

[33] Gurgen Khachatrian, Martun Karapetyan, "On a public key encryption algorithm based on Permutation Polynomials and performance analyses," *International Journal Information Theories and Applications (ITHEA), Vol. 23, Number 1, pp.34-38*, 2016.

[34] Gurgen Khachatrian, Martun Karapetyan, "White-box Encryption Algorithm based on SAFER+," *in proceedings of international workshop on Information theory and Data*

Science, "From Information Age to Big Data Era", Yerevan, Armenia, October 3-5, pp. 77-88, 2016.

[35] Karapetyan Martun, "Review of White-box Implementations of AES Block," *Mathematical Problems of Computer Science 46*, pp. 26–36, 2016.

[36] National Institute of Standards and Technology, "Data Encryption Standard," in *Federal Information Processing Standard (FIPS), Publication 46,U.S. Department of Commerce*. Washington D.C., 1977, pp. 250-259.

[37] P. Eisen, H. Johnson, P.C. van Oorschot S. Chow, "A White-box DES Implementation for DRM Applications.," *In Proceedings of 2nd ACM Workshop on Digital Rights Management (DRM 2002), volume 2696 of Lecture Notes in Computer Science*, pp. 1-15, 2002.

[38] A.Shamir E.Biham, "Differential cryptanalysis against DES-like cryptosystems," *Advances in Cryptology – "CRYPTO-90", Lecture notes in Computer Science N 537, Springer*, pp. 212-241, 1990.

[39] M. Matsui, "Linear Cryptanalysis Method for DES cipher," *Advances in Cryptology-Eurocrypt "93" Lecture Notes in Computer Science, Springer N 765, Springer*, pp. 386-397, 1994.

[40] Hamilton E. Link and William D. Neumann, "Clarifying Obfuscation: Improving the Security of White-Box Encoding," *In ITCC*, vol. 1, pp. 679–684, 2005.

[41] Jean-Michel Masereel, and Michael Quisquater Louis Goubin, "Cryptanalysis of white box DES," *In Carlisle Adams, Ali Miri, and Michael Wiener, editors, SAC 2007. Springer Berlin Heidelberg*, vol. 4876, pp. 278–295, 2007.

[42] Wil Michiels, Paul Gorissen, and Bart Preneel Brecht Wyseur, "Cryptanalysis of white-box DES implementations with arbitrary external encodings," *In Carlisle Adams, Ali Miri, and Michael Wiener, editors, SAC 2007. Springer Berlin Heidelberg, 2007.*, vol. 4876, pp. 264–277.

[43] Herve Chabanne, and Emmanuelle Dottax Julien Bringer, "White-box cryptography: Another attempt," *ONLINE: https://eprint.iacr.org/2006/468.pdf*, 2006.

[44] Brecht Wyseur, and Bart Preneel Yoni Mulder, "Cryptanalysis of a perturbated White-box AES implementation.," *In Guang Gong and KishanChand Gupta, editors, Progress in Cryptology - INDOCRYPT 2010,volume 6498 of Lecture Notes in Computer Science, pages 292–310. http://dx.doi.org/10.1007/978-3-642-17401-8_21*, 2010.

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

[46] W., Gorissen, P., Hollmann, H.D.L. Michiels, "Cryptanalysis of a generic class of white-box implementations," *In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS. Springer, Heidelberg* , vol. 5381, pp. 414–428, 2009.

[47] Lepoint T., Paillier P., Rivain M. Delerablée C., "White-Box Security Notions for Symmetric Encryption Schemes," *In: Lange T., Lauter K., Lisoněk P. (eds) Selected Areas in Cryptography -- SAC 2013. SAC 2013. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg*, vol. 8282, 2013.

[48] B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K. Barak, "On the (Im)possibility of Obfuscating Programs," *In Kilian, J., ed.: Advances in Cryptology – CRYPTO 2001. Volume 2139 of Lecture Notes in Computer Science., Springer Verlag 1–18*, 2001.

[49] James A. Muir, "A Tutorial on White-box AES," *Mathematics in Industry. To appear. http://www.ccsl.carleton.ca/~jamuir/papers/wb-aes-tutorial.pdf*, 2012.

[50] Jacques Patarin, "Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new Families of asymmetric Algorithms," *EUROCRYPT'96, LNCS vol. 1070, Springer*, pp. 33-48, 1996.

[51] Herv′e Chabanne and Emmanuelle Dottax Julien Bringer, "Perturbing and Protecting a Traceable Block Cipher," *CMS 2006, LNCS vol. 4237, Springer*, pp. 109–119, 2006.

[52] S. Niksefat, S. Sadeghian and B. Sadeghiyan P. Mohassel, "An Efficient Protocol for Oblivious DFA Evaluation and Applications," *In Proceedings of Cryptographers' Track at the RSA Conference*, pp. 398-415, 2012.

[53] S. Katzenbeisser, and M. Celik J.R. Troncoso-Pastoriza, "Privacy preserving error resilient DNA searching through oblivious automata," *n Proceedings of the 14th ACM conference on Computer and communications security, ACM*, pp. 519–528, 2007.

[54] K. Frikken, "Practical private DNA string searching and matching through efficient oblivious automata evaluation," *Data and Applications Security XXIII*, pp. 81–94, 2009.

[55] J. Katz and L. Malka, "Secure text processing with applications to private DNA matching," *In Proceedings of the 17th ACM conference on Computer and communications security. ACM*, pp. 485–492, 2010.

[56] C. Hazay, and J. Sorensen R. Gennaro, "Text search protocols with simulation based security," *Public Key Cryptography–PKC*, pp. 332–350, 2010.

[57] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," *Advances in Cryptology-ASIACRYPT*, pp. 195–212, 2010.

[58] Hacigümüs H., "Privacy in Database-as-a-Service Model," *University of California, PhD Thesis*, 2003.

[59] Mitzenmacher M. Chang Y.-C., "Privacy Preserving Keyword Searches on Remote Encrypted Data," *Lecture Notes in Computer Science: Applied Cryptography and Network Security*, vol. 3531, pp. 442-455, 2005.

[60] Hovsepyan M., Jivanyan A Khachatryan G., "Efficient Secure Pattern Search Algorithm," *In Proc. of 10th International Conference on Computer Science and Information Technologies (CSIT) and IEEE: Conference, pp. 147-151 in CSIT and 90-94 in IEEE, Yerevan, Armenia*, 2015.

[61] Boldyreva A., O'Neill A. Bellare M., "Deterministic and Efficiently Searchable Encryption," *In Proc. of the 27th Annual International Cryptology Conference on Advances in Cryptology*, pp. 535-552, 2007.

[62] C. Papamanthou, and T. Roeder S. Kamara, "Dynamic searchable symmetric encryption," *In T. Yu, G. Danezis, and V. D. Gligor, editors, ACM CCS 12*, pp. 965–976, 2012.

[63] Premasathian N., "Searchable Encryption Schemes: With Multiplication and Simultaneous Congruences," *International Conference on Information Security and Cryptology (ISCISC)*, 2012.

[64] Sedghi S., Doumen J., Hartel P., Jonker W. Van Liesdonk P., "Computationally Efficient Symmetric Encryption," *In Proc. of Secure Data Management (SDM) 7th VLDB Workshop*, 2010.

[65] Papamanthou C. Kamara S., "Parallel and Dynamic Searchable Symmetric Encryption," *Financial cryptography and data security*, pp. 258–274, 2013.

[66] Wanger D., Perrig A. Song D. X., "Practical Techniques for Search on Encrypted Data," *In Proceedings of IEEE Symposium on Security and Privacy*, 2000.

[67] Goh E.-J., "Secure indexes," *Cryptology ePrint Archive, no. 216. [Online]. http://eprint.iacr.org/2003/216/*, 2003.

[68] Papamanthou C., Shi E. Stefanov E., "Practical Dynamic Searchable Encryption with Small Leakage," *NDSS Symposium*, 2014.

[69] Prabhakaran M. Gunter C. A. Naveed M., "Dynamic Searchable Encryption via Blind Storage," *Security and Privacy (SP), IEEE Symposium*, pp. 639-654, 2014.

[70] Ray I. Strizhov M., "Multi-Keyword Similarity Search Over Encrypted Cloud Data," *Cryptology ePrint Archive, no. 137*, 2015.

[71] V.S. Miller, "Use of Elliptic curves in cryptography," *Advanced in Cryptology-Crypto-85 Proceedings, Springer-Verlag*, pp. 417--426, 1986.

[72] Kumaraswamy S., Latif S. Mather T., "Cloud Security and Privacy : An Enterprise Perspective on Risks and Compliance.," *O'Reilly Media, Inc.*, 2009.

[73] Benson K., Hohenberg S. Ateniese G., "Key-Private Proxy Re-encryption," *In Proc. of RSA Conference on Topics in Cryptology: Cryptographers' Track*, pp. 279-294,

2009.

[74] Ateniese G. Green M., "Identity-Based Proxy Re-encryption," *In Proc. of the 5th International Conference on Applied Cryptography and Network Security (ACNS)*, pp. 288-306, 2007.

[75] Bleumer G., Strauss M. Blaze M., "Divertible Protocols and Atomic Proxy Cryptography," *In Proc. of Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques*, 1998.

[76] G. Rothblum, A. Shelat, and V. Vaikuntanathan S. Hohenberger, "Securely Obfuscating Re-Encryption," *In "Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007", Lecture Notes in Computer Science 4392*, pp. 233-252, 2007.

[77] Frakes, W. B., "Stemming algorithms," *Information retrieval: data structures and algorithms.: Upper Saddle River, NJ: Prentice-Hall, Inc*, 1992.

[78] W. Michiels, "Opportunities in white-box cryptography," *IEEE Security & Privacy, 8(1)*, pp. 64–67, 2010.

APPENDIX A. GLOSSARY OF ACRONYMS

AES – Advanced Encryption Standard

API – Application Programming Interface

ASP – Application Service Provider

BGE - Billet, Gilbert, Ech-Chatbi

DaaS – Data as a Service

DFA – Deterministic Finite Automata

DES - Data Encryption Standard

DLP - discrete logarithm problem

DRM - Digital Right Management

GCD – Greatest Common Divisor

IDE - input output data encoding

ODE - output data encoding

IP - Isomorphism of Polynomials

MB - mixing bijection

OOP – Object Oriented Programming

OT – Oblivious Transfer

PK – Public Key

PRF – Pseudo-Random Function

SaaS – Software as a Service

SE – Searchable Encryption

SFE – Secure Function Evaluation

SPM – Secure Pattern Matching

SSE – Searchable Symmetric Encryption

SLT – Substitution-Linear Transformation

WB – White Box

WBC – White Box Cryptography

WBAC – White box attack context

AEw/oS – Advanced Encryption without standard S-boxes