ON VERIFICATION OF RESTRICTED EXTENDED AFFINE EQUIVALENCE OF VECTORIAL BOOLEAN FUNCTIONS

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

BY

AHMET SINAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

SEPTEMBER 2012

Approval of the thesis:

ON VERIFICATION OF RESTRICTED EXTENDED AFFINE EQUIVALENCE OF VECTORIAL BOOLEAN FUNCTIONS

submitted by **AHMET SINAK** in partial fulfillment of the requirements for the degree of **Master of Science in Department of Cryptography, Middle East Technical University** by,

Prof. Dr. Bülent KARASÖZEN ________________
Director, Graduate School of **Applied Mathematics**

Prof. Dr. Ferruh ÖZBUDAK ________________
Head of Department, **Cryptography**

Prof. Dr. Ferruh ÖZBUDAK ________________
Supervisor, **Mathematics Department, METU**

Dr. Oğuz Yayla ________________
Co-supervisor, **Cryptography Department**

Examining Committee Members:

Assoc. Prof. Dr. Ali Doğanaksoy ________________
Mathematics Department, METU

Prof. Dr. Ferruh Özbudak ________________
Mathematics Department, METU

Assoc. Prof. Dr. Melek Diker Yücel ________________
Electrics and Electronics Engineering Dept., METU

Assist. Prof. Dr. Zülfükar Saygı ________________
Mathematics Department, TOBB ETU

Dr. Oğuz Yayla ________________
Cryptography Department, METU

Date: ________________

ABSTRACT

ON VERIFICATION OF RESTRICTED EXTENDED AFFINE EQUIVALENCE OF VECTORIAL BOOLEAN FUNCTIONS

Sınak, Ahmet
M.S., Department of Cryptography
Supervisor : Prof. Dr. Ferruh ÖZBUDAK
Co-Supervisor : Dr. Oğuz Yayla

SEPTEMBER 2012, 66 pages

Vectorial Boolean functions are used as S-boxes in cryptosystems. To design inequivalent vectorial Boolean functions resistant to known attacks is one of the challenges in cryptography. Verifying whether two vectorial Boolean functions are equivalent or not is the final step in this challenge. Hence, finding a fast technique for determining whether two given vectorial Boolean functions are equivalent is an important problem. A special class of the equivalence called restricted extended affine (REA) equivalence is studied in this thesis. We study the verification complexity of REA-equivalence of two vectorial Boolean functions for some types, namely types I to VI. We first review the verification of the REA-equivalence types I to IV given in the recent work of Budaghyan and Kazymyrov (2012). Furthermore, we present the complexities of the verification of REA-equivalence types I and IV in the case basic simultaneous Gaussian elimination method is used. Next, we present two new REA-equivalence types V and VI with their complexities. Finally, we give the algorithms of each type I to VI with their MAGMA codes.

Keywords: Vectorial Boolean functions, REA-equivalence, AB-functions, APN-functions, system of linear equations.

ÖZ

VEKTÖR BOOLE FONKSİYONLARININ KISITLI GENİŞLETİLMİŞ AFİN DENKLİĞİ ÜZERİNE

Sınak, Ahmet
M.S., Kriptografi Bölümü
Tez Yöneticisi : Prof. Dr. Ferruh Özbudak
Ortak Tez Yöneticisi : Dr. Oğuz Yayla

EYLÜL 2012, 66 sayfa

Vektör Boole fonksiyonları kriptosistemlerde S-kutularıdır. Kriptografik ataklara dayanıklı denk olmayan vektör Boole fonksiyonları tasarlamak kriptografide önemli amaçlardan birisidir. İki vektör Boole fonsiyonların birbiri ile denk olup olmadığını anlamak bu amacın son basamağıdır. Dolayısıyla, vektör Boole fonksiyonlarının denkliğine bakmak için hızlı bir yöntem bulmak önemli bir problemdir. Denkliğin bir sınıfı olan kısıtlı genişletilmiş afin (KGA)-denkliği bu tezde çalışılmıştır. KGA-denkliğinin I ile VI arasındaki çeşitleri için iki vektör Boole fonksiyonların birbiri ile denk olup olmadığını anlamanın karmaşıklığını çalıştık. Öncelikle, yakın zamandaki Budaghyan ve Kazymyrov (2012)'un çalışmasındaki KGA-denkliğinin I ile IV arasındaki çeşitlerini inceledik. Ayrıca, KGA-denkliğinin I ve IV çeşitleri için temel eş-zamanlı Gauss eleme yöntemi kullanılmasıyla oluşan karmaşıklıkları sunduk. Sonra, KGA-denkliğinin iki yeni çeşidi V ve VI'yı karmaşıklıkları ile birlikte verdik. Son olarak, KGA-denkliğinin I ile VI arasındaki çeşitleri için algoritmaları MAGMA kodları ile birlikte sunduk.

Anahtar Kelimeler: Vektör Boole fonksiyonlar, KGA-denkliği, AB-fonksiyonlar, APN-fonksiyonlar, doğrusal denklemler sistemi.

Aileme ve Yeğenlerime

ACKNOWLEDGMENTS

I acknowledge financial support from TUBITAK-BIDEB 2210 program during my graduate academic life. I would like to express my gratitude to my supervisor Prof. Dr. Ferruh Özbudak for his inspiring guidance and valuable suggestions.

I am indebted to my co-supervisor Dr. Oğuz Yayla who helped me to organize my thesis. I am grateful to him for correcting my poor English in this paper. I also gratefully appreciate his considerable assistance concerning my graduate academic life.

Furthermore, I thank to all my friends, who are with me, for their patience and understanding. Specifically, my colleagues at the Institute of Applied Mathematics deserve many thanks for encouraging me.

Lastly, and most importantly, I owe my thanks to my family for their continuous support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

1.1 Introduction

In this thesis, the verification of equivalence of vectorial Boolean functions is studied. In particular, the special case of *Extended Affine (EA)-equivalence*, called *Restricted Extended Affine (REA)-equivalence* of vectorial Boolean functions is considered. Boolean functions play an important role in providing high-level security for modern ciphers. They are mainly used in cryptography as *non-linear combining* in stream ciphers and as *substitution boxes (S-boxes)* in block ciphers. The most important contribution of these functions is high resistance to the differential and linear cryptanalysis, which are the main attacks on modern ciphers.

The verification and generation of vectorial boolean functions with optimal characteristics, which are resistant to attacks, are two important problems in cryptography. The purpose of this thesis is to contribute to solving the verification problem of *s-boxes* in block ciphers. Sometimes equivalence classes of Boolean functions are helpful for cryptographers to achieve necessary properties without losing others (i.e. δ-uniformity, nonlinearity, etc). Therefore, it is required to check several functions for equivalence. For instance, once one finds a new vectorial boolean function, it is necessary to verify whether it is equivalent to already known ones which are used in block ciphers as substitution boxes (see [17]). We present some fundamental work related to vectorial Boolean functions and their equivalences in Chapter 2.

The complexity of exhaustive search for checking EA-equivalence of two functions from $\mathbb{F}_2^n$ to itself equals to $O(2^{3n^2+2n})$ (see [7]). When $n = 6$, the complexity is 2^{120} which makes it impossible to compute with the help of existing tools. The restricted form of *EA-equivalence* needs to be defined because of this high cost. Thus, *REA-equivalence* was studied in [2] and

[7]. Alex Biryukov et al.[2] showed that if given functions are permutations of $\mathbb{F}_2^n$, the complexity of determining REA-equivalence equals to $O(n^2 2^n)$ for the case of linear equivalence, and $O(n^2 2^{2n})$ for affine equivalence. In addition, L. Budaghyan and Kazymyrov [7] showed that some types of REA-equivalence can be verified for arbitrary functions. These types and two new types of REA-equivalence are studied in Chapter 5 in this thesis.

1.2 Overview of the thesis and of our contributions

We deal with the complexities of verification procedures of REA-equivalence types. There are four motivations of this thesis. The first one is to expose the verification procedures of REA-equivalence types given in [7]. The verification procedures of these types are analyzed in Section 5.1. Our procedures are supported with examples. Next motivation is to resolve the complexities of the verification procedures. In the calculation of the complexities, we count the number of polynomial evaluations and the explicit number of field operations modulo 2. The detailed complexities of verification for these types are given in Table 5.17. Third one is to obtain new REA-equivalence types. The verification procedures of two new types, namely types V and VI, are constructed in Section 5.2. The detailed complexities of verification for these types are given in Table 5.18. Finally, we introduce the pseudocodes of the verification procedures of REA-equivalence types. We also implement these pseudocodes in mathematical software package MAGMA [8]. The thesis is organized as follows:

- Chapter 2 introduces basic definitions, equivalence algorithms for vectorial Boolean functions and their properties.
- Chapter 3 presents previous work in [7] connected with our studies.
- Chapter 4 describes a basic method, namely the simultaneous Gaussian elimination method, to verify some REA-equivalence types.
- Chapter 5 provides verification procedures of REA-equivalence types with their complexities and pseudocodes.
- Chapter 6 gives a brief summary of this thesis.
- Finally, Appendix gives the MAGMA codes of the algorithms of verification of REA-equivalence types.

CHAPTER 2

PRELIMINARIES

In this chapter, we present fundamental definitions and results used in the subsequent chapters. For further explanations, applications and previous work, please see [10, 7, 9, 12, 2] and references there in.

Throughout this thesis, field additions modulo 2 (in $\mathbb{F}_2$) will be denoted by $\oplus$ (*XOR*), and field multiplications modulo 2 (in $\mathbb{F}_2$) will be denoted by $\odot$ (*AND*). However, the additions of elements of the finite fields $\mathbb{F}_{2^n}$ will be denoted by $+$ despite the fact that they are performed in characteristic 2. Since $\mathbb{F}_2^n$ can often be identified with $\mathbb{F}_{2^n}$, we shall also denote by $+$ the addition of vectors of $\mathbb{F}_2^n$ when $n > 1$.

2.1 Vectorial Boolean Functions

This section presents the vectorial boolean function theory from the point of view of cryptography. First, the definition of vectorial Boolean functions is given. Second, the techniques of their representation and transformation are defined. Finally, the fundamental properties of vectorial Boolean functions and concepts related to cryptography are summarized.

Definition 2.1.1 *Let n and m be two positive integers. The functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ are called (n, m)-functions. Such function F being given, the Boolean functions $f_1, \cdots, f_m$ defined at every $x \in \mathbb{F}_2^n$, by $F(x) = (f_1(x), \cdots, f_m(x))$, are called the coordinate functions of F. When the numbers m and n are not specified, (n, m)-functions are called multi-output Boolean functions, vectorial Boolean functions or S-boxes.*

The family of (n, m)-functions obviously includes the (single-output) Boolean functions which correspond to the case $m = 1$

2.2 Representation of Vectorial Boolean Functions

There are two different ways of representing vectorial Boolean functions.

2.2.1 Algebraic Normal Form

The notion of algebraic normal form of Boolean functions can easily be extended to vectorial Boolean functions. Each coordinate function of such a function F is uniquely represented as a polynomial on n variables, with coefficients in $\mathbb{F}_2$ and in which every variable appears in each monomial with degree 0 or 1. Therefore, the function F itself is uniquely represented as a polynomial of the same form with coefficients in $\mathbb{F}_2^m$:

$$F(x) = \sum_{I \in P(N)} a_I \left(\prod_{i \in I} x_i \right) = \sum_{I \in P(N)} a_I x^I,$$

where $P(N)$ denotes the power set of $N = \{1, \cdots, n\}$, and a_I belongs to $\mathbb{F}_2^m$. This polynomial is called the *algebraic normal form (ANF)* of F. Keeping the i-th coordinate of each coefficient in this expression gives back the ANF of the i-th coordinate function of F. The *algebraic degree* of the function is by definition the *global degree* of its ANF: $d^\circ F = max\{|I| | a_I \neq (0, \cdots, 0); I \in P(N)\}$

2.2.2 Univariate Polynomial Representation

A second representation of the (n, m)-functions exists when $m = n$. If we identify vector space $\mathbb{F}_2^n$ with the finite field $\mathbb{F}_{2^n}$, a function $F : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is then uniquely represented as *univariate polynomial* over $\mathbb{F}_{2^n}$ of degree at most $2^n - 1$:

$$F(x) = \sum_{j=0}^{2^n-1} \delta_j x^j, \delta_j \in \mathbb{F}_{2^n}. \tag{2.1}$$

If m is a divisor of n, then any (n, m)-function F can also be viewed as a function from $\mathbb{F}_{2^n}$ to itself, since $\mathbb{F}_{2^m}$ is a subfield of $\mathbb{F}_{2^n}$ [10]. Hence, the function admits a univariate polynomial representation. Note that this unique polynomial can be represented in the form

$tr_{n/m}\left(\sum_{j=0}^{2^n-1}\delta_j x^j\right)$, where $tr_{n/m}(x) = x + x^{2^m} + x^{2^{2m}} + x^{2^{3m}} + \cdots + x^{2^{n-m}}$ is the trace function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$. It is possible to read the algebraic degree of F from the univariate polynomial representation. Let us denote by $w_2(j)$ the number of nonzero coefficients j_s in the binary expansion $\sum_{s=0}^{n-1} j_s 2^s$ of j, i.e. $w_2(j) = \sum_{s=0}^{n-1} j_s$. The number $w_2(j)$ is called the 2-*weight* of j. Then, the function F has algebraic degree $max_{j=0,\cdots,2^n-1|\delta_j\neq 0} w_2(j)$. Indeed, according to the above equalities, the algebraic degree of F is clearly bounded above by this number. Moreover, it can not be strictly smaller, because the number $2^{n\sum_{i=0}^{d}\binom{n}{i}}$ of those (n,n)-functions of algebraic degrees at most d equals to the number of those univariate polynomials

$$\sum_{j=0}^{2^n-1} \delta_j x^j, \delta_j \in \mathbb{F}_{2^n},$$

such that $max_{j=0,\cdots,2^n-1|\delta_j\neq 0} w_2(j) \leq d$.

In particular, F is $\mathbb{F}_2$-linear if and only if F is a linearized polynomial over $\mathbb{F}_{2^n}$:

$$F(x) = \sum_{j=0}^{n-1} \delta_j x^{2^j}, \delta_j \in \mathbb{F}_{2^n}.$$

Function $L : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^m}$ is called *linear mapping* if it satisfies $L(x + y) = L(x) + L(y)$ for $x, y \in \mathbb{F}_2^n$. The sum of a linear function and a constant is called an *affine function*. Any affine function $A : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^m}$ can be represented in matrix form

$$A(x) = M \cdot x + V, \tag{2.2}$$

where $m \times n$ matrix $M \in \mathbb{F}_2^{m\times n}$ and $m \times 1$ vector $V \in \mathbb{F}_2^m$. All operations are performed in $\mathbb{F}_2$. Thus, the equation (2.2) can be rewritten as

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,n} \\ m_{2,1} & m_{2,2} & \dots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m,1} & m_{m,2} & \dots & m_{m,n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$

with $a_i, v_i, x_l, m_{j,k} \in \mathbb{F}_2$ for all $i, j \in \{1, \cdots, m\}$ and $k, l \in \{1, \cdots, n\}$.

Example 2.2.1 *A vectorial Boolean function* $F : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$ *given as*

$$F(x_3, x_2, x_1) = (x_3 + 1, x_1 + x_1x_3, x_1 + x_2 + x_1x_3 + x_2x_3)$$

is represented in the ANF, its algebraic degree is 2 and Univariate polynomial form of F:

$$F(x) = x^6 + x^3 + x + 1, x \in \mathbb{F}_2^3,$$

$max\{w_2(6), w_2(3), w_2(1), w_2(0)\} \leq d = 2.$

2.3 Differentially δ-uniform

For any positive integers n and m, a function $F : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^m}$ is called *differentially* δ-uniform [18] if for every $a \in \mathbb{F}_2^n \backslash \{0\}$ and every $b \in \mathbb{F}_2^m$, the equation $F(x) + F(x+a) = b$ admits at most δ solutions. Vectorial Boolean functions used as S-boxes in block ciphers must have low differential uniformity to allow high resistance to differential cryptanalysis (see [1]).

2.4 The Walsh Transform

We shall define the *Walsh transform* of an (n, m)-function F or the function which maps any ordered pair $(u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ to the value at u of the Walsh transform of the component function $v \cdot F$, that is,

$$\lambda(u, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus u \cdot x},$$

where $\cdot$ denotes inner products in $\mathbb{F}_2^m$ and F_2^n, respectively. The set

$$\left\{ \lambda(u, v) | (u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0 \right\} \tag{2.3}$$

is called the *Walsh spectrum* of F and the set $\Lambda_F = \left\{ |\lambda(u, v)| \, |(u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0 \right\}$ is called the *extended Walsh spectrum* of F.

2.5 AB and APN Functions

Definition 2.5.1 *[9] The function $F : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^n}$ is said to be almost perfect nonlinear (APN) if the equation*

$$F(x) + F(x + a) = b \tag{2.4}$$

have no more than two solutions in $\mathbb{F}_2^n$ for all $a \in \mathbb{F}_2^n \backslash \{0\}$, $b \in \mathbb{F}_2^n$.

Clearly, the equation (2.4) must have either 0 or 2 solutions. The notion of APN function is closely connected to the notion of almost bent (AB) function which can be described in terms of the Walsh transform of a function [11].

Definition 2.5.2 *[9] The function $F : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^n}$ is said to be an almost bent (AB) function if the Walsh spectrum of F given by (2.3) equals to $\left\{0, \pm 2^{\frac{n+1}{2}}\right\}$.*

AB functions exist only when n is odd. Definition 2.5.2 does not depend on a particular choice of the inner product in $\mathbb{F}_2^n$. If we identify $\mathbb{F}_2^n$ with $\mathbb{F}_{2^n}$, we can take $x \cdot y = tr(xy)$ in (2.3), where tr is the trace function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$.

2.5.1 Properties of Stability

Let us recall the properties of stability of APN functions given by Nyberg [18] and give some others. Every AB function is an APN function but the converse is not true in general. See [10] for a comprehensive survey on APN and AB functions.

Proposition 2.5.3 *[9] The right and the left compositions of an APN (resp. AB) function by an affine permutation are APN (resp. AB). The inverse of an APN (resp. AB) permutation is APN (resp. AB).*

Proposition 2.5.4 *[9] Let $F(x)$ be an APN function (resp. an AB function) from $\mathbb{F}_2^n$ to itself, and $A(x)$ be an affine function. Then the function $F(x) + A(x)$ also is an APN (resp. AB).*

Finding the vectorial Boolean functions over finite fields which satisfy nonlinearity conditions is one of the main problems in cryptography. Once found, the second problem occurs in determining whether they are equivalent or not in some sense to any of the functions already known [15]. In this thesis, our main question is whether two vectorial Boolean functions are equivalent or not. Some definitions of equivalence relations for vectorial Boolean functions will be given in Section 2.6 . Several notions of equivalence exist in [14], but the most useful equivalences for Boolean functions appear to be the Carlet-Charpin-Zinoviev (CCZ)-equivalence and the extended affine (EA)-equivalence. The restricted form of EA-equivalence, called REA-equivalence, is studied in this thesis.

2.6 Equivalences

Equivalence classes provide a powerful tool in verification of vectorial Boolean functions. However, the research has not been yet fully exploited in this area. More studies are needed to fully characterize equivalence classes in relation to their cryptographic properties, as well as to determine the number of equivalence classes for functions with more than six input variables. Another important problem in this area is to find a fast technique for determining whether two functions with more than six input variables are equivalent or not.

Definition 2.6.1 *[2] Let $F, G : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^n}$ be vectorial boolean functions. The functions F, G are called linear equivalent (LE) if there exist two invertible linear functions L_1 and L_2 of $\mathbb{F}_{2^n}$ such that*

$$F(x) = L_1 \circ G \circ L_2(x).$$

Definition 2.6.2 *[2] Let $F, G : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^n}$ be vectorial Boolean functions. The functions F, G are called affine equivalent (AE) if there exist two affine permutations A_1 and A_2 of $\mathbb{F}_{2^n}$ such that*

$$F(x) = A_1 \circ G \circ A_2(x).$$

Definition 2.6.3 *[7] Let $F, G : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^m}$ be vectorial Boolean functions. The functions F, G are called extended affine equivalent (EA-equivalent) if there exist an affine permutation A_1 of $\mathbb{F}_{2^m}$, an affine permutation A_2 of $\mathbb{F}_{2^n}$ and a linear function L_3 from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^m}$ such that*

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

Clearly, Affine functions A_1 and A_2 can be represented as $A_1(x) = L_1(x) + V_1$ and $A_2(x) = L_2(x) + V_2$ for some linear functions L_1 and L_2, and for some vectors $V_1 \in \mathbb{F}_2^{m\times 1}$, $V_2 \in \mathbb{F}_2^{n\times 1}$. Moreover, L_1, L_2, L_3 can be represented as $L_1(x) = M_1 \cdot x$, $L_2(x) = M_2 \cdot x$, $L_3(x) = M_3 \cdot x$ for some matrices $M_1 \in \mathbb{F}_2^{m\times m}$, $M_2 \in \mathbb{F}_2^{n\times n}$, $M_3 \in \mathbb{F}_2^{m\times n}$, respectively.

Remark 2.6.4 *EA-equivalence is represented in the matrix form as follows*

$$F(x) = M_1 \cdot G(M_2 \cdot x + V_2) + M_3 \cdot x + V_1$$

where $V_1 \in \mathbb{F}_2^{m\times 1}$, $V_2 \in \mathbb{F}_2^{n\times 1}$, $M_1 \in \mathbb{F}_2^{m\times m}$, $M_2 \in \mathbb{F}_2^{n\times n}$ and $M_3 \in \mathbb{F}_2^{m\times n}$.

The complexity of verification of *EA-equivalence* is extremely high. Therefore, L. Budaghyan and Kazymyrov [7] define the restricted form of EA-equivalence, called restricted extended affine-equivalence.

Definition 2.6.5 *[7] Let $F, G : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^m}$ be vectorial Boolean functions. The functions F, G are called restricted EA-equivalent (REA-equivalent) if at least one of the following holds*

- *L_1 or L_2 is the identity map,*
- *L_3 is the zero or the identity map,*
- *V_1 or V_2 is the zero vector.*

Linear and affine equivalences are particular cases of the REA-equivalence. Some REA-equivalence algorithms are presented in the Table 2.1.

Table 2.1: Some types of REA-equivalence

$L_1(x)$	$L_2(x)$	$L_3(x)$	V_1	V_2	Types	REA-equivalence	Source
		0	0	0	LE	$F(x) = M_1 \cdot G(M_2 \cdot x)$	[2]
		0			AE	$F(x) = M_1 \cdot G(M_2 \cdot x + V_2) + V_1$	[2]
	x	0		0	I	$F(x) = M_1 \cdot G(x) + V_1$	[7]
x		0	0		II	$F(x) = G(M_2 \cdot x + V_2)$	[7]
x	x			0	III	$F(x) = G(x) + M_3 \cdot x + V_1$	[7]
	x			0	IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	[7]
x				0	V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$	New
x		0		0	VI	$F(x) = G(M_2 \cdot x) + V_1$	New

The verification procedures for REA-equivalence types in [7] will be exposed in Chapter 5. We will also illustrate the verification procedures for new types, namely types V and VI, in Chapter 5.

The extended version of EA-equivalence, which is first defined in [4] is given below.

Proposition 2.6.6 *[9] Let F be an APN (resp. AB) function on $\mathbb{F}_2^n$ and L_1, L_2 be two linear functions from $\mathbb{F}_2^n \times \mathbb{F}_2^n$ to $\mathbb{F}_2^n$. Assume that (L_1, L_2) is a permutation on $\mathbb{F}_2^n \times \mathbb{F}_2^n$ such that the function $F_2(x) = L_2(F(x), x)$ is a permutation on $\mathbb{F}_2^n$. Then, the function $F_1 \circ F_2^{-1}$, where $F_1(x) = L_1(F(x), x)$ is APN (resp. AB).*

Remark 2.6.7 *For a function F from $\mathbb{F}_2^n$ to itself, we denote G_F the graph of the function F by*

$$G_F := \left\{(x, F(x)) | x \in \mathbb{F}_2^n\right\} \subset \mathbb{F}_2^{2n}.$$

The property of stability of APN and AB functions given in Proposition 2.6.6 leads to the definition of the following equivalence relation of functions.

Definition 2.6.8 *[4] (CCZ-Equivalence) Two functions $F, F' : \mathbb{F}_2^n \to \mathbb{F}_2^n$ are called Carlet-Charpin-Zinoviev(CCZ)-equivalent if there exists a linear (affine) permutation $L : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^{2n}$ such that $L(G_F) = G_{F'}$.*

We shall consider only the case of linear functions, but all statements related to the CCZ-equivalence can be easily extended for the case of affine functions, as well. A linear function $L : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^{2n}$ can be considered as a pair of linear functions $L_1, L_2 : \mathbb{F}_2^{2n} \times \mathbb{F}_2^n$ such that $L(x, y) = (L_1(x, y), L_2(x, y))$ for all $x, y \in \mathbb{F}_2^n$. Then for a function $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ we have $L(x, F(x)) = (F_1(x), F_2(x))$, where

$$F_1(x) = L_1(x, F(x)),$$
$$F_2(x) = L_2(x, F(x)).$$

Hence, the set $L(G_F) = \left\{(F_1(x), F_2(x)) | x \in \mathbb{F}_2^n\right\}$ is the graph of a function F' if and only if the function F_1 is a permutation. If L and F_1 are permutations then $L(G_F) = G_{F'}$, where $F' = F_2 \circ F_1^{-1}$ and the functions F and F' are CCZ-equivalent.

CCZ-equivalence is the most general known equivalence [4, 9] which partitions the set of functions into classes with the same nonlinearity and differential uniformity. Since CCZ-equivalent functions have the same differential uniformity and the same nonlinearity, the resistance of a function to linear and differential attacks is CCZ-invariant. CCZ-equivalent functions have also the same weakness/strength with respect to algebraic attacks.

Remark 2.6.9 *Every function CCZ-equivalent to an APN (resp. AB) function is also APN (resp. AB).*

Theorem 2.6.10 *([6, Theorem 1]) Two Boolean functions (i.e. the case of $m = 1$) of $\mathbb{F}_2^n$ are CCZ-equivalent if and only if they are EA-equivalent.*

Theorem 2.6.11 *([6, Theorem 3]) Let $n \geq 5$ and $k > 1$ be the smallest divisor of n. Then for any $m \geq k$, the CCZ-equivalence of (n, m)-functions is strictly more general than their EA-equivalence.*

Hence, EA-equivalent functions are CCZ-equivalent and also if a function F is a permutation, then F is CCZ-equivalent to F^{-1} [9]. However, CCZ-equivalent functions are not necessarily EA-equivalent even if these functions are inverse of each other [4]. Since EA-equivalent functions have the same differential uniformity and the same nonlinearity, the resistance of a function to linear and differential attacks is EA-invariant. EA-equivalence classes of functions over $\mathbb{F}_2^n$ preserve APN and AB properties desirable for S-box functions in block cipher [15]. They lead to infinite classes of AB and APN polynomials, which are EA-inequivalent to power functions. The algebraic degree of a function (if it is not affine) is invariant under EA-equivalence but, in general, it is not preserved by CCZ-equivalence. It is obvious that the properties which are EA-invariant is also REA-invariant.

Remark 2.6.12 *AE and LE are special cases of REA-equivalence. Similarly, REA-equivalence is a special case of EA-equivalence. On the other hand, CCZ-equivalence is a more general form of EA-equivalence. (i.e. LE and AE-equivalences $\Longrightarrow$ REA-equivalence $\Longrightarrow$ EA-equivalence $\Longrightarrow$ CCZ-equivalence, for any two vectorial Boolean functions.)*

Definition 2.6.13 *[6, Definition 1] Two (n, m)-functions F and $F^{'}$ are called Extended CCZ-equivalent if the indicators of their graphs $G_F = \left\{(x, F(x)); x \in \mathbb{F}_2^n\right\}$ and $G_F^{'} = \left\{(x, F^{'}(x)); x \in \mathbb{F}_2^n\right\}$ are CCZ-equivalent.*

Corollary 2.6.14 *[6, Corollary 3] Let F and $F^{'}$ be two (n, m)-functions. Then, F and $F^{'}$ are ECCZ-equivalent if and only if they are CCZ-equivalent.*

Example 2.6.15 *[16, Example 5] There are 7 CCZ classes of functions $F : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$, each of which is a single EA class. When $F : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$, we consider functions with low differential uniformity $\delta = 2, 4, 8$. The two EA classes of APN functions which form a single CCZ-class are already known.*

Example 2.6.16 *[5, Proposition 3] The functions $F(x) = x^3 + tr(x^9)$ is CCZ-equivalent to function $G(x) = x^3 + tr(x^9) + (x^2 + x)tr(x^3 + x^9)$ on $\mathbb{F}_2^n$ when $n = 3$.*

The functions $F(x) = x + x^2 + x^3 + x^4$ and $G(x) = x^3 + x^4$ can be easily computed over $\mathbb{F}_2^n$ for $n = 3$. When these functions are checked for REA-equivalence of type III, it is seen that they are REA-equivalent and in the same equivalence class. However, they are REA-inequivalent for REA-equivalence of type VI.

Example 2.6.17 *[5, Theorem 3] The function $F(x) = x^3 + tr(x^9)$ is CCZ-inequivalent to function $G(x) = x^9$ on $\mathbb{F}_2^n$ when $n = 7$.*
The function F can be easily computed over $\mathbb{F}_2^n$ for $n = 7$, $F(x) = x^3 + x^9 + x^{17} + x^{18} + x^{34} + x^{36} + x^{68} + x^{72}$. When these functions are checked for REA-equivalences of types I, II, III, IV, V, VI, it is seen that they are REA-inequivalent for all types. Because they are CCZ-inequivalent.

CHAPTER 3

PREVIOUS WORK

In this chapter, the previous work [7] related to solving the system of equations is presented. Let $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $G : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be two vectorial Boolean functions. The aim of this chapter is to present basic methods to find an $m \times n$ matrix M satisfying the equation $F(x) = M \cdot G(x)$. The trivial method is the exhaustive search method, that is checking the equation $F(x) = M \cdot G(x)$ for all matrices $M \in \mathbb{F}_2^{m \times n}$ and for all $x \in \mathbb{F}_2^n$. Consequently, the total complexity of obtaining the $m \times n$ matrix M satisfying the equation $F(x) = M \cdot G(x)$ by exhaustive search method (as given in [7]) equals to

$$O(2^{nm+n}),$$

where $O(2^{nm})$ and $O(2^n)$ are complexities of checking for all matrices $M \in \mathbb{F}_2^{m \times n}$ and for all $x \in \mathbb{F}_2^n$, respectively. Apart from the trivial method, there also exist some other methods to find M satisfying the equation $F(x) = M \cdot G(x)$. One of them is the natural method which is based on solving the system of equations. Any system of m equations with n variables can be solved for $n \leq 32$ with the complexity $u = O(max\{n, m\}^2)$ (see [7] for details), which gives the complexity of finding m by n matrix M. The values of $F(x)$ and $G(x)$ for all $x \in \mathbb{F}_2^n$ should be first calculated. Then the matrix representation of the equation $F(x) = M \cdot G(x)$, where M is unknown, $F(x)_i$ is the i-th component of $F(x)$ for $i = 1, 2, \ldots, m$ and $G(x)_j$ is the j-th component of $G(x)$ for $j = 1, 2, \ldots, n$, is as follows:

$$\begin{pmatrix} F(x)_1 \\ F(x)_2 \\ \vdots \\ F(x)_m \end{pmatrix} = \begin{pmatrix} m_{1,1} & m_{1,2} & \ldots & m_{1,n} \\ m_{2,1} & m_{2,2} & \ldots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m,1} & m_{m,2} & \ldots & m_{m,n} \end{pmatrix} \cdot \begin{pmatrix} G(x)_1 \\ G(x)_2 \\ \vdots \\ G(x)_n \end{pmatrix} \tag{3.1}$$

To find M, it is required to solve 2^n equations with n variables for each row of the matrix M. Thus, the i-th row of the matrix M can be found by solving the system of equations given below:

$$\begin{aligned}
F(x_1)_i &= m_{i,1}G(x_1)_1 + m_{i,2}G(x_1)_2 + \cdots + m_{i,n}G(x_1)_n \\
F(x_2)_i &= m_{i,1}G(x_2)_1 + m_{i,2}G(x_2)_2 + \cdots + m_{i,n}G(x_2)_n \\
&\vdots \\
F(x_{2^n})_i &= m_{i,1}G(x_{2^n})_1 + m_{i,2}G(x_{2^n})_2 + \cdots + m_{i,n}G(x_{2^n})_n
\end{aligned} \tag{3.2}$$

The matrix representation of the equations in (3.2) for the i-th row of M is as follows:

$$\begin{pmatrix} F(x_1)_i \\ F(x_2)_i \\ \vdots \\ F(x_{2^n})_i \end{pmatrix} = \begin{pmatrix} G(x_1)_1 & G(x_1)_2 & \dots & G(x_1)_n \\ G(x_2)_1 & G(x_2)_2 & \dots & G(x_2)_n \\ \vdots & \vdots & \ddots & \vdots \\ G(x_{2^n})_1 & G(x_{2^n})_2 & \dots & G(x_{2^n})_n \end{pmatrix} \cdot \begin{pmatrix} m_{i,1} \\ m_{i,2} \\ \vdots \\ m_{i,n} \end{pmatrix}. \tag{3.3}$$

The system in (3.3) is defined as $F(x)_i = G(x) \cdot M_i$ such that $F(x)_i$ is the i-th component of $F(x)$ and M_i is the i-th row of M, for all $i \in \{1, \cdots, m\}$, $x_j \in \mathbb{F}_2^n$ for all $j \in \{1, \cdots, 2^n\}$. For simplicity, denoted the system (3.3)in (3.3) $F_i = \mathbf{G} \cdot M_i$, where $\mathbf{G} \in \mathbb{F}_2^{2^n \times n}$ is the coefficients matrix of the system. The augmented matrix $[\mathbf{G}|F_i]$ of the system (3.3) for the i-th row of M is as follows for all $i \in \{1, \cdots, m\}$:

$$\mathbf{Aug} = \left(\begin{array}{cccc|c} G(x_1)_1 & G(x_1)_2 & \dots & G(x_1)_n & F(x_1)_i \\ G(x_2)_1 & G(x_2)_2 & \dots & G(x_2)_n & F(x_2)_i \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ G(x_{2^n})_1 & G(x_{2^n})_2 & \dots & G(x_{2^n})_n & F(x_{2^n})_i \end{array}\right) \tag{3.4}$$

Budaghyan and Kazymyrov [7, Proposition 2] use the Williams method [21] to reach that the system in (3.4) can be solved with the complexity of

$$O(max\{2^n, n\}^2) = O(2^{2n}).$$

It is obvious that the i-th row of M can be found by solving the system of equations (3.2). It is required to solve m-system of 2^n equations in n unknowns to find the matrix M. Consequently, the complexity of finding M (as given in [7]) equals to

$$O(m \cdot max\{2^n, n\}^2) = O(m2^{2n}). \tag{3.5}$$

Apparently, this complexity is extremely high. In order to reduce this high value, basic Gaussian elimination method is used simultaneously to solve (3.4) for all $i \in \{1, \cdots, m\}$ in Chapter 4.

CHAPTER 4

SIMULTANEOUS GAUSSIAN ELIMINATION METHOD

The main aim is to find the $m \times n$ matrix M satisfying the equation

$$F(x) = M \cdot G(x). \tag{4.1}$$

The system of equations (3.2) can also be solved by simultaneous application of the Gaussian elimination method for all $i \in \{1, \cdots, m\}$. The purpose of applying this method is to realize the better complexity of finding the matrix M. We first present the consecutive application of the Gaussian elimination method. Next, we will consider simultaneous Gaussian elimination method to solve the equation $[\mathbf{G}|F_i]$ in (3.4) for all $i \in \{1, \cdots, m\}$.

4.1 Consecutive Gaussian Elimination Method

We now give the basic details of the algorithm to solve the system of equations $[\mathbf{G}|F_i]$ in (3.4) by Gaussian elimination method. Given the system of equations we reduce their augmented matrix to reduced row echelon form. We can obtain a matrix $\mathbf{G}' \in \mathbb{F}_2^{2^n \times n}$, where $\mathbf{G}'$ is in reduced row echelon form, from matrix $\mathbf{G} \in \mathbb{F}_2^{2^n \times n}$ by a sequence of elementary row operations. There are three main steps of elementary row operations. We note that all operations are over $\mathbb{F}_2$.

Step I: Swap two rows: $R_i \longleftrightarrow R_j$

Use Step I if a row does not have any non zero entry in the corresponding column, it requires to change other row having non zero entry in this column.

Step II: Multiply a row by a non zero scalar: $\lambda \cdot R_i$ such that $\lambda \neq 0$.

Use Step II so that the first nonzero entry of a row is 1.

Step III: Add a scalar multiple of one row to a different row: $\lambda \cdot R_i + R_j$ such that $\lambda \neq 0$. Use Step III so that all nonzero entries in the columns containing the leading nonzero entry are zero.

We note that it is unnecessary to apply Step II in Algorithm 1 since the augmented matrix is over $\mathbb{F}_2$, i.e. the nonzero entry of a row is 1.

Algorithm 1 Gaussian elimination method over GF(2).

```
Require: G
Ensure: Row Reduced Ech(G)
1: If the first row don't have leading nonzero entry in the first column, any row having leading
   nonzero entry in the first column is interchanged with the first row by applying step I.
2: Step III must be applied to the first row so that all nonzero entries except leading nonzero entry
   in first column are 0. There may be at most $(2^n - 1)$ entries except leading nonzero entry in
   this column. So, we apply at most $(2^n - 1)$ times Step III, each of which is applied to entries of
   corresponding column.
3: Apply again the steps 1 and 2 to the $i$-th row of G for $i = 2, \cdots, n$.
4: return Row Reduced Ech(G)
```

Thus, the complexity of Algorithm 1 is as follows: Step I can be applied many times with complexity zero. Step III can be applied at most $n(2^n - 1)$ times. There are $(2^n - 1)(t - i + 1)$ field additions *modulo* 2 for the i-th row of **G** for $i = 1, 2, \ldots, n$, where the value t is the number of columns in the augmented matrix. Consequently, the number of field additions *modulo* 2 is

$$\begin{aligned}(2^n - 1)\,[t + (t-1) + \cdots + (t - (n-1))] &= (2^n - 1)\,[(t - n + 1) + \cdots + t] \\ &= (2^n - 1)\left[\tfrac{t(t+1)}{2} - \tfrac{(t-n)(t-n+1)}{2})\right] \\ &= 2^n(\tfrac{n(2t+1)-n^2}{2}) - (\tfrac{n(2t+1)-n^2}{2}).\end{aligned} \tag{4.2}$$

As there are $n + 1$ columns of the system in (3.4), the required number of field additions *modulo* 2 is

$$2^n(\frac{n^2 + 3n}{2}) - \frac{n^2 + 3n}{2} \tag{4.3}$$

to solve the system by applying Gaussian elimination method presented in Algorithm 1. For each row of M, we need to repeat the process of solving the system of equations consecutively. As a result, the required number of field additions *modulo* 2 to find M in (3.1) is

$$m(2^n(\frac{n^2 + 3n}{2}) - \frac{n^2 + 3n}{2}). \tag{4.4}$$

4.2 Simultaneous Gaussian Elimination Method

We continue with the presentation of the simultaneous application of Gaussian elimination method for finding the $m \times n$ matrix M satisfying the equation $F(x) = M \cdot G(x)$ if it exists. The main point is that the same coefficient matrix **G** is used in all systems (3.3) for each row of the matrix M. Thus, we solve m-systems of 2^n-equations in n unknowns to find the matrix M in a unique augmented matrix so that the complexity of finding the matrix M can be reduced. We call this as simultaneous Gaussian elimination method. The augmented matrix of m-systems for all rows of M is $[G|F_1|F_2|\dots|F_m]$, equivalently

$$\mathbf{Aug} = \left(\begin{array}{cccc|c|c|c} G'(x_1)_1 & G'(x_1)_2 & \dots & G'(x_1)_n & F'(x_1)_1 & \dots & F'(x_1)_m \\ G'(x_2)_1 & G'(x_2)_2 & \dots & G'(x_2)_n & F'(x_2)_1 & \dots & F'(x_2)_m \\ \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\ G'(x_{2^n})_1 & G'(x_{2^n})_2 & \dots & G'(x_{2^n})_n & F'(x_{2^n})_1 & \dots & F'(x_{2^n})_m \end{array}\right). \quad (4.5)$$

One can easily solve the system of equation in (4.5) by applying Algorithm 1. One obtain the complexity of the simultaneous application of Gaussian elimination method by substituting $t = m + n$ into (4.2). Hence, we obtain the following result.

Lemma 4.2.1 *The complexity of solving the system in (4.5), that is m-systems of 2^n equations in n unknowns, with simultaneous Gaussian elimination method is*

$$2^n\left(\frac{2mn + n^2 + n}{2}\right) - \frac{2mn + n^2 + n}{2} \quad (4.6)$$

field additions modulo 2, which gives the complexity of finding the m by n matrix M in the equation (4.1).

Remark 4.2.2 *Although we used the basic Gaussian elimination method simultaneously to solve the m-systems of 2^n equations in n unknowns, the corresponding complexity (4.6) is not consistent with the complexity (3.5) of solving the same systems of equations as given in [7]. That is, it seems that (4.6) is better than (3.5). We also note that the complexity (4.6) is better than the complexity (4.4).*

We now give a handy result that is used in our procedures.

Proposition 4.2.3 *[7] Any linear function $L : \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be converted to a matrix with n evaluations of L.*

Proof. The aim is to find an $m \times n$ matrix M satisfying

$$L(x) = M \cdot x$$

Suppose $rows_M(i)$ and $cols_M(j)$ are the i-th row and the j-th column of matrix M for $i \in \{0, 1, \cdots, m-1\}$ and $j \in \{0, 1, \cdots, n-1\}$, respectively.

$$\mathbf{M} = \left[\mathbf{m_{i,j}}\right]_{\mathbf{0 \le i \le m-1, 0 \le j \le n-1}} = \begin{pmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \dots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m-1,0} & m_{m-1,1} & \dots & m_{m-1,n-1} \end{pmatrix}$$

Each element of the set $U = \left\{2^j | 0 \le j \le n-1\right\}$ is equivalent to a vector having the value 1 in the j-th position and zeros elsewhere;

$$2^0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, 2^1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \cdots, 2^{n-1} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

It is clear that the set U is the subset of $\mathbb{F}_2^n$. The linear transformation L can then be computed for all $x \in U$;

$$\mathbf{L(2^j)} = \mathbf{M} \cdot \mathbf{2^j} = \begin{pmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \dots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m-1,0} & m_{m-1,1} & \dots & m_{m-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \tag{4.7}$$

$$= \begin{pmatrix} m_{0,j} \\ m_{1,j} \\ \dots \\ m_{m-1,j} \end{pmatrix} = cols_M(j), j \in \{0, 1, \cdots, n-1\}.$$

Obviously, every column, except the j-th column, becomes zero when multiplying the matrix M with the vector 2^j. In deed, $L(2^j) = M_j$ give us $cols_M(j)$ as in (4.7). To find all columns of M, it is necessary to compute n values of $L(2^j)$ for all $2^j \in U$. Consequently, the m by n matrix M can be found by n evaluations of L. □

CHAPTER 5

REA-EQUIVALENCE TYPES

In this chapter, the verification procedures of some REA-equivalence types are presented with their complexities. Some of them in the Tables 5.1 were studied by Budaghyan and Kazymyrow [7]. We now expose the verification procedures of these types with their complexities. In addition, our verification procedures are more extensive than the procedures given in [7].

Table 5.1: The types of REA-equivalence in [7]

Types	REA-equivalence
I	$F(x) = M_1 \cdot G(x) + V_1$
II	$F(x) = G(M_2 \cdot x + V_2)$
III	$F(x) = G(x) + M_3 \cdot x + V_1$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$

We note that the required number of field operations for the multiplication of the matrix $M \in \mathbb{F}_2^{m \times n}$ with vector $V \in \mathbb{F}_2^{n \times 1}$ given by $M \cdot V$ is

$$mn - m \text{ XORs}, \; mn \text{ ANDs}. \tag{5.1}$$

We also note that the required number of field operations for the addition of the vectors $M, V \in \mathbb{F}_2^{m \times 1}$ given by $M + V$ is

$$m \text{ XORs}. \tag{5.2}$$

5.1 Verification of REA-Equivalence Types

The complexities of REA-equivalence algorithms are explicitly calculated in terms of the number of polynomial evaluations, the number of field additions modulo 2 (*XORs*) and the number of field multiplications modulo 2 (*ANDs*).

We note that every vectorial Boolean function can be written in the form

$$H(x) = H'(x) + H(0) \tag{5.3}$$

where $H'(x)$ has terms of algebraic degree at least 1.

Remark 5.1.1 *We define $k_{F,v}$ for any given function $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ and any given set $V \subseteq img(F)$ as*

$$k_{F,v} := \left|\left\{y \in \mathbb{F}_2^n | F(y) = v\right\}\right|, \tag{5.4}$$

for $v \in V$.

Proposition 5.1.2 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions and G be defined by (5.3). Then the complexity of checking F and G for REA-equivalence of type I ($F(x) = M_1 \cdot G(x) + V_1$) is*

i. $2km^2$ *XORs,* $2km^2$ *ANDs and* $2^{n+1} + 2$ *evaluations in case the set* $V = \{2^i | 0 \leq i \leq m-1\} \subseteq img(G')$, *where* $k = \prod_{v \in V} k_{G',v}$.

ii. $2^n(\frac{m^2(2k+1)+m}{2}) + \frac{m^2(2k-1)-m}{2}$ *XORs,* $2km^2$ *ANDs and* $2^{n+1} + 2$ *evaluations in case G is arbitrary, where* $k = \prod_{v \in img(G')} k_{G',v}$.

Proof. Using (5.3), REA-equivalence of type I can be written in the following form

$$F'(x) + F(0) = M_1 \cdot G'(x) + M_1 \cdot G(0) + V_1,$$

and it gives us the following equations

$$\begin{cases} F(0) &= M_1 \cdot G(0) + V_1, \\ F'(x) &= M_1 \cdot G'(x). \end{cases} \tag{5.5}$$

To begin with, it is necessary to find M_1 from the second equation in (5.5). There exist two different cases for the function G'. If the set $V = \{2^i | 0 \leq i \leq m-1\} \subseteq img(G')$, then there exists at least one $x \in \mathbb{F}_2^n$ such that $G'(x) = 2^i$ for all $2^i \in V$. If not, we interchange F and G for REA-equivalence. To find those values of $x \in \mathbb{F}_2^n$, we need to compute the values $G'(x)$ for all $x \in \mathbb{F}_2^n$ whose complexity is 2^n evaluations of G'. Then, we compute the values $F'(x)$ for those values of $x \in \mathbb{F}_2^n$ in order to find matrix M_1 whose complexity is at most 2^n evaluations of F'. As a result, the i-th column of M_1 can be found by means of the values $F'(x) = M_1 \cdot 2^i = M_{1_i} = cols_{M_1}(i)$ such that $G'(x) = 2^i$ for all $i = 0, \cdots, m-1$ as in Proposition 4.2.3,

$$F'(x) = M_1 \cdot 2^i = \begin{pmatrix} m_{1,i} \\ m_{2,i} \\ \vdots \\ m_{m,i} \end{pmatrix} = M_{1_i}.$$

If there are more than one x such that $G'(x) = 2^i$, then there may be more than one i-th column of matrix M_1 any $i = 0, \cdots, m-1$. On the basis of this, there may be more than one matrices M_1 which can satisfy the equation $F'(x) = M_1 \cdot G'(x)$. The number of possible matrices M_1 is the value k which is the product of the size of the inverse image of elements 2^i in the set $img(G')$ for all $i \in \{0, \cdots, m-1\}$. For a given matrix M_1, one can easily compute V_1 from the first equation in (5.5). At this point, one needs 2 evaluations of $F(0)$ and $G(0)$. Secondly, the required number of field operations of matrix-vector product $M_{1_{m\times m}} \cdot G(0)_{m\times 1}$ as in (5.1) is $m^2 - m$ *XORs*, m^2 *ANDs*. Finally, the required number of field operations of m-bit vectors *XORs* $M_1 \cdot G(0) \oplus F(0)$ is m *XORs*. Therefore, the complexity of finding V_1 is m^2 *XORs*, m^2 *ANDs* and 2 evaluations. Moreover, we need to verify whether or not the matrices M_1 and V_1 satisfy the equation

$$F(x) = M_1 \cdot G(x) + V_1 \tag{5.6}$$

as well as finding these matrices. For this purpose, the equation in (5.6) must be checked for each given M_1 and V_1 (see Section 5.3) whose complexity is m^2 *XORs* and m^2 *ANDs*. For each matrix M_1, we need to repeat the process of finding V_1 and checking the equation in (5.6) in order to verify whether they are REA-equivalent or not. The number of this repetition is at most k which is the number of possible matrices M_1. If the equation (5.6) is satisfied some M_1 and V_1, two functions F and G are REA-equivalent of type I. If the equation (5.6) can not be satisfied any M_1 and V_1, they are REA-inequivalent of type I. Consequently, the total complexity of verification for REA-equivalence of F and G is $k(2m^2)$ *XORs*, $k(2m^2)$ *ANDs* and $2^{n+1}+2$ evaluations. This completes the proof of part (i).

The next step is the proof of part (ii). Let now G be arbitrary and $F'(x)_i$ be the i-th bit of $F'(x)$. Denote $img(G')$ the image set of G' and $u_{G'} = |img(G')|$. Let also $N_{G'}$ be any subset of $\mathbb{F}_2^n$ which satisfies $|N_{G'}| = |\{G'(a)|a \in N_{G'}\}| = u_{G'}$. To begin with, we need to compute the values of $F'(x)$ and $G'(x)$ for all $x \in \mathbb{F}_2^n$ whose complexity is 2^{n+1} evaluations as in part (i). To find the matrix M_1, we can construct the system of equations as in (3.2) for each row of M. To construct the system of equations, $u_G{}'$ many $x \in N_{G'}$ out of 2^n must be used. To find i-th row of M for all $i \in \{1,\dots,m\}$, the system of equation can be constructed as follows for all $x_j \in N_{G'}, 1 \le j \le u_{G'}$.

$$\begin{aligned}
F'(x_1)_i &= m_{i,1}G'(x_1)_1 \oplus m_{i,2}G'(x_1)_2 \oplus \cdots \oplus m_{i,m}G'(x_1)_m \\
F'(x_2)_i &= m_{i,1}G'(x_2)_1 \oplus m_{i,2}G'(x_2)_2 \oplus \cdots \oplus m_{i,m}G'(x_2)_m \\
&\vdots \\
F'(x_{u_{G'}})_i &= m_{i,1}G'(x_{u_{G'}})_1 \oplus m_{i,2}G'(x_{u_{G'}})_2 \oplus \cdots \oplus m_{i,m}G'(x_{u_{G'}})_m
\end{aligned} \tag{5.7}$$

To find M_1 it is necessary to solve the system (5.7) for all $i \in \{1,\dots,m\}$. The matrix representation of the system of equations for all $i \in \{1,\dots,m\}$ is as follows:

$$\begin{pmatrix} F'(x_1)_i \\ F'(x_2)_i \\ \vdots \\ F'(x_{u_{G'}})_i \end{pmatrix} = \begin{pmatrix} G'(x_1)_1 & G'(x_1)_2 & \dots & G'(x_1)_m \\ G'(x_2)_1 & G'(x_2)_2 & \dots & G'(x_2)_m \\ \vdots & \vdots & \ddots & \vdots \\ G'(x_{u_{G'}})_1 & G'(x_{u_{G'}})_2 & \dots & G'(x_{u_{G'}})_m \end{pmatrix} \cdot \begin{pmatrix} m_{i,1} \\ m_{i,2} \\ \vdots \\ m_{i,m} \end{pmatrix}, \tag{5.8}$$

equivalently, $F'_i = \mathbf{G}' \cdot M_{1_i}$ where $\mathbf{G}'$ is the coefficients matrix of the system. The systems $[\mathbf{G}'|F'{}_i]$ can be solved in a unique augmented matrix for all $i \in \{1,\cdots,m\}$ similar to (4.5). Therefore, the augmented matrix of the system (5.8) for all rows of M_1 is $[\mathbf{G}'|N_{G'}] = [\mathbf{G}'|F_1{}'|F_2{}'|\dots|F_m{}']$, or equivalently

$$\mathbf{Aug} = \left(\begin{array}{cccc|c|c|c} G'(x_1)_1 & G'(x_1)_2 & \dots & G'(x_1)_m & F'(x_1)_1 & \dots & F'(x_1)_m \\ G'(x_2)_1 & G'(x_2)_2 & \dots & G'(x_2)_m & F'(x_2)_1 & \dots & F'(x_2)_m \\ \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\ G'(x_{u_{G'}})_1 & G'(x_{u_{G'}})_2 & \dots & G'(x_{u_{G'}})_m & F'(x_{u_{G'}})_1 & \dots & F'(x_{u_{G'}})_m \end{array}\right). \tag{5.9}$$

On the other hand, if G is not one-to-one function, it is required to find all sets $N_{G'}$ satisfying $|N_{G'}| = |\{G'(a)|a \in N_{G'}\}| = u_{G'}$ to find all possible matrices M_1. Because, for distinct sets $N_{G'}$, there may be distinct matrices M_1. If the size of the inverse image of any element in the set img(G') is known, then so is the number of the sets $N_{G'}$. The number of the sets $N_{G'}$ is the value k which is the product of the size of the inverse image of all elements in the set $img(G')$. Accordingly, there exist k distinct augmented matrices which have the same coefficient matrix as in (5.9). That is, there exist exactly k distinct systems $\left[\mathbf{G}'|N^l_{G'}\right]$ for all $l \in \{1, \cdots, k\}$. Furthermore, there may be distinct matrices M_1 corresponding to each set $N^l_{G'}$ if the system $\left[\mathbf{G}'|N^l_{G'}\right]$ is consistent for each l. Thus, the number of all matrices M_1 is less than or equals to k. The systems $\left[\mathbf{G}'|N^l_{G'}\right]$ can be solved in a unique augmented matrix since the same coefficient matrix $\mathbf{G}'$ is used in these systems for all $l \in \{1, \cdots, k\}$. The augmented matrix of the k distinct systems for all $N_{G'}$ sets is

$$\left[\mathbf{G}'|N^1_{G'}|N^2_{G'}|\ldots|N^k_{G'}\right], \tag{5.10}$$

which can be solved by simultaneous application of the Gaussian elimination method. Therefore, the complexity of solving the system (5.10) with simultaneous Gaussian elimination method is $2^n(\frac{m^2(2k+1)+m}{2}) - \frac{m^2(2k+1)+m}{2}$ *XORs*. One can easily obtain this value by substituting $t = km + m$ into (4.2). This complexity together with 2^{n+1} evaluations give the complexity of finding the possible matrices M_1. Similar to part (i), the total complexity of computing V_1 and checking process for all possible matrices M_1 is $k(2m^2)$ *XORs* $k(2m^2)$ *ANDs* and 2 evaluations. Therefore, the total complexity of verification for REA-equivalence of F and G is $2^n(\frac{m^2(2k+1)+m}{2}) + \frac{m^2(2k-1)-m}{2}$ *XORs*, $k(2m^2)$ *ANDs* and $2^{n+1} + 2$ evaluations. □

The total complexity of type I is demonstrated for $6 \leq m = n \leq 9$ and $k = 1$ in the Table 5.2.

Table 5.2: The complexity of type I

$k = 1$	part(i)			part(ii)		
$n = m$	*#XOR*	*#AND*	*#Evaluation*	*#XOR*	*#AND*	*#Evaluation*
6	72	72	130	3663	72	130
7	98	98	258	9877	98	258
8	128	128	514	25628	128	514
9	162	162	1026	64548	162	1026

Remark 5.1.3 *The reason why we use only the elements of the set $N_{G'}$ in $\mathbb{F}_2^n$ to construct the equation system in (5.7) is the following: if we use $x_0 \in N_{G'}$ and $x_1 \notin N_{G'}$ (i.e. $G'(x_0) =$*

$G'(x_1)$), there exist two cases:

$$\begin{cases} \textit{If } F'(x_0) \neq F'(x_1), & \textit{the system is inconsistent.} \\ \textit{If } F'(x_0) = F'(x_1), & \textit{some row operations would be unnecessary.} \end{cases}$$

These two cases are undesirable cases.

Example 5.1.4 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_1x_2 \oplus x_2x_3 \oplus 1, x_1x_2 \oplus x_3 \oplus 1, x_3 \oplus x_1x_2)$ and $G(x) = (x_1x_3 \oplus x_1x_2, x_1x_2 \oplus x_2 \oplus 1, x_1x_3 \oplus x_3 \oplus 1)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-inequivalent of type I.*

Proof. REA-equivalence of type I can be written in the following form

$$F(0) = M_1 \cdot G(0) + V_1$$
$$F'(x) = M_1 \cdot G'(x),$$

where $F'(x) = (x_1x_2 \oplus x_2x_3, x_1x_2 \oplus x_3, x_3 \oplus x_1x_2)$, $G'(x) = (x_1x_3 \oplus x_1x_2, x_1x_2 \oplus x_2, x_1x_3 \oplus x_3)$, $F(0) = (1, 1, 0)$ and $G(0) = (0, 1, 1)$. We first check whether part (i) of type I is satisfied or not. For this purpose, we calculate $G'(x)$ and $F'(x)$ for all $x \in \mathbb{F}_2^n$ in the Table 5.3.

Table 5.3: Calculating the F' and G' in Example 5.1.4

$x = (x_3, x_2, x_1)$	$F'(x)$	$G'(x)$	2^i
(0,0,0)	(0,0,0)	(0,0,0)	
(0,0,1)	(0,0,0)	(0,0,0)	
(0,1,0)	(0,0,0)	(0,1,0)	2^1
(0,1,1)	(1,1,1)	(1,0,0)	2^0
(1,0,0)	(0,1,1)	(0,0,1)	2^2
(1,0,1)	(0,1,1)	(1,0,0)	2^0
(1,1,0)	(1,1,1)	(0,1,1)	
(1,1,1)	(0,0,0)	(0,0,0)	

It is easy to see in the Table 5.3 that G' satisfies part (i) since $2^i \in img(G')$ for $i = 0, 1, 2$. As a result, the possible matrices M_1 can be found by means of $F'(x) = M_1 \cdot 2^i = cols_{M_1}(i)$ such that $G'(x) = 2^i$ for $i = 0, 1, 2$. So, there exist 2 distinct possible matrices M_1:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

We continue with finding V_1 for matrix $M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$. The values 0 and M_1 are substituted into the equation $F(0) = M_1 \cdot G(0) + V_1$ to find V_1. It is easy to see that $V_1 = F(0) + M_1 \cdot G(0) = (1, 0, 1)$. Finally, we need to check whether the equation $F(x) = M_1 \cdot G(x) + V_1$ for given M_1, V_1 holds. For this purpose, we define $K(x) := M_1 \cdot G(x) + V_1$ and substitute the values M_1, V_1 into $K(x)$,

$$\begin{aligned} K(x) &:= \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1x_3 \oplus x_1x_2 \\ x_1x_2 \oplus x_2 \oplus 1 \\ x_1x_3 \oplus x_3 \oplus 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\ &= (x_1x_3 \oplus x_1x_2 \oplus 1, x_3 \oplus x_1x_2 \oplus 1, x_3 \oplus x_1x_2). \end{aligned}$$

Since $K(x) \neq F(x)$, F is REA-inequivalent to G for this matrix M_1. We can not say that F is REA-inequivalent to G for type I before we also check the other matrix $M_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$. The process of finding V_1 and checking operation are repeated for this matrix M_1. Similarly, it is easily seen that $K(x) \neq F(x)$. The function F is REA-inequivalent to G for this matrix M_1. Thus, we say that F is REA-inequivalent to G for type I. □

Example 5.1.5 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_2 \oplus x_3 \oplus 1, x_2 \oplus 1, x_1x_3)$ and $G(x) = (x_1 \oplus x_2 \oplus 1, x_2, x_1x_3 \oplus 1)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-inequivalent of type I.*

Proof. REA-equivalence of type I can be written in the following form

$$\begin{aligned} F(0) &= M_1 \cdot G(0) + V_1 \\ F'(x) &= M_1 \cdot G'(x), \end{aligned}$$

where $F'(x) = (x_2 \oplus x_3, x_2, x_1x_3)$ and $G'(x) = (x_1 \oplus x_2, x_2, x_1x_3)$, $F(0) = (1, 1, 0)$ and $G(0) = (1, 0, 1)$. First, it should be checked whether part (i) is satisfied or not. For this purpose, we calculate $G'(x)$ and $F'(x)$ for all $x \in \mathbb{F}_2^n$ in the Table 5.4. It is obvious that G' does not satisfy part (i) since $2^i \notin img(G')$ for $i = 2$ as it is seen in the Table 5.4. The functions F and G should be interchanged if one of them does not satisfy part (i). Similarly, F' does not satisfy part (i) since $2^i \notin img(F')$ for $i = 2$. So, it is required to use part (ii). We first constitute the

Table 5.4: Calculating the F' and G' in Example 5.1.5

$x = (x_3, x_2, x_1)$	$F'(x)$	2^i	$G'(x)$	2^i
(0,0,0)	(0,0,0)		(0,0,0)	
(0,0,1)	(0,0,0)		(1,0,0)	2^0
(0,1,0)	(1,1,0)		(1,1,0)	
(0,1,1)	(1,1,0)		(0,1,0)	2^1
(1,0,0)	(1,0,0)	2^0	(0,0,0)	
(1,0,1)	(1,0,1)		(1,0,1)	
(1,1,0)	(0,1,0)	2^1	(1,1,0)	
(1,1,1)	(0,1,1)		(0,1,1)	

sets $N_{G'} \subseteq \mathbb{F}_2^3$ by taking into account the elements of $img(G')$.

$$\begin{aligned}
S-box_{G'} &= [0,0,3,3,1,5,2,6] \\
img(G') &= [1,2,5,6,0,3] = \{(1,0,0),(0,1,0),(1,0,1),(0,1,1),(0,0,0),(1,1,0)\} \\
N_{G'}^1 &= [4,6,5,7,0,2] = \{(0,0,1),(0,1,1),(1,0,1),(1,1,1),(0,0,0),(0,1,0)\} \\
N_{G'}^2 &= [4,6,5,7,0,3] = \{(0,0,1),(0,1,1),(1,0,1),(1,1,1),(0,0,0),(1,1,0)\} \\
N_{G'}^3 &= [4,6,5,7,1,2] = \{(0,0,1),(0,1,1),(1,0,1),(1,1,1),(1,0,0),(0,1,0)\} \\
N_{G'}^4 &= [4,6,5,7,1,3] = \{(0,0,1),(0,1,1),(1,0,1),(1,1,1),(1,0,0),(1,1,0)\}
\end{aligned}$$

There exist 4 distinct $N_{G'}$ sets. Hence, the system of equations can be constructed by using the sets $N_{G'}$. The augmented matrix of the systems is $\left[\mathbf{G}'|N_{G'}^1|N_{G'}^2|N_{G'}^3|N_{G'}^4\right]$:

$$\mathbf{Aug} = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}\right).$$

The augmented matrix can be reduced to row reduced echelon form by using simultaneous

Gaussian elimination method:

$$\mathbf{Aug}' = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right). \tag{5.11}$$

Hence, it is easy to see that there exists a unique matrix $M_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ from (5.11). We continue with finding V_1 for matrix M_1. It is easy to see that $V_1 = F(0) + M_1 \cdot G(0) = (1, 0, 0)$. Finally, we need to check whether the equation $F(x) = M_1 \cdot G(x) + V_1$ for given M_1, V_1 holds. We define $K(x) := M_1 \cdot G(x) + V_1$ and substitute the values M_1, V_1 into $K(x)$,

$$K(x) := \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \oplus x_2 \oplus 1 \\ x_2 \\ x_1x_3 \oplus 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = (1, x_1 + 1, x_1 \oplus x_2 \oplus x_1x_3).$$

Since $K(x) \neq F(x)$, F is REA-inequivalent to G for type I. □

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalence of type I in Algorithm 2.

Proposition 5.1.6 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be vectorial Boolean functions and G be a permutation. Then the complexity of checking F and G for REA-equivalence of type II ($F(x) = G(M_2 \cdot x + V_2)$) is n^2 XORs, n^2 ANDs and $n + 3$ evaluations.*

Proof. The equation $F(x) = G(M_2 \cdot x + V_2)$ can be written as $G^{-1}(F(x)) = M_2 \cdot x + V_2$ since G is a permutation. The value $G^{-1}(F(x))$ can be easily computed whose complexity is 2 evaluations. Denote $H(x) = G^{-1}(F(x))$. Using (5.3), the equation $H(x) = M_2 \cdot x + V_2$ can be rewritten as follows

$$H(x) = H'(x) + H(0) = M_2 \cdot x + V_2$$

Algorithm 2 REA-equivalence of Type I, $F(x) = M_1 \cdot G(x) + V_1$

```
Require: $F(x) = F'(x) + F(0), G(x) = G'(x) + G(0)$.
Ensure: True if F is REA-equivalent to G.
 1: for $i := 0$ to $m - 1$ do
 2:     Find $K_{[i+1]} = \left\{y \in \mathbb{F}_{2^n} | G'(y) = 2^i\right\}$
 3: end for
 4: for $z$ in $K = Car < K_{[1]}, K_{[2]}, \ldots, K_{[m]} >$ do
 5:     for $i := 1$ to $m$ do
 6:         Set Column($M_1, i, F'(z[i])$)
 7:     end for
 8:     $V_1 = M_1 * G(0) + F(0)$
 9:     if $F(x)! = M_1 * G(x) + V_1$ then
10:         goto next $z$
11:     end if
12:     return True
13: end for
14: return False
```

and it gives the system of equations

$$\begin{cases} H'(x) = M_2 \cdot x, \\ H(0) = V_2. \end{cases} \tag{5.12}$$

The column vector V_2 can be found easily from the second equation of the system (5.12) by computing the value $H(0)$ with 1 evaluation. Moreover, the method and the complexity of finding n by n matrix M_2 from the first equation of the system (5.12) are similar to finding the matrix corresponding to the linear function as in Proposition 4.2.3. For $x = 2^i$ find M_2 by $H'(2^i) = M_2 \cdot 2^i = cols_{M_2}(i)$ for all $i \in \{0, 1, \ldots, n-1\}$ with n evaluations of the function H'. Therefore, the complexity of finding matrix M_2 is n evaluations. In addition, the equation

$$H(x) = M_2 \cdot x + V_2$$

must be checked for given M_2 and V_2 whose complexity is n^2 *XORs*, n^2 *ANDs*. Consequently, the total complexity of verification for REA-equivalence of F and G is n^2 *XORs*, n^2 *ANDs* and $n + 3$ evaluations. □

The total complexity of type II is demonstrated for $6 \leq m = n \leq 9$ in the Table 5.5.

Example 5.1.7 *Let $F, G : \mathbb{F}_2^2 \longrightarrow \mathbb{F}_2^2$ be vectorial Boolean functions and $F(x) = (x_1x_2 \oplus 1, x_1)$*

Table 5.5: The complexity of type II

$n = m$	#XOR	#AND	#Evaluation
6	36	36	9
7	49	49	10
8	64	64	11
9	81	81	12

and $G(x) = (x_1 \oplus x_2, x_2 \oplus 1)$, *where* $x = (x_2, x_1) \in \mathbb{F}_2^2$. *Functions* F *and* G *are REA-inequivalent of type II.*

Proof. The inverse function $H(x) = G^{-1}(F(x)) = (x_1 \oplus 1, x_1 \oplus x_1 x_2)$ can be computed since G is a permutation. The equation $H(x) = M_2 \cdot x + V_2$ can be rewritten as follows

$$\begin{aligned} H'(x) &= M_2 \cdot x \\ H(0) &= V_2 \end{aligned} \tag{5.13}$$

where $H'(x) = (x_1, x_1 \oplus x_1 x_2)$, $H(0) = (1, 0)$. The values $2^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $2^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ are substituted into $H'(x)$ since $H'(2^i) = M_2 \cdot 2^i = cols_{M_2}(i)$ for $i = 0, 1$ as in Proposition 4.2.3.

$$H'(2^0) = H'(1, 0) = (0, 0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = M_{2_1} = cols_{M_2}(1),$$

$$H'(2^1) = H'(0, 1) = (1, 1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = M_{2_2} = cols_{M_2}(2).$$

Thus, the 2×2 matrix $M_2 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$ is obtained. Finally, we need to check whether the equation $H(x) = M_2 \cdot x + V_2$ for M_2 and V_2 holds. We define $K(x) := M_2 \cdot x + V_2$ and substitute M_2 and V_2 into $K(x)$:

$$K(x) = M_2 \cdot x + V_2 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (x_1 \oplus 1, x_1).$$

Since $K(x) \neq H(x)$, The function F is REA-inequivalent to G for type II. □

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalence of type II in Algorithm 3.

Algorithm 3 REA-equivalence of Type II, $F(x) = G(M_2 \cdot x + V_2)$

```
Require: $H(x) := G^{-1}(F(x)), H(x) = H'(x) + H(0)$.
Ensure: True if F is REA-equivalent to G.
1: for $i := 0$ to $n - 1$ do
2:     Set Column($M_2, i + 1, H'(2^i)$)
3: end for
4: $V_2 = H(0)$
5: if $F(x) = G(M_2 * x + V_2)$ then
6:     return True
7: end if
8: return False
```

Proposition 5.1.8 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions. Then the complexity of checking F and G for REA-equivalence of type III ($F(x) = G(x) + M_3 \cdot x + V_1$) is $mn + m$ XORs, mn ANDs and $n + 1$ evaluations.*

Proof. Denote $H(x) = F(x) + G(x)$ whose complexity is *m XORs*. Then REA-equivalence of type III, $F(x) = G(x) + M_3 \cdot x + V_1$, takes the form

$$H(x) = M_3 \cdot x + V_1. \tag{5.14}$$

Using (5.3), the equation (5.14) can be rewritten as follows:

$$\begin{cases} H'(x) = M_3 \cdot x, \\ H(0) = V_1. \end{cases} \tag{5.15}$$

We have the same situation as in Proposition 5.1.6. Thus, the column vector V_1 can be found easily by means of the value $H(0) = V_1$ with 1 evaluation. For $x = 2^i$ find M_3 by $H'(2^i) = M_3 \cdot 2^i = cols_{M_3}(i)$ for all $i \in \{0, 1, \ldots, n-1\}$ with n evaluations of the function H'. In additions, the equation

$$H(x) = M_3 \cdot x + V_1$$

must be checked for given M_3 and V_1 whose complexity is *mn XORs* and *mn ANDs*. Consequently, the total complexity of verification for the equivalence of F and G is *mn + m XORs*, *mn ANDs* and $n + 1$ evaluations. □

The total complexity of type III is demonstrated for $6 \leq m = n \leq 9$ in the Table 5.6.

Table 5.6: The complexity of type III

$n = m$	*#XOR*	*#AND*	*#Evaluation*
6	42	36	7
7	56	49	8
8	72	64	9
9	90	81	10

Example 5.1.9 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial Boolean functions and $F(x) = (x_1 \oplus 1, x_2 \oplus x_3, x_1 \oplus x_3)$ and $G(x) = (x_2, x_1 \oplus x_2, x_3 \oplus 1)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-equivalent of type III.*

Proof. When we write $H(x) = F(x) + G(x)$, then REA-equivalence of type III takes the form $H(x) = (x_1 + x_2 + 1, x_1 + x_3, x_1 + 1)$. The equation $H(x) = M_3 \cdot x + V_1$ can be rewritten as follows

$$H'(x) = M_3 \cdot x$$
$$H(0) = V_1,$$

where $H'(x) = (x_1 \oplus x_2, x_1 \oplus x_3, x_1)$ and $H(0) = (1, 0, 1)$. The columns of M_3:

$$H'(2^0) = H'(1,0,0) = (0,1,0) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = M_{3_1} = cols_{M_3}(1),$$

$$H'(2^1) = H'(0,1,0) = (1,0,0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = M_{3_2} = cols_{M_3}(2),$$

$$H'(2^2) = H'(0,0,1) = (1,1,1) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = M_{3_3} = cols_{M_3}(3).$$

Thus, the 3×3 matrix $M_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ is observed. Finally, it is easily seen that

$$\begin{aligned} K(x) &:= M_3 \cdot x + V_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} x_1 \oplus x_2 + 1 \\ x_1 \oplus x_3 \\ x_1 + 1 \end{pmatrix} = (x_1 \oplus x_2 \oplus 1, x_1 \oplus x_3, x_1 \oplus 1)\,. \end{aligned}$$

Hence, F is REA-equivalent to G for type III since $K(x) = H(x)$. □

Example 5.1.10 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial Boolean functions and $F(x) = (x_3 \oplus 1, x_2x_1, x_1)$ and $G(x) = (x_3 \oplus x_2, x_2, x_3x_1)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-inequivalent of type III.*

Proof. Compute $H(x) := F(x) + G(x) = (x_2 + 1, x_2 \oplus x_1x_2, x_1 \oplus x_3x_1)$. The equation $H(x) = M_3 \cdot x + V_1$ can be rewritten as follows

$$\begin{aligned} H'(x) &= M_3 \cdot x \\ H(0) &= V_1, \end{aligned}$$

where $H'(x) = (x_2, x_2 \oplus x_2x_1, x_1 \oplus x_3x_1)$ and $H(0) = (1, 0, 0)$.

To find columns of M_3, the values $2^0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $2^1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, $2^2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ are substituted into $H'(x)$:

Algorithm 4 REA-equivalence of Type III, $F(x) = G(x) + M_3 \cdot x + V_1$

```
Require: $H(x) := F(x) + G(x)$, $H(x) = H'(x) + H(0)$
Ensure: True if F is REA-equivalent to G
1: for $i := 0$ to $n - 1$ do
2:     Set Column($M_3, i + 1, H'(2^i)$)
3: end for
4: $V_1 = K(0)$
5: if $F(x) = G(x) + M_3 * x + V_1$ then
6:     return True
7: end if
8: return False
```

$$H'(2^0) = H'(1,0,0) = (0,0,0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = M_{3_1} = cols_{M_3}(1),$$

$$H'(2^1) = H'(0,1,0) = (1,1,0) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = M_{3_2} = cols_{M_3}(2),$$

$$H'(2^2) = H'(0,0,1) = (0,0,1) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = M_{3_3} = cols_{M_3}(3).$$

Thus, the 3×3 matrix $M_3 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is obtained. For the check operation, we define $K(x) := M_3 \cdot x + V_1$ for given M_3 and V_1, and calculate

$$K(x) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = (x_2 \oplus 1, x_2, x_1).$$

The function F is REA-inequivalent to G for type III since $K(x) \neq H(x)$. □

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalence of type III in Algorithm 4.

We note that every vectorial Boolean function can be written in the form

$$H(x) = H'(x) + L_H(x) + H(0), \tag{5.16}$$

where H' has terms of algebraic degree at least 2 and L_H is a linear function.

Proposition 5.1.11 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions and G be defined by (5.16). Then the complexity of checking F and G for REA-equivalence of type IV ($F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$) is*

i. $k(nm^2 + 2m^2 + mn)$ XORs, $k(nm^2 + 2m^2 + mn)$ ANDs and $2^{n+1} + 2n + 2$ evaluations in case the set $V = \{2^i | 0 \le i \le m-1\} \subseteq img(G')$, where $k = \prod_{v \in V} k_{G',v}$.

ii. $2^n(\frac{m^2(2k+1)+m}{2}) + \frac{k(2m^2n+2mn+2m^2)-m^2-m}{2}$ XORs, $k(nm^2 + 2m^2 + mn)$ ANDs and $2^{n+1} + 2n + 2$ evaluations in case G is arbitrary, where $k = \prod_{v \in img(G')} k_{G',v}$.

Proof. Using (5.16), REA-equivalence of type IV can be rewritten as follows

$$F'(x) + L_F(x) + F(0) = M_1 \cdot G'(x) + M_1 \cdot L_G(x) + M_3 \cdot x + M_1 \cdot G(0) + V_1 \tag{5.17}$$

and the system of equations can be obtained from (5.17):

$$\begin{cases} F'(x) = M_1 \cdot G'(x), \\ L_F(x) = M_1 \cdot L_G(x) + M_3 \cdot x, \\ F(0) = M_1 \cdot G(0) + V_1. \end{cases} \tag{5.18}$$

To begin with, it is required to find the matrix M_1 from the first equation of the system (5.18) before finding M_3, V_1. The first equation of the system leads to two different cases for the function G' considered in Proposition 5.1.2. First, we prove part (i). The method and the complexity of finding m by m matrix M_1 are the same as part (i) in Proposition 5.1.2. Thus, the complexity of finding the possible matrices M_1 is 2^{n+1} evaluations. The number of possible matrices M_1 is the value k which is the product of the size of the inverse image of elements 2^i in the set $img(G')$ for all $i \in \{0, \cdots, m-1\}$. For each given matrix M_1, the matrix M_3 and the vector V_1 can be found from the second and third equation of the system (5.18), respectively. For $x = 2^i$ find M_3 by $L_F(2^i) + M_1 \cdot L_G(2^i) = M_3 \cdot 2^i = cols_{M_3}(i)$ for all $i \in \{0, 1, \ldots, n-1\}$ whose complexity is $n(m^2 - m) + mn$ *XORs*, nm^2 *ANDs* and $2n$ evaluations. Moreover,$V_1 = F(0) + M_1 \cdot G(0)$ can be computed for $x = 0$ with complexity is m^2 *XORs*,

m^2 *ANDs* and 2 evaluations. We need to verify whether or not the matrices M_1, M_3 and V_1 satisfy the equation

$$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1, \tag{5.19}$$

as well as finding these matrices. For this purpose, the equation in (5.19) must be also checked for given M_1, M_3 and V_1 (see Section 5.3) whose complexity is $(m^2+mn-2m)+2m$ *XORs* and $m^2 + mn$ *ANDs*. For each M_1, we need to repeat the process of finding V_1, M_3 and checking the equation (5.19) in order to verify whether they are REA-equivalent or not. The number of this repetition is at most k which is the number of possible matrices M_1. If the equation (5.19) is satisfied some M_1, M_3 and V_1, then we say that the functions F and G are REA-equivalent of type IV. If the equation (5.19) can not be satisfied any M_1, M_3 and V_1, they are REA-inequivalent of type IV. Consequently, the total complexity of verification for the equivalence of F and G is $k(nm^2+2m^2+mn)$ *XORs*, $k(nm^2+2m^2+mn)$ *ANDs* and $2^{n+1}+2n+2$ evaluations.

Next, the proof of part (ii) is given. The method and the complexity of finding m by m matrix M_1 are the same as part (ii) in Proposition 5.1.2. Hence, the complexity of finding M_1 is $2^n(\frac{m^2(2k+1)+m}{2}) - \frac{m^2(2k+1)+m}{2}$ *XORs* and 2^{n+1} evaluations, where the value k is the number of the set $N_{G'}$. For each given matrix M_1, one can easily compute M_3 and V_1 from the second and the third equation of the system (5.18) as in part(i), respectively. Moreover, we need to verify whether or not the matrices M_1, M_3 and V_1 satisfy the equation (5.19) as well as finding these matrices. Similar to part (i), the total complexity of computing M_3, V_1 and checking process for all possible matrices M_1 is $k(nm^2+2m^2+mn)$ *XORs*, $k(nm^2+2m^2+mn)$ *ANDs* and $2n+2$ evaluations. Consequently, the total complexity of verification for the equivalence of F and G is $2^n(\frac{m^2(2k+1)+m}{2}) + \frac{k(2m^2n+2mn+2m^2)-m^2-m}{2}$ *XORs*, $k(nm^2 + 2m^2 + mn)$ *ANDs* and $2^{n+1} + 2n + 2$ evaluations, where the value k is the product of the size of the inverse image of all elements in the set $img(G')$. □

The total complexity of type IV is demonstrated for $6 \leq m = n \leq 9$ and $k = 1$ in the Table 5.7.

Corollary 5.1.12 *Let $n = m$ and G be a permutation. Then the complexity of checking F and G for REA-equivalence are $2n^2$ XORs, $2n^2$ ANDs and $2^n + n + 2$ evaluations for type I, and $n^3 + 3n^2$ XORs, $n^3 + 3n^2$ ANDs and $2^n + 3n + 2$ evaluations for type IV.*

Proof. The function G' satisfies part (i) of Proposition 5.1.11 and the value k is equals to 1

Table 5.7: The complexity of type IV

$k = 1$	part(i)			part(ii)		
$n = m$	*#XOR*	*#AND*	*#Evaluation*	*#XOR*	*#AND*	*#Evaluation*
6	324	324	142	3915	324	142
7	490	490	272	10269	490	272
8	704	704	530	26204	704	530
9	972	972	1044	65358	972	1044

since G is a permutation. Hence, the proof follows from part (i) in Proposition 5.1.11 and in Proposition 5.1.2 □

Example 5.1.13 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus 1, x_2 \oplus x_1x_3 \oplus x_2x_3, x_1 \oplus x_2x_3 \oplus 1)$ and $G(x) = (x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2, x_1 \oplus x_3x_2 \oplus 1, x_1 \oplus x_2 \oplus x_3x_1)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-equivalent of type IV.*

Proof. REA-equivalence of type IV can be rewritten as

$$\begin{aligned} F'(x) &= M_1 \cdot G'(x) \\ L_F(x) &= M_1 \cdot L_G(x) + M_3 \cdot x \\ F(0) &= M_1 \cdot G(0) + V_1, \end{aligned}$$

where $F'(x) = (x_1x_2, x_1x_3 \oplus x_2x_3, x_2x_3)$, $G'(x) = (x_1x_2, x_2x_3, x_3x_1)$, $L_F(x) = (x_1 \oplus x_2 \oplus x_3, x_2, x_1)$, $L_G(x) = (x_1 \oplus x_2 \oplus x_3, x_1, x_1 \oplus x_2)$, $F(0) = (1, 0, 1)$ and $G(0) = (0, 1, 0)$. First, it is verified whether G satisfy part(i) or not. We calculate $G'(x)$ and $F'(x)$ for all $x \in \mathbb{F}_2^n$ in the Table 5.8. It is easy to see that G' satisfies part (i) since $2^i \in img(G')$ for all $i = 0, 1, 2$. There

Table 5.8: Calculating the F' and G' in Example 5.1.13

$x = (x_3, x_2, x_1)$	$F'(x)$	$G'(x)$	2^i
(0,0,0)	(0,0,0)	(0,0,0)	
(0,0,1)	(0,0,0)	(0,0,0)	
(0,1,0)	(0,0,0)	(0,0,0)	
(0,1,1)	(1,0,0)	(1,0,0)	2^0
(1,0,0)	(0,0,0)	(0,0,0)	
(1,0,1)	(0,1,0)	(0,0,1)	2^2
(1,1,0)	(0,1,1)	(0,1,0)	2^1
(1,1,1)	(1,0,1)	(1,1,1)	

exists only one matrix $M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$. We continue with finding the matrix M_3 for the matrix M_1. The values $x = 2^i$ and M_1 are substituted into $L_F(x) + M_1 \cdot L_G(x) = M_3 \cdot x$ in order to find the i-th column of M_3 for $i = 0, 1, 2$. For $i = 0$,

$$L_F(1,0,0) + M_1 \cdot L_G(1,0,0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Similarly, for $i = 1$, $L_F(0,1,0) + M_1 \cdot L_G(0,1,0) = (0,0,0)$ and for $i = 2$, $L_F(0,0,1) + M_1 \cdot L_G(0,0,1) = (0,0,0)$. Hence, the 3 by 3 matrix M_3 is as follows:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

It is easy to see that $V_1 = F(0) + M_1 \cdot G(0) = (1,1,0)$. Finally, we need to check whether the equation $F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$ for given M_1, M_3 and V_1 holds. We define $K(x) := M_1 \cdot G(x) + M_3 \cdot x + V_1$ and substitute the values M_1, M_3 and V_1 into $K(x)$,

$$\begin{aligned} K(x) \quad &:= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \\ x_1 \oplus x_3x_2 \oplus 1 \\ x_1 \oplus x_2 \oplus x_3x_1 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\ &= (x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus 1, x_2 \oplus x_1x_3 \oplus x_2x_3, x_1 \oplus x_2x_3 \oplus 1). \end{aligned}$$

The function F is REA-equivalent to G for type IV since $F(x) = K(x)$.

We note that the univariate polynomial representation of these functions in Example 5.1.13 are as follows: $F(x) = x^3 + a^2 + 1$ and $G(x) = x^6 + a$ where a is a primitive element of $\mathbb{F}_2^3$. When these functions are applied to REA-equivalence of types I, IV, V, VI in MAGMA codes given in the Appendix, it is seen that they are REA-equivalent for these types. □

Example 5.1.14 *Let $F, G : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus 1, x_1 \oplus x_3 \oplus x_4 \oplus x_3x_4, x_1 \oplus x_4 \oplus x_1x_3 \oplus 1)$ and $G(x) = (x_3 \oplus x_1x_3 \oplus x_3x_4, x_1 \oplus x_2 \oplus x_3x_4 \oplus 1, x_1 \oplus x_4 \oplus x_1x_3)$, where $x = (x_4, x_3, x_2, x_1) \in \mathbb{F}_2^4$. Functions F and G are REA-inequivalent of type IV.*

Proof. REA-equivalence of type IV can be rewritten as

$$\begin{aligned}
F'(x) &= M_1 \cdot G'(x),\\
L_F(x) &= M_1 \cdot L_G(x) + M_3 \cdot x,\\
F(0) &= M_1 \cdot G(0) + V_1,
\end{aligned}$$

where $F'(x) = (x_1x_3, x_3x_4, x_1x_3)$, $G'(x) = (x_1x_3 \oplus x_3x_4, x_3x_4, x_1x_3)$, $L_F(x) = (x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_4)$, $L_G(x) = (x_3, x_1 \oplus x_2, x_1 \oplus x_4)$, $F(0) = (1, 0, 1)$ and $G(0) = (0, 1, 0)$. It is required to check whether part (i) is satisfied or not. The values $F'(x)$ and $G'(x)$ are calculated for all $x \in \mathbb{F}_2^4$ in the Table 5.9.

Table 5.9: Calculating the F' and G' in Example 5.1.14

$x = (x_4, x_3, x_2, x_1)$	$F'(x)$	2^i	$G'(x)$
(0,0,0,0)	(0,0,0)		(0,0,0)
(0,0,0,1)	(0,0,0)		(0,0,0)
(0,0,1,0)	(0,0,0)		(0,0,0)
(0,0,1,1)	(0,0,0)		(0,0,0)
(0,1,0,0)	(0,0,0)		(0,0,0)
(0,1,0,1)	(1,0,1)		(1,0,1)
(0,1,1,0)	(0,0,0)		(0,0,0)
(0,1,1,1)	(1,0,1)		(1,0,1)
(1,0,0,0)	(0,0,0)		(0,0,0)
(1,0,0,1)	(0,0,0)		(0,0,0)
(1,0,1,0)	(0,0,0)		(0,0,0)
(1,0,1,1)	(0,0,0)		(0,0,0)
(1,1,0,0)	(0,1,0)	2^1	(1,1,0)
(1,1,0,1)	(1,1,1)		(0,1,1)
(1,1,1,0)	(0,1,0)	2^1	(1,1,0)
(1,1,1,1)	(1,1,1)		(0,1,1)

Part (i) can not be satisfied since $2^i \notin img(G') = \{(0, 0, 0), (1, 0, 1), (1, 1, 0), (0, 1, 1)\}$ for $i = 0, 1, 2$ and $2^i \notin img(F') = \{(0, 0, 0), (1, 0, 1), (0, 1, 0), (1, 1, 1)\}$ for $i = 0, 2$ as it is seen in the Table 5.9. We continue with using part (ii). We first constitute the sets $N_{G'}$ by using the elements of $img(G')$. There are 80 distinct sets $N_{G'}$, but there exists a unique value $F'(x)$ for all sets $N_{G'}$, see Table 5.9. It is enough to use only one set $N_{G'}$. The augmented matrix of the systems is $[\mathbf{G}'|N_{G'}]$:

$$\mathbf{Aug} = \left(\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

It is easy to see that there are 8 possible matrices those are solutions to the above augmented system. One of them is

$$M_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Similar to previous example, M_3 and V_1 can be obtained corresponding to M_1:

$$M_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \; V_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

These matrices satisfy the equation $F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$. Thus, F and G are REA-equivalent of type IV. □

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalent of type IV in Algorithm 5.

5.2 Verification of New Types of REA-equivalence

We introduce two new types of REA-equivalence in the Table 5.10. The verification procedures of these types, namely types V and VI, with their complexities and algorithms are presented in this section.

Table 5.10: New types of REA-equivalence

Types	REA-equivalence
V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$
VI	$F(x) = G(M_2 \cdot x) + V_1$

Algorithm 5 REA-equivalence of Type IV, $F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$

```
Require: $F(x) = F'(x) + L_F(x) + F(0), G(x) = G'(x) + L_G(x) + G(0)$
Ensure: True if F is REA-equivalent to G
 1: for $i := 0$ to $m-1$ do
 2:     Find $K_{[i+1]} = \left\{y \in \mathbb{F}_{2^n} | G'(y) = 2^i\right\}$
 3: end for
 4: for $z$ in $K = Car < K_{[1]}, K_{[2]}, \dots, K_{[m]} >$ do
 5:     for $i := 1$ to $m$ do
 6:         Set Column($M_1, i, F'(z[i])$)
 7:     end for
 8:     $V_1 = M_1 * G(0) + F(0)$
 9:     for $i := 0$ to $n-1$ do
10:         Set Column($M_3, i+1, L_F(2^i) + M_1 * L_G(2^i)$)
11:     end for
12:     if $F(x)! = M_1 * G(x) + M_3 * x + V_1$ then
13:         goto next $z$
14:     end if
15:     return True
16: end for
17: return False
```

Proposition 5.2.1 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions and F, G be defined by (5.16). Then the complexity of checking F and G for REA-equivalence of type V ($F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$) is $k(n^2+2mn+m-n)+m$ XORs, $k(n^2+mn)$ ANDs and $2^n+2n+2+k(n+1)$ evaluations in case $V = \{F'(2^i) | 0 \le i \le n-1\} \subseteq img(G')$, where $k = \prod_{v \in V} k_{G',v}$.*

Proof. Using (5.16), REA-equivalence of type V can be rewritten as follows

$$F'(x) + L_F(x) + F(0) = G'(M_2 \cdot x) + L_G(M_2 \cdot x) + M_3 \cdot x + G(0) + V_1. \tag{5.20}$$

The system of equations obtained from (5.20) is as follows

$$\begin{cases} F'(x) = G'(M_2 \cdot x) \\ L_F(x) = L_G(M_2 \cdot x) + M_3 \cdot x \\ F(0) = G(0) + V_1. \end{cases} \tag{5.21}$$

It is obvious that one can easily compute V_1 from the equation $V_1 = F(0) + G(0)$ whose complexity is *m XORs* and 2 evaluations. When the matrix M_2 is achieved from the first equation, the matrix M_3 can be computed easily from the second equation of the system (5.21). To begin with, it is necessary to find M_2 from the equation $F'(x) = G'(M_2 \cdot x)$. It is clear that the set $U = \left\{2^i | 0 \leq i \leq n-1\right\}$ is a subset of $\mathbb{F}_2^n$. To find the i-th column of M_2, $cols_{M_2}(i) = M_{2_i} = M_2 \cdot 2^i$, it is necessary first to multiply M_2 with 2^i next to solve the equation

$$G'(M_{2_i}) = F'(2^i) \tag{5.22}$$

for all $2^i \in U$. To solve the equation (5.22), we need to compute the values $F'(2^i)$ for all $2^i \in U$ and $G'(y)$ for all $y \in \mathbb{F}_2^n$ whose complexity is $2^n + n$ evaluations. Thus, the possible i-th column of M_2 are found by means of $y = M_{2_i}$ which satisfy the equation (5.22) for all $i \in \{0, 1, \cdots, n-1\}$. The number of possible matrices M_2 is the value k which is the product of the number of elements $y \in \mathbb{F}_2^n$ which satisfy the equation $G'(y) = F'(2^i)$ for all $2^i \in U$. For each given matrix M_2, one can easily compute M_3 from the second equation of the system (5.21) by means of $L_F(2^i) = L_G(M_{2_i}) + M_{3_i}$ for all $2^i \in U$. The complexity of finding the matrix M_3 is $2n$ evaluations and *mn XORs*. We need to verify whether or not the matrices M_2, M_3 and V_1 satisfy the equation of type V as well as finding these matrices. For this purpose, the equation

$$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1 \tag{5.23}$$

must be checked for given M_2 M_3 and V_1 (see Section 5.3) whose complexity is $(n^2+mn-m-n)+2m$ *XORs*, n^2+mn *ANDs* and 1 evaluation. For each given matrix M_2, we need to repeat the process of finding M_3 and checking the equation in (5.23) in order to verify whether they are REA-equivalent or not. The number of this repetition is at most k which is the number of possible matrices M_2. If the equation (5.23) is satisfied some M_2, M_3 and V_1, the functions F and G are REA-equivalent of type V. If it does not satisfy any M_2, M_3 and V_1, they are REA-inequivalent of type V. Consequently, the total complexity of verification for equivalence of F and G is $k(n^2+2mn+m-n)+m$ *XORs*, $k(n^2+mn)$ *ANDs* and $2^n+2n+2+k(n+1)$ evaluations. □

The total complexity of type V is demonstrated for $6 \leq m = n \leq 9$ and $k = 1$ in the Table 5.11.

Example 5.2.2 *Let $F, G : \mathbb{F}_2^2 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_2, x_1 \oplus x_2, x_1 \oplus 1)$ and $G(x) = (x_1x_2 \oplus 1, x_1, x_2)$, where $x = (x_2, x_1) \in \mathbb{F}_2^2$. Functions F and G are REA-equivalent of type V.*

Table 5.11: The complexity of type V

$n = m$	*#XOR*	*#AND*	*#Evaluation*
6	114	72	85
7	154	98	152
8	200	128	283
9	252	162	542

Proof. REA-equivalence of type V can be rewritten as follows:

$$F'(x) = G'(M_2 \cdot x)$$
$$L_F(x) = L_G(M_2 \cdot x) + M_3 \cdot x$$
$$F(0) = G(0) + V_1,$$

where $F'(x_2, x_1) = (0, 0, 0)$, $G'(x_2, x_1) = (x_1x_2, 0, 0)$, $L_F(x_2, x_1) = (x_2, x_1 \oplus x_2, x_1)$, $L_G(x_2, x_1) = (0, x_1, x_2)$, $F(0) = (0, 0, 1)$ and $G(0) = (1, 0, 0)$. To find the i-th column M_{2_i} of matrix M_2, we first compute the values $F'(2^i)$ and find $y \in \mathbb{F}_2^2$ satisfying $G'(y) = F(2^i)$ for $i = 0, 1$. For this purpose, we calculate $F'(x)$ and $G'(x)$ for all $x \in \mathbb{F}_2^2$ in the Table 5.12.

Table 5.12: Calculating the F' and G' in Example 5.2.2

2^i	$x = (x_2, x_1)$	$F'(x)$	$G'(x)$	$L_F(x)$	$L_G(x)$
	(0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
2^1	(0,1)	(0,0,0)	(0,0,0)	(0,1,1)	(0,1,0)
2^0	(1,0)	(0,0,0)	(0,0,0)	(1,1,0)	(0,0,1)
	(1,1)	(0,0,0)	(1,0,0)	(1,0,1)	(0,1,1)

The possible first columns of M_2 are $(0, 0), (0, 1), (1, 0)$ and the possible second columns of M_2 are also the same values $(0, 0), (0, 1), (1, 0)$ in the Table 5.12. Hence, there exist 9 distinct possible matrices M_2:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix},$$
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

We continue with finding M_3 for the matrix $M_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. The values $x = 2^i$ and M_2 are substituted into $L_F(x) + L_G(M_2 \cdot x) = M_3 \cdot x$ to find M_{3_i} as in Proposition (4.2.3):

$$L_F(1,0) + L_G\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) = (1,0,0)$$

$$L_F(0,1) + L_G\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = (0,1,0).$$

Hence, the 3 by 2 matrix M_3 is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$. It is easy to see that $V_1 = F(0) + G(0) = (1,0,1)$. Finally, we need to check whether the equation $F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$ for given M_2, M_3, V_1 holds. We define $K(x) := G(M_2 \cdot x) + M_3 \cdot x + V_1$ and substitute the values M_2, M_3, V_1 into $K(x)$,

$$K(x) := G\left(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_1 \end{pmatrix}\right) + \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = (x_2 \oplus x_1x_2, x_1 \oplus x_2, x_1 \oplus 1).$$

Since $K(x) \neq F(x)$, the functions F is REA-inequivalent to G for this matrix M_2. However, we can not say that F is REA-inequivalent to G for type V since there exist 9 distinct cases for type V. Another matrix M_2 is used to find the matrix M_3. Similarly, the matrix $M_3 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$ is found for the other matrix $M_2 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ by using the same method. Finally, we need to check whether the equation $F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$ for given M_2, M_3, V_1 holds. When the values M_2, M_3, V_1 are substituted into $K(x)$,

$$K(x) := G\left(\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_1 \end{pmatrix}\right) + \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = (x_2, x_1 \oplus x_2, x_1 \oplus 1).$$

Therefore, the function F is REA-equivalent to G for type V since $F(x) = K(x)$. $\square$

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalent of type V in Algorithm 6.

Algorithm 6 REA-equivalence of Type V, $F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$

```
Require: $F(x) = F'(x) + L_F(x) + F(0), G(x) = G'(x) + L_G(x) + G(0)$
Ensure: True if F is REA-equivalent to G
 1: for $i := 0$ to $n - 1$ do
 2:     Find $K_{[i+1]} = \left\{y \in \mathbb{F}_{2^n} | G'(y) := F'(2^i)\right\}$
 3: end for
 4: for $z$ in $K = Car < K_{[1]}, K_{[2]}, \ldots, K_{[n]} >$ do
 5:     for $i := 1$ to $n$ do
 6:         Set Column($M_2, i, z[i]$)
 7:     end for
 8:     $V_1 = F(0) + G(0)$
 9:     for $i := 0$ to $n - 1$ do
10:         Set Column($M_3, i + 1, L_F(2^i) + L_G(M_2 * 2^i)$)
11:     end for
12:     if $F(x)! = G(M_2 * x) + M_3 * x + V_1$ then
13:         goto next $z$
14:     end if
15:     return True
16: end for
17: return False
```

Proposition 5.2.3 *Let $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions and F, G be defined by (5.3). Then the complexity of checking F and G for REA-equivalence of type VI ($F(x) = G(M_2 \cdot x) + V_1$) is $k(n^2 - n + m) + m$ XORs, kn^2 ANDs and $2^n + n + 2 + k$ evaluations in case $V = \{F'(2^i) | 0 \leq i \leq n - 1\} \subseteq img(G')$, where $k = \prod_{v \in V} k_{G',v}$.*

Proof. Using (5.3), REA-equivalent of type VI can be rewritten as $F'(x) + F(0) = G'(M_2 \cdot x) + G(0) + V_1$ and it gives the following form

$$\begin{cases} F'(x) = G'(M_2 \cdot x) \\ F(0) = G(0) + V_1. \end{cases} \tag{5.24}$$

It is obvious that one can easily compute V_1 from the second equation of (5.24) as in Proposition 5.1.6 whose complexity is *m XORs* and 2 evaluations. The first equation in (5.24) can be solved by the same method as in Proposition 5.2.1 to find M_2. Thus, the possible matrices M_2 can be found whose complexity $2^n + n$ evaluations. The number of possible matrices M_2 is k. For each given matrix M_2, we need to verify whether or not the matrices M_2 and V_1 satisfy the equation of type VI as well as finding these matrices. For this purpose, the equation

$$F(x) = G(M_2 \cdot x) + V_1$$

must be also checked for given matrices M_2 and V_1 (see Section 5.3) whose complexity is $k(n^2 - n + m)$ *XORs*, kn^2 *ANDs* and k evaluations for all possible matrices M_2. Consequently, the total complexity of verification for equivalence of F and G is $k(n^2 - n + m) + m$ *XORs*, kn^2 *ANDs* and $2^n + n + 2 + k$ evaluations. □

The total complexity of type VI is demonstrated for $6 \leq m = n \leq 9$ and $k = 1$ in the Table 5.13.

Table 5.13: The complexity of type VI

$n = m$	#XOR	#AND	#Evaluation
6	42	36	73
7	56	49	138
8	72	64	267
9	90	81	524

We should say that two vectorial Boolean functions $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ can be checked whether they are REA-equivalent of types V and VI in case G is a permutation. However, if G is not a permutation, these functions may not be applied to these types.

Corollary 5.2.4 *Let $m = n$ and G be a permutation. Then the complexity of checking F and G for REA-equivalence is $3n^2 + n$ XORs, $2n^2$ ANDs and $2^n + 3n + 3$ evaluations for type V, and $n^2 + n$ XORs, n^2 ANDs and $2^n + n + 3$ evaluations for type VI.*

Proof. Since G is a permutation, the value k equals to 1. Hence, the proof follows from part (i) in Proposition 5.2.1 and 5.2.3, respectively. □

Example 5.2.5 *Let $F, G : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$ be vectorial boolean functions and $F(x) = (x_1 \oplus x_2, x_3, x_2 \oplus 1)$ and $G(x) = (x_1, x_3 \oplus 1, x_2)$, where $x = (x_3, x_2, x_1) \in \mathbb{F}_2^3$. Functions F and G are REA-equivalent of type VI ($F(x) = G(M_2 \cdot x) + V_1$).*

Proof. REA-equivalence of type VI can be rewritten as follows

$$F'(x) = G'(M_2 \cdot x)$$
$$F(0) = G(0) + V_1,$$

where $F'(x_3, x_2, x_1) = (x_1 \oplus x_2, x_3, x_2)$, $G'(x_3, x_2, x_1) = (x_1, x_3, x_2)$, $F(0) = (0, 0, 1)$ and $G(0) = (0, 1, 0)$. To find M_{2_i}, we first need to compute the values $F'(2^i)$ and find $x \in \mathbb{F}_2^3$ which satisfy $G'(x) = F'(2^i)$ for $i = 0, 1, 2$. For this purpose, we calculate the values $F'(x)$ and $G'(x)$ for all $x \in \mathbb{F}_2^3$ in the Table 5.14. Hence, it can be easily seen in the Table 5.14 that there exists

Table 5.14: Calculating the F' and G' in Example 5.2.5

$x = (x_3, x_2, x_1)$	$F'(x)$	$F(2^i)$	$G'(x)$
(0,0,0)	(0,0,0)		(0,0,0)
(0,0,1)	(1,0,0)	$F(2^2)$	(1,0,0)
(0,1,0)	(1,0,1)	$F(2^1)$	(0,0,1)
(0,1,1)	(0,0,1)		(1,0,1)
(1,0,0)	(0,1,0)	$F(2^0)$	(0,1,0)
(1,0,1)	(1,1,0)		(1,1,0)
(1,1,0)	(1,1,1)		(0,1,1)
(1,1,1)	(0,1,1)		(1,1,1)

only one matrix $M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$.

It is easy to see that $V_1 = F(0) + G(0) = (0, 1, 1)$. Finally, we need to check whether the equation $F(x) = G(M_2 \cdot x) + V_1$ for given M_2, V_1 holds. We define $K(x) := G(M_2 \cdot x) + V_1$ and substitute the values M_2, V_1 into $K(x)$;

$$K(x) := G\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix}\right) + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = (x_2 \oplus x_1, x_3, x_2 \oplus 1).$$

Therefore, the function F is REA-equivalent to G for type VI since $F(x) = K(x)$. □

Remark 5.2.6 *We note that the functions in Example 5.2.2 can not be verified for REA-equivalence of type VI since the functions do not satisfy the assumptions of Proposition 5.2.3. Moreover, we should say that the complexity of verification for REA-equivalence of type V for functions in Example 5.2.5 is extremely higher than the complexity of verification for REA-equivalence of type VI for the same functions. For instance, if one checks functions given in*

Algorithm 7 REA-equivalence of Type VI, $F(x) = G(M_2 \cdot x) + V_1$

```
Require: $F(x) = F'(x) + F(0), G(x) = G'(x) + G(0)$
Ensure: True if F is REA-equivalent to G
 1: for $i := 0$ to $n - 1$ do
 2:     Find $K_{[i+1]} = \left\{y \in \mathbb{F}_{2^n} | G'(y) := F'(2^i)\right\}$
 3: end for
 4: for $z$ in $K = Car < K_{[1]}, K_{[2]}, \ldots, K_{[n]} >$ do
 5:     for $i := 1$ to $n$ do
 6:         Set Column$(M_2, i, z[i])$
 7:     end for
 8:     $V_1 = F(0) + G(0)$
 9:     if $F(x)! = G(M_2 * x) + V_1$ then
10:         goto next $z$
11:     end if
12:     return True
13: end for
14: return False
```

Example 5.2.5 for REA-equivalence of type VI, then it is obtained that $k = 1$; on the other hand, if one checks the same functions for REA-equivalence of type V, then it is obtained that $k = 2^{24}$. These results show that type VI is a particular case of type V, but sometimes type VI may have significantly low complexity compared to type V.

We present the pseudocode of verification procedure of two vectorial Boolean functions F and G for REA-equivalence of type VI in Algorithm 7.

5.3 Summary of Complexities

To verify whether F and G are REA-equivalent or not, Examples 5.1.4, 5.1.5, 5.1.7, 5.1.10 and 5.1.14 given in this chapter clearly show that finding the matrices M and V seems to be not enough. In additions to that, it is necessary to check whether the matrices M and V satisfy the equation of the corresponding type even if the matrices M and V are unique. On the other hand, the authors in [7] give the complexity of just finding the matrices M and V without checking the equation except for part (i) of types I and IV. Furthermore, Example 5.2.2 illustrates that checking REA-equivalence of any two functions for only one $x \in \mathbb{F}_2^n$

which satisfies $G'(x) = F'(2^i)$ for any $i = 0, 1, 2$ is not enough. Thus, there may exist another matrix M for which F and G are REA-equivalent for Types I, IV, V and VI. On the other hand, the authors in [7] follow a procedure without taking into account that the matrix M may be more than one. We will further investigate this difference.

Linear and affine equivalences when F and G are two permutations are studied in [2]. We present the results of [2] in the Table 5.15.

Table 5.15: The complexities of the linear and affine equivalences in [2]

	REA-equivalence	Complexity	F, G
Linear Equivalence	$F(x) = M_1 \cdot G(M_2 \cdot x)$	$O(n^2 2^n)$	Permutations
Affine Equivalence	$F(x) = M_1 \cdot G(M_2 \cdot x + V_2) + V_1$	$O(n^2 2^{2n})$	Permutations

Some REA-equivalence types when G is arbitrary or under some condition on G are studied in [7]. We expose the complexities of these REA-equivalence types. The complexities of REA-equivalence types in [7] are given in the Table 5.16. We also present our results for these REA-equivalence types in the Table 5.17.

Table 5.16: The complexities of REA-equivalence types in [7]

Types	REA-equivalence	Complexity	G
I	$F(x) = M_1 \cdot G(x) + V_1$	$O(2^{n+1})$	$\{2^i \| 0 \leq i \leq m-1\} \subseteq img(G')$
I	$F(x) = M_1 \cdot G(x) + V_1$	$O(m2^{2n})$	Arbitrary
II	$F(x) = G(M_2 \cdot x + V_2)$	$O(n)$	Permutation
III	$F(x) = G(x) + M_3 \cdot x + V_1$	$O(n)$	Arbitrary
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$O(2^{n+1})$	$\{2^i \| 0 \leq i \leq m-1\} \subseteq img(G')$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$O(m2^{2n})$	Arbitrary

Moreover, we study two new REA-equivalence types when G is under some condition with their complexities. The complexities for new REA-equivalence types are presented in the Table 5.18.

Table 5.17: Our complexities of REA-equivalence types in [7]

		Complexity in case G Arbitrary		
Type	REA-equivalence	$\#XOR$	$\#AND$	$\#Evaluation$
I	$F(x) = M_1 \cdot G(x) + V_1$	$2^n(\frac{m^2(2k+1)+m}{2}) + \frac{m^2(2k-1)-m}{2}$	$2km^2$	$2^{n+1} + 2$
III	$F(x) = G(x) + M_3 \cdot x + V_1$	$mn + m$	mn	$n + 1$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$2^n(\frac{m^2(2k+1)+m}{2}) + \frac{k(2m^2n+2mn+2m^2)-m^2-m}{2}$	$k(nm^2 + 2m^2 + mn)$	$2^{n+1} + 2n + 2$

		Complexity in case $\{2^i \mid 0 \leq i \leq m-1\} \subseteq img(G')$		
Type	REA-equivalence	$\#XOR$	$\#AND$	$\#Evaluation$
I	$F(x) = M_1 \cdot G(x) + V_1$	$2km^2$	$2km^2$	$2^{n+1} + 2$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$k(nm^2 + 2m^2 + mn)$	$k(nm^2 + 2m^2 + mn)$	$2^{n+1} + 2n + 2$

		Complexity in case G Permutation		
Type	REA-equivalence	$\#XOR$	$\#AND$	$\#Evaluation$
I	$F(x) = M_1 \cdot G(x) + V_1$	$2n^2$	$2n^2$	$2^n + n + 2$
II	$F(x) = G(M_2 \cdot x + V_2)$	n^2	n^2	$n + 3$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$n^3 + 3n^2$	$n^3 + 3n^2$	$2^n + 3n + 2$

Table 5.18: The complexities of two new types of REA-equivalence

		Complexity in case $\{F'(2^i) \mid 0 \leq i \leq n-1\} \subseteq img(G')$		
Type	REA-equivalence	$\#XOR$	$\#AND$	$\#Evaluation$
V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$	$k(n^2 + 2mn + m - n) + m$	$k(n^2 + mn)$	$2^n + 2n + 2 + k(n+1)$
VI	$F(x) = G(M_2 \cdot x) + V_1$	$k(n^2 - n + m) + m$	kn^2	$2^n + n + 2 + k$

		Complexity in case G Permutation		
Type	REA-equivalence	$\#XOR$	$\#AND$	$\#Evaluation$
V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$	$3n^2 + n$	$2n^2$	$2^n + 3n + 3$
VI	$F(x) = G(M_2 \cdot x) + V_1$	$n^2 + n$	n^2	$2^n + n + 3$

We study the complexities of verification of some types of REA-equivalence. To do that, we count the explicit number of field operations, $\oplus$ and $\odot$, and polynomial evaluation. Polynomial evaluation of vectorial boolean functions can be computed by table look-up operation, where the tables consist of values of the functions. Hence, the polynomial evaluations of vectorial boolean functions are table look-up operations and, each table look-up operation is considered as 1 field operation. Similarly, each *XOR* and *AND* operations are considered as 1 field operation. These are meaningful as we are in $\mathbb{F}_2$. Thus, we add the number of *XOR*, *AND* and table look-up operations. This enables us to see the total number of operations of checking process for REA-equivalence. We present the total complexity of REA-equivalence types in the Table 5.19 and 5.20.

Table 5.19: Our total complexities of REA-equivalence types in [7]

Type	REA-equivalence	Complexity in case G Arbitrary
I	$F(x) = M_1 \cdot G(x) + V_1$	$2^n(\frac{m^2(2k+1)+m+4}{2}) + \frac{m^2(6k-1)-m+4}{2}$
III	$F(x) = G(x) + M_3 \cdot x + V_1$	$2mn + m + n + 1$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$2^n(\frac{m^2(2k+1)+m+4}{2}) + \frac{k(4m^2n+4mn+6m^2)+4n-m^2-m+4}{2}$

		$\{2^i \mid 0 \le i \le m-1\} \subseteq img(G')$
Type	REA-equivalence	Complexity
I	$F(x) = M_1 \cdot G(x) + V_1$	$2^{n+1} + 4km^2 + 2$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$2^{n+1} + 2k(nm^2 + 2m^2 + mn) + 2n + 2$

Type	REA-equivalence	Complexity in case G Permutation
I	$F(x) = M_1 \cdot G(x) + V_1$	$2^n + 4n^2 + n + 2$
II	$F(x) = G(M_2 \cdot x + V_2)$	$2n^2 + n + 3$
IV	$F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$	$2^n + 2n^3 + 6n^2 + 3n + 2$

Table 5.20: Our total complexities of two new types of REA-equivalence

		$\{F'(2^i) \mid 0 \le i \le n-1\} \subseteq img(G')$
Type	REA-equivalence	complexity
V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$	$2^n + k(2n^2 + 3mn + m + 1) + m + 2n + 2$
VI	$F(x) = G(M_2 \cdot x) + V_1$	$2^n + k(2n^2 + m - n + 1) + m + n + 2$

Type	REA-equivalence	complexity in case G Permutation
V	$F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$	$2^n + 5n^2 + 4n + 3$
VI	$F(x) = G(M_2 \cdot x) + V_1$	$2^n + 2n^2 + 2n + 3$

We note that if we add one of the matrices V_1, V_2 to REA-equivalence types, the complexity of corresponding type then will increase in 2^m, 2^n times, respectively. Because all operations in checking procedure for REA-equivalence will be repeated for each value $V_1 \in \mathbb{F}_2^m$ or $V_2 \in \mathbb{F}_2^n$.

CHAPTER 6

CONCLUSION

This thesis studies restricted cases of *Extended Affine (EA)-equivalence*, called *Restricted Extended Affine (REA)-equivalence*. Firstly, we analyze in Section 5.1 the verification procedures of REA-equivalence types given in [7]. We also show the verification procedures with some examples. We expose the complexities of checking procedures for these types. We count explicitly the number of operations which are *XOR*, *AND* and polynomial evaluation in the verification procedures. We have shown our complexities particularly in the Table 5.17 for these types. Secondly, we introduce two new types of REA-equivalence in Section 5.2. We construct the verification procedures of these types, namely types V and VI, with their complexities. Finally, we have presented the pseudocodes of the verification procedures for REA -equivalence types studied in this thesis. We also implement these algorithms for REA-equivalence in MAGMA [8].

As a result, this thesis provides the fast technique for determining whether two vectorial Boolean functions with more than six variables are equivalent or not. When one constructs a new *s-box*, one can also check whether it is equivalent to already known ones in block cipher by using our MAGMA codes presented in the Appendix. Since *CCZ-equivalence* and *EA-equivalence* have extremely high complexity, this research can be useful for verification of *s-boxes* in block cipher. This study will contribute to the cryptography in terms of the verification of vectorial Boolean functions.

REFERENCES

[1] E. Biham, A. Shamir, *Differential cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[2] A. Biryukov, C. De Canniere, A. Braeken, and B. Preneel, *A tool-box for cryptanalysis: Linear and affine equivalence algorithms*, In Advances in Cryptology-EUROCRYPT 2003, Lecture Notes in Computer Science, Eli Biham, editor, Springer- Verlag, 33-50, 2003.

[3] A. Bogdanov, M. Mertens, C. Paar, J. Pelzl, and A. Rupp, *A parallel hardware architecture for fast gaussian elimination over $GF(2)$*, In: FCCM 2006: Proceedings of the 14-th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Washington,DC, USA, IEEE Computer Society, Los Alamitos, 237-248, 2006.

[4] L. Budaghyan, C. Carlet and A. Pott, *New classes of almost bent and almost perfect nonlinear polynomials*, IEEE Trans. Inform. Theory, 52:1141-1152, 2006.

[5] L. Budaghyan, C. Carlet, and G. Leander, *Constructing new APN functions from known ones*, Finite Fields Appl., 15:150-159, 2009.

[6] L. Budaghyan and C. Carlet, *CCZ-equivalence and Boolean functions*, Preprint available at IACR ePrint Archive, 2009.

[7] L. Budaghyan and O. Kazymyrov, *Verification of Restricted EA-Equivalence for Vectorial Boolean Functions*, WAIFI 2012, Lecture Notes in Comput. Sci., 7369 Eds. F. Özbudak and F. Rodriguez-Henriquez, Springer-Verlag, Berlin, 108-118, 2012.

[8] W. Bosma, J. Cannon, and C. Playoust, *The Magma algebra system*, I. The user language. J. Symbolic Comput. 24:235-265, 1997.

[9] C. Carlet, P. Charpin, and V. Zinoviev, *Codes, bent functions and permutations suitable for DES-like cryptosystems*, Designs, Codes and Cryptography, 15(2):125-156, 1998.

[10] C. Carlet, *Vectorial Boolean functions for Cryptography*, to appear as a chapter of the monograph Boolean methods and models, Cambridge University Press (Ed. Peter Hammer and Yves Crama), available a http://www-rocq.inria.fr/secret/Claude.Carlet/chap-vectorial-fcts.pdf

[11] F. Chabaud and S. Vaudenay, *Links between differential and linear cryptanalysis*, Advances in Cryptology-EUROCRYPT'94, LNCS, Springer-Verlag, New York,950, 356-365, 1995.

[12] J.E. Fuller, *Analysis of Affine Equivalent Boolean Functions for Cryptography*, Phd Thesis, Queensland University of Technology, Australia, 2003.

[13] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd edition, Cambridge University Press, Cambridge, UK, 2003.

[14] K.J. Horadam, *Hadamard Matrices and Their Applications*, Princeton University Press, Princeton, 2007.

[15] K.J. Horadam, *EA and CCZ Equivalence of Functions over* $GF(2^n)$, WAIFI 2008, LNCS 5130, Springer-Verlag, Berlin Heidelberg, 134-143, 2008.

[16] K.J. Horadam and R. East, *Partitioning CCZ classes into EA classes*, ADVANCE IN MATHEMATICS OF COMMUNICATIONS, 6(1):95-106, 2012.

[17] D. Kwon et al., *New Block Cipher: ARIA*, In Jong In Lim and Dong Hoon Lee, editors, ICISC, volume 2971 of Lecture Notes in Computer Science, Springer-Verlag, 432-445, 2003.

[18] K. Nyberg, *Differentially uniform mappings for cryptography*, Advances in Cryptography, EUROCRYPT'93, LNCS, Springer-Verlag, New York, 765:55-64, 1994.

[19] M. Matsui, *Linear Cryptanalysis Method for DES Cipher*, Proceedings of Eurocrypt 93, Springer-Verlag, LNCS 765:386-397, 1993.

[20] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, 2005. Cambridge Books Online. http://dx.doi.org/10.1017/CBO9781139165464

[21] V.V. Williams, *Breaking the Coppersmith-Winograd barrier*, November 2011, Available at http://www.cs.berkeley.edu/ virgi/matrixmult.pdf

APPENDIX A

MAGMA CODES

We begin with a common function, named as SetColumn, of each types.

```
SetColumn:=function(M,i,V)
      M1:=M;
   for j in [1..Nrows(M)] do
      M1[j,i]:=V[j];
   end for;
   return M1;
end function;
```

A.1 REA-equivalence of Type I

We present the source code of F and G for REA-equivalence of Type I in the form $F(x) = M_1 \cdot G(x) + V_1$ in MAGMA.

```
REA_TypeI:=function(F,G,P,n,m,a)
  P<x>:=PolynomialRing(GF(2^n));
  a:=PrimitiveElement(GF(2^n));
  Fm<t>:=FiniteField(2^m);
  Fn<a>:=FiniteField(2^n);
  F_0:=Evaluate(F,Fn!0);
  F_1:=F-F_0;
  G_0:=Evaluate(G,Fn!0);
  G_1:=G-G_0;
```

```
    inverse_2_is:=[];
  for i:=0 to m-1 do
    zero_vector_m:=Eltseq(Fm!0);
    zero_vector_m[i+1]:=GF(2)!1;
    two_i_m:=zero_vector_m;
      ysayac:=1;
      inverse_2_is[i+1]:=[];
    for y in Fn do
      if Eltseq(Fn!Evaluate(G_1,y)) eq two_i_m then
        inverse_2_is[i+1][ysayac] := y;
        ysayac +:= 1;
      end if;
    end for;
  end for;
      car_inverses:=car<inverse_2_is[1],inverse_2_is[2]>;
    for i in [3..m] do
      car_inverses:=car<car_inverses,inverse_2_is[i]>;
    end for;
      car_inverses:=Flat(car_inverses);
      printf"car_inverses: %o\n",car_inverses;
for z in car_inverses do
    M_1:=Matrix(GF(2),m,m,[]);
  for i in [1..m] do
    M_1:=SetColumn(M_1,i,Eltseq(Evaluate(F_1,Fn!z[i]),GF(2)));
  end for;
    V_1:=Matrix(GF(2),m,1,[]);
    C:=M_1*Matrix(m,1,Eltseq(G_0,GF(2)));
    D:=Matrix(m,1,Eltseq(F_0,GF(2)));
    V_1:=C+D;
    check_y:=0;
  for y in Fn do
    E:=Matrix(m,1,Eltseq(Evaluate(F,y),GF(2)));
    L:=M_1*Matrix(m,1,Eltseq(Evaluate(G,y),GF(2)));
```

```
        if E ne L+V_1 then
          break y;
        end if;
          check_y+:=1;
      end for;
        printf"check_y: %o\n",check_y;
      if check_y eq 2^n then
        print M_1,V_1;
        return true;
      end if;
  end for;
  return false;
end function;
n:=9;
m:=9;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
F:=x+x^4+x^8+a*x^8+a^2*x^4+a^3*x^8+a^3*x+a^3*x^2+a^3*x^4;
G:=x+a*x+a*x^4+a*x^8+a^2*x^4+a^2*x^8+a^3*x^2+a^3*x^4;
REA_TypeI(F,G,P,n,m,a);
```

A.2 REA-equivalence of Type II

We present the source code of F and G for REA-equivalence of Type II in the form $F(x) = G(M_2 \cdot x + V_2)$ in MAGMA.

```
REA_TypeII:=function(F,G,P,n,a)
  P<x>:=PolynomialRing(GF(2^n));
  a:=PrimitiveElement(GF(2^n));
  Fn<a>:=FiniteField(2^n);
  y:=[];
  z:=[];
  for i:=1 to 2^n-1 do
  y[i]:=a^i;
  z[i]:=Evaluate(G,y[i]);
end for;
  y[2^n]:=GF(2^n)!0;
  z[2^n]:=Evaluate(G,y[2^n]);
  G_inverse:=Interpolation(z,y);
  H:=Evaluate(G_inverse,F);
  H_0:=Evaluate(H,Fn!0);
  H_1:=H-H_0;
  H:=H_1+H_0;
  M_2:=Matrix(GF(2),n,n,[]);
for i:=0 to n-1 do
  zero_vector_n:=Eltseq(Fn!0);
  zero_vector_n[i+1] := GF(2)!1;
  two_i_n:=zero_vector_n;
  M_2:=SetColumn(M_2,i+1,Eltseq(Evaluate(H_1,Fn!two_i_n),GF(2)));
end for;
  V_2:=Matrix(GF(2),n,1,[]);
  V_2:=Matrix(n,1,Eltseq(H_0,GF(2)));
  check_y:=0;
for y in Fn do
  E:=Matrix(n,1,Eltseq(Evaluate(H,y),GF(2)));
  T:=M_2*Matrix(n,1,Eltseq(y,GF(2)));
  if E ne T+V_2 then
    break y;
  end if;
  check_y+:=1;
end for;
if check_y eq 2^n then
  print M_2,V_2;
  return true;
end if;
return false;
```

```
end function;
n:=8;
m:=8;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
F:=x^2+a^5+1;
G:=x^2+a;
REA_TypeII(F,G,P,n,a);
```

A.3 REA-equivalence of Type III

We present the source code of F and G for REA-equivalence of Type III in the form $F(x) = G(x) + M_3 \cdot x + V_1$ in MAGMA.

```
REA_TypeIII:=function(F,G,P,n,m,a)
  P<x>:=PolynomialRing(GF(2^n));
  a:=PrimitiveElement(GF(2^n));
  Fm<t>:=FiniteField(2^m);
  Fn<a>:=FiniteField(2^n);
  K:=F+G;
  K_0:=Evaluate(K,Fn!0);
  K_1:=K-K_0;
  K:=K_1+K_0;
    M_3:=Matrix(GF(2),m,n,[]);
  for i:=0 to n-1 do
    zero_vector_n:=Eltseq(Fn!0);
    zero_vector_n[i+1] := GF(2)!1;
    two_i_n:=zero_vector_n;
    M_3:=SetColumn(M_3,i+1,Eltseq(Evaluate(K_1,Fn!two_i_n),GF(2)));
  end for;
    V_1:=Matrix(GF(2),m,1,[]);
    V_1:=Matrix(m,1,Eltseq(K_0,GF(2)));
    check_y:=0;
  for y in Fn do
    E:=Matrix(m,1,Eltseq(Evaluate(F,y),GF(2)));
    L:=Matrix(m,1,Eltseq(Evaluate(G,y),GF(2)));
    T:=M_3*Matrix(m,1,Eltseq(y,GF(2)));
    if E ne L+T+V_1 then
      break y;
    end if;
      check_y+:=1;
  end for;
    printf"check_y: %o\n",check_y;
  if check_y eq 2^n then
    print M_3,V_1;
    return true;
  end if;
  return false;
end function;
n:=10;
m:=10;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
F:=x^3;
G:=x^9;
REA_TypeIII(F,G,P,n,m,a);
```

A.4 REA-equivalence of Type IV

We present the source code of F and G for REA-equivalence of Type IV in the form $F(x) = M_1 \cdot G(x) + M_3 \cdot x + V_1$ in MAGMA.

```
REA_TypeIV:=function(F,G,P,n,m,a)
  P<x>:=PolynomialRing(GF(2^n));
  a:=PrimitiveElement(GF(2^n));
```

```
Fm<t>:=FiniteField(2^m);
Fn<a>:=FiniteField(2^n);
F_0:=Evaluate(F,Fn!0);
L_F:=F mod x^2;
L_F:=L_F-F_0;
F_1:=F-L_F-F_0;
G_0:=Evaluate(G,Fn!0);
L_G:=G mod x^2;
L_G:=L_G-G_0;
G_1:=G-L_G-G_0;
    inverse_2_is:=[];
  for i:=0 to m-1 do
    zero_vector_m:=Eltseq(Fm!0);
    zero_vector_m[i+1] := GF(2)!1;
    two_i_m:=zero_vector_m;
    ysayac := 1;
    inverse_2_is[i+1] := [];
    for y in Fn do
      if Eltseq(Fn!Evaluate(G_1,y)) eq two_i_m then
        inverse_2_is[i+1][ysayac] := y;
        ysayac +:= 1;
      end if;
    end for;
  end for;
      car_inverses:=car<inverse_2_is[1],inverse_2_is[2]>;
    for i in [3..m] do
      car_inverses:=car<car_inverses,inverse_2_is[i]>;
    end for;
      car_inverses:=Flat(car_inverses);
      printf"car_inverses: %o\n",car_inverses;
for z in car_inverses do
    M_1:=Matrix(GF(2),m,m,[]);
  for i in [1..m] do
```

```
    M_1:=SetColumn(M_1,i,Eltseq(Evaluate(F_1,Fn!z[i]),GF(2)));
  end for;
    V_1:=Matrix(GF(2),m,1,[]);
    C:=M_1*Matrix(m,1,Eltseq(G_0,GF(2)));
    D:=Matrix(m,1,Eltseq(F_0,GF(2)));
    V_1:=C+D;
    M_3:=Matrix(GF(2),m,n,[]);
  for i:=0 to n-1 do
    zero_vector_n:=Eltseq(Fn!0);
    zero_vector_n:=Eltseq(Fn!0);
    zero_vector_n[i+1] := GF(2)!1;
    two_i_n:=zero_vector_n;
    B:=Matrix(m,1,Eltseq(Evaluate(L_F,Fn!two_i_n),GF(2)));
    A:=M_1*Matrix(m,1,Eltseq(Evaluate(L_G,Fn!two_i_n),GF(2)));
    M_3:=SetColumn(M_3,i+1,Eltseq(B+A));
  end for;
    check_y:=0;
  for y in Fn do
    E:=Matrix(m,1,Eltseq(Evaluate(F,y),GF(2)));
    L:=M_1*Matrix(m,1,Eltseq(Evaluate(G,y),GF(2)));
    T:=M_3*Matrix(m,1,Eltseq(y,GF(2)));
    if E ne L+T+V_1 then
      break y;
    end if;
      check_y+:=1;
  end for;
    printf"check_y: %o\n",check_y;
  if check_y eq 2^n then
    print M_1,M_3,V_1;
    return true;
  end if;
end for;
return false;
```

```
end function;
n:=3;
m:=3;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
G:=x^6+a*x^5+a^5*x^4+a*x^4+x^4+a^4*x^3+a^6*x^2+a^2*x+a^3;
F:=x^6+1;
REA_TypeIV(F,G,P,n,m,a);
```

A.5 REA-equivalence of Type V

We present the source code of F and G for REA-equivalence of Type V in the form $F(x) = G(M_2 \cdot x) + M_3 \cdot x + V_1$ in MAGMA.

```
REA_TypeV:=function(F,G,P,n,m,a)
  P<x>:=PolynomialRing(GF(2^n));
  a:=PrimitiveElement(GF(2^n));
  Fm<t>:=FiniteField(2^m);
  Fn<a>:=FiniteField(2^n);
  F_0:=Evaluate(F,Fn!0);
  L_F:=F mod x^2;
  L_F:=L_F-F_0;
  F_1:=F-L_F-F_0;
  G_0:=Evaluate(G,Fn!0);
  L_G:=G mod x^2;
  L_G:=L_G-G_0;
  G_1:=G-L_G-G_0;
      inverse_f_2_is:=[];
    for i:=0 to n-1 do
      zero_vector_n:=Eltseq(Fn!0);
      zero_vector_n[i+1] := GF(2)!1;
      two_i_n:=zero_vector_n;
        ysayac := 1;
        inverse_f_2_is[i+1]:=[];
      for  y in Fn do
        if Evaluate(F_1,Fn!two_i_n) eq Evaluate(G_1,y) then
          inverse_f_2_is[i+1][ysayac] := y;
          ysayac +:= 1;
        end if;
      end for;
    end for;
        car_inverses:=car<inverse_f_2_is[1],inverse_f_2_is[2]>;
      for i in [3..n] do
        car_inverses:=car<car_inverses,inverse_f_2_is[i]>;
      end for;
        car_inverses:=Flat(car_inverses);
        printf"car_inverses: %o\n",car_inverses;
  for z in car_inverses do
      printf"z: %o\n",z;
      M_2:=Matrix(GF(2),n,n,[]);
    for i in [1..n] do
      M_2:=SetColumn(M_2,i,Eltseq(z[i],GF(2)));
    end for;
      V_1:=Matrix(GF(2),m,1,[]);
      C:=Matrix(m,1,Eltseq(G_0,GF(2)));
      D:=Matrix(m,1,Eltseq(F_0,GF(2)));
      V_1:=C+D;
      M_3:=Matrix(GF(2),m,n,[]);
    for i:=0 to n-1 do
      zero_vector_n:=Eltseq(Fn!0);
      zero_vector_n[i+1] := GF(2)!1;
      two_i_n:=zero_vector_n;
      two_i_n_v:=M_2*Matrix(n,1,two_i_n);
      M_3:=SetColumn(M_3,i+1,Eltseq(Evaluate(L_F,Fn!two_i_n)
           +Evaluate(L_G,Fn!Eltseq(two_i_n_v)),GF(2)));
    end for;
```

```
      check_y:=0;
    for y in Fn do
      E:=Matrix(m,1,Eltseq(Evaluate(F,y),GF(2)));
      matrix_y:=M_2*Matrix(GF(2),n,1,Eltseq(y));
      L:=Matrix(m,1,Eltseq(Evaluate(G,Fn!Eltseq(matrix_y)),GF(2)));
      T:=M_3*Matrix(n,1,Eltseq(y,GF(2)));
      if E ne L+T+V_1 then
        break y;
      end if;
        check_y+:=1;
    end for;
      printf"check_y: %o\n",check_y;
    if check_y eq 2^n then
      print M_2,M_3,V_1;
      return true;
    end if;
  end for;
  return false;
end function;
n:=9;
m:=9;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
F:=x^3+a^12*x^2+a^13*x+a;
G:=x^3+a^12*x^2+a^13*x+a^12;
REA_TypeV(F,G,P,n,m,a);
```

A.6 REA-equivalence of Type VI

We present the source code of F and G for REA-equivalence of Type VI in the form $F(x) = G(M_2 \cdot x) + V_1$ in MAGMA.

```
REA_TypeVI:=function(F,G,P,n,m,a)
```

```
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
Fm<t>:=FiniteField(2^m);
Fn<a>:=FiniteField(2^n);
F_0:=Evaluate(F,Fn!0);
F_1:=F-F_0;
G_0:=Evaluate(G,Fn!0);
G_1:=G-G_0;
    inverse_f_2_is:=[];
  for i:=0 to n-1 do
    zero_vector_n:=Eltseq(Fn!0);
    zero_vector_n[i+1] := GF(2)!1;
    two_i_n:=zero_vector_n;
      ysayac := 1;
      inverse_f_2_is[i+1] := [];
    for  y in Fn do
      if Evaluate(F_1,Fn!two_i_n) eq Evaluate(G_1,y) then
        inverse_f_2_is[i+1][ysayac] := y;
        ysayac +:= 1;
      end if;
    end for;
  end for;
    car_inverses:=car<inverse_f_2_is[1],inverse_f_2_is[2]>;
  for i in [3..n] do
    car_inverses:=car<car_inverses,inverse_f_2_is[i]>;
  end for;
    car_inverses:=Flat(car_inverses);
    printf"car_inverses: %o\n",car_inverses;
for z in car_inverses do
    printf"z: %o\n",z;
    M_2:=Matrix(GF(2),n,n,[]);
  for i in [1..n] do
    M_2:=SetColumn(M_2,i,Eltseq(z[i],GF(2)));
    end for;
      V_1:=Matrix(GF(2),m,1,[]);
      C:=Matrix(m,1,Eltseq(G_0,GF(2)));
      D:=Matrix(m,1,Eltseq(F_0,GF(2)));
      V_1:=C+D;
      check_y:=0;
    for y in Fn do
      E:=Matrix(m,1,Eltseq(Evaluate(F,y),GF(2)));
      matrix_y:=M_2*Matrix(GF(2),n,1,Eltseq(y));
      L:=Matrix(m,1,Eltseq(Evaluate(G,Fn!Eltseq(matrix_y)),GF(2)));
      if E ne L+V_1 then
        break y;
      end if;
      check_y+:=1;
    end for;
      printf"check_y: %o\n",check_y;
    if check_y eq 2^n then
      print M_2,V_1;
      return true;
    end if;
  end for;
  return false;
end function;
n:=4;
m:=4;
P<x>:=PolynomialRing(GF(2^n));
a:=PrimitiveElement(GF(2^n));
F:=x^3;
G:=x^9;
REA_TypeVI(F,G,P,n,m,a);
```