THE USE OF THE BIG ENCRYPTION SYSTEM (BES) FOR THE CRYPTANALYSIS OF THE ADVANCED ENCRYPTION SYSTEM (AES)

YING YU

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

APRIL 2005

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-04404-7
Our file Notre référence
ISBN: 0-494-04404-7

Abstract

The use of the Big Encryption System (BES) for the Cryptanalysis of the

Advanced Encryption System (AES)

Ying Yu

Because the operations of AES are based on two different Galois Fields, GF(2) and GF(2^8), this brings a lot of difficulties and complexities to the cryptanalysis of AES. Therefore, BES, which stands for Big Encryption System, is introduced. By enlarging the message space and key space, BES can replicate the actions of AES by just using simple operations only on GF (2^8), thereby enabling an easy and equivalent approach to analyze AES. In order to give more practical support to the cryptanalysis of the AES, the BES implementation is strongly encouraged.

This thesis discusses the issues related to the BES implementation. Firstly, a detailed exploration of the implementation of BES encryption is addressed. Then the inverse BES cipher is proposed with full respect to the design principles of BES. Finally, a testing analysis of the BES implementation is given.

Acknowledgments

I would begin by expressing my gratitude to my supervisor, Hovhannes A. Harutyunyan, for the support and help in writing my thesis. His charisma of being a intelligent, dedicated and passionate researcher has inspired me throughout the course of my learning, and will continue to do so.

Special thanks to my husband, Yang Yu, for his love and patience, for being incredibly understanding and supportive, and for being such a perfect husband. Thanks to my family and friends for their sincere encouragement and constant support.

Finally, merci beaucoup, Montreal!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptography has a long and fascinating history, which can be traced back to its initial and limited use by the Egyptians some 4000 years ago. With the proliferation of computer technology, it becomes more crucially used as a tool to manage the risk in a very wide range of situations. This chapter introduces, in general terms, the relevant background and basics of cryptography, and then gives an overall structure of how this thesis is organized.

1.1 The Overview of Cryptography

Cryptography is the study of secret writing, which involves mathematical techniques related to aspects of information security such as confidentiality, entity authentication and data origin authentication.

Fundamentally, encryption is the process of the rendering of information into a different unintelligible form so that its contents are not readily accessible. The original information is known as plaintext, and the encrypted information as ciphertext. The reverse process of extracting the original message from ciphertext is called decryption. Both encryption and decryption depend on particular algorithms, known as ciphers. The shared secret knowledge used in conjunction with a cipher is generally known as the key, which determines the mapping of the plaintext to the ciphertext. Cryptanalysis is the study of attacks on ciphers.

The ciphers used to encrypt and decrypt data fall into two categories: public key (asymmetric) and secret key (symmetric). The public key system, also known as asymmetric cryptography, involves two keys that are mathematically linked such that one key (the public key) is used for encrypting plain text, which can then only be decrypted by the other corresponding key (the private key). The secret key system, also known as symmetric cryptography (which is oldest kind of encryption method), uses the same key both to encrypt the plain text, and subsequently to decrypt the cipher text. The symmetric systems are considerably faster than the asymmetric methods, but are vulnerable because of the need to share the key, and thereby risk it falling into the wrong hands. However, with the developments in mathematics and the growth of computing power, now it can be effectively made unbreakable. Symmetric ciphers can be implemented using block ciphers or stream ciphers. Block ciphers encrypt a fixed-length block of the plain text, under the control of the secret key, into the cipher text of the same length. The ciphers related to this thesis are all block ciphers, including DES (the Data encryption Standard), AES (the Advanced Encryption Standard) and BES (the Big Encryption System). Moreover, all of the above are iterated ciphers, composed of several rounds, which helps improve security.

DES, deriving from the work of IBM, was adopted a U.S. Federal Information Processing Standard for encrypting unclassified information in 1977. It is designed to encipher and decipher blocks of data consisting of 64-bit block under control of a 64-bit key (although the effective key strength is only 56 bits). DES has 16 rounds, which entails that the plaintext is processed through 16 times to produce the ciphertext. In a single round, we split the 64 bits into two 32-bit blocks, and take the 32-bit block on the right-hand-side to make it the left 32-bit block for the next round, and XOR the left 32-bit block with the calculation of a Feistel function that operates on the right 32-bit block and a 48-bit key block to produce the right 32-bit block for the next round. This function will further involve a block expansion from 32 bits to 48 bits, a key XOR with 48 selected bits out of the available 56 key bits, a substitution using eight 6-to-4 S-Boxes and a permutation to reorder the bits. For a detailed specification, see [1].

However, DES was showing signs of aging in the 1990s. The resistance to a *forced* or *brute* attack of an encoding system is directly related to its key space: the more bits that are used will translate into more possible keys, and having more keys means that it takes longer to compute the full range of possible keys of the entire key space in a forced attack. DES cuts eight bits off the top of its key, which limits the key space to a great degree, thereby making the system easier to crack. Moreover, with the increase of well-designed *DES Crackers*, DES is no longer secure. This has been especially proved when the DES cracker, developed by Electronic Frontier Foundation (EFF) for under 250 000 dollars, completed the *DES Challenge II* in less than 3 days in 1998 [2]. Half a year later, in Jan 1999, they won the *DES Challenge III* together with Distributed.Net in 22 hours and 15 minutes by inexpensively applying off-the-shelf technology and minimal engineering. Due to the insufficient security of DES mentioned above, a replacement was overdue.

In January 1997, the US National Institute of Standards and Technology (NIST) announced the start of an open competition to develop a new encryption standard: the Advanced Encryption Standard (AES), replacing the old DES and triple-DES. The minimum functional requirements for candidate nominations asked for symmetric block ciphers capable of supporting 128-bit data blocks with 128-bit, 192-bit and 256-bit key blocks. Instead of performing any security or efficiency evaluation on 15 AES candidate algorithms by NIST itself, the whole cryptology community was invited to review and evaluate them. In August 1999, NIST narrowed down the candidates to five finalists: MARS (IBM), RC6 (RSA Laboratories), Rijndael (Daemen and Rijmen), Serpent (Anderson, Biham and Knudsen) and Twofish (Counterpane Sys and Univ. Berkeley). The evaluation criteria were based on security, costs, versatility, key agility and simplicity.

Furthermore, these five algorithms were tested and evaluated on features, performance in Hardware and Software, IP issues, cross-cutting analysis, attacks against the algorithms, etc. When considered together, Rijndael's combination of security, performance, efficiency, ease of implementation and flexibility made it an appropriate selection for the AES. On October 2, 2000, NIST announced Rijndael as the winner of the AES competition. Finally, after another year of evaluation, the US Department of Commerce officially declared Rijndael to be AES. The National Security Agency also endorsed Rijndael and the public evaluation effort led to widespread acceptance. More importantly, Rijndael, or any of its implementations, are not subject to any patents. For a detailed AES specification, see [3].

As Rijndael was crowned as the Advanced Encryption Standard, a considerable amount of cryptanalytic attention has been paid to it. However, the operations of the cipher over two different finite fields, GF(2) and GF(2^8), bring many difficulties and complexities to the AES cryptanalysis. Thus, an extension of AES, BES cipher that involves only simple operations over GF(2^8), has been introduced in order to avoid the conflict.

BES, defined by S. Murphy and M.J.B. Robshaw [7], is a 128-byte iterated block cipher, which is composed of only simple algebraic operations over GF(2^8) with fully respecting encryption in the AES. Since AES can be thought of as being identical to the BES, only with a restricted message and key space, the properties of the new cipher have a close affinity with the properties of the AES. This enables BES to preserve algebraic curves of AES within an enlarged message and key space. Moreover, by using no more operations over GF$(2)^8$, the new cipher is easier to analyze, which possibly offers a significant improvement to the cryptanalysis of the AES. In particular, one consequence of analyzing the BES is that the security of the AES is equivalent to the solubility of certain extremely sparse multivariate quadratic systems over GF(2^8) [7]. However, the BES is only intended to be used as an analysis tool, and the implementation of BES in my thesis is intended to facilitate the practical use of BES.

1.2 Thesis Organization

The thesis primarily endeavors to contribute to the issues related to implementation of the Big Encryption System *(BES).*

- Chapter 2 deals with the basic structure of AES based on the second version of the original Rijndael documentation [4] .

- Chapter 3 starts with introducing the common mathematical framework required to establish the relationship of AES and BES, and then gives a comprehensive description of the BES structure.

- Chapter 4 gives a detailed exploration of the encryption process of BES.

- Chapter 5 deals with the implementation of the BES decryption with respect to the original design idea of BES cipher.

- Chapter 6 attempts to analyze the results from the BES implementation.

- The final chapter lists the conclusions derived and suggests a direction for future research.

- A thorough list of bibliographic references is presented next.

- The source code is added in the appendix to assist the analysis in Chapter 6.

Chapter 2

The Specifications of AES

Rijndael, deriving from SQUARE, is an iterated symmetric block cipher with a variable block length and a variable key length. The block length and the key length in Rijndael can be independently chosen as 128, 192 or 256 bits, while the block size in the Advanced Encryption Standard (AES) is a mandatory 128-bit block length irrespective of the 128-bit, 192-bit or 256-bit key lengths.

The following sections explain the AES cipher structure based on the second version of the original Rijndael documentation [4] .

2.1 The Cipher Structure

Unlike DES based on the Feistel structure (typically part of the bits of the intermediate state are simply transposed in an unchanged format to another position), the round transformation of Rijndael is based on SP-Network model (typically each block of data is divided into smaller manageable pieces of the same length, and in parallel every piece goes through the confusion layer (S-Box) and the diffusion layer (P-Box)). Rijndael consists of three distinct invertible uniform transformations (called layers), where every bit of the state is treated in a similar way. These two typical structures are shown in Figure 1.

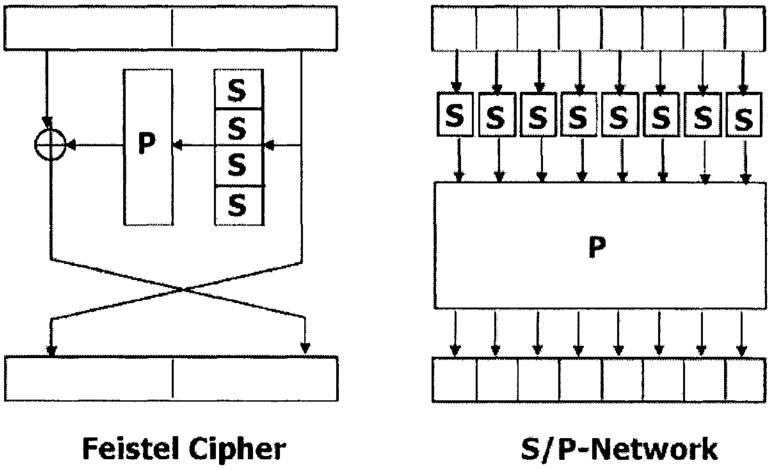


Figure 1: Feistel Structure (DES) and SP-Network Model (AES)

2.2 State and the Cipher Key Representation

The cipher block to which the operations are applied is termed as the *State*. A state is pictured as a rectangular array of 8-bit bytes with four rows. The number of columns in the state denoted by N_b is equal to the block length divided by 32. The cipher key is organized in the same form as the state. Thus, the number of columns of the cipher key denoted by N_k is equal to the key length divided by 32 as well. The following figure is the example layout of State (with N_b = 4) and Cipher Key (with N_k = 4) layout.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figure 2: The State and the Cipher Key (N_k = 4) Layout

Both the state and the cipher key also can be considered to be one-dimensional array of 4-byte vectors or words. Therefore, the bytes of the state are mapped onto the array in the order of $a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, a_{2,1}, a_{3,1}, a_{0,2}...$, while the bytes of the key are mapped onto the array in the order of $k_{0,0}, k_{1,0}, k_{2,0}, k_{3,0}, k_{0,1}, k_{1,1}, k_{2,1}, k_{3,1}, k_{0,2}...$

After the FinalRound, the ciphertext will be extracted from the state bytes in the similar order. Accordingly, the same principle applies to the decryption process.

2.3 Number of Rounds

Rijndael consists of a number of equivalent rounds. The number of rounds denoted by N_r is a function of N_b and N_k. For the AES, N_b is fixed at the value 4: $N_r = 10$ for a 128-bit key ($N_k = 4$), $N_r = 12$ and $N_r = 14$ for key sizes of 192-bits ($N_k = 6$) and 256-bits ($N_r = 8$) respectively.

2.4 Round Transformation

The round transfromation denoted by *Round* is composed of four different transformations: ByteSub[1], ShiftRow[2], MixColumn[3] and AddRoundKey. The final round of the cipher denoted by *FinalRound* is slightly different: the step of MixColumn is not executed. A pseudo-C notation of Round transformations is given below:

```
Round(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State,RoundKey);
}
FinalRound(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State,RoundKey);
}
```

[1] SubBytes in the AES specification

[2] ShiftRows in the AES specification

[3] MixColumns in the AES specification

2.4.1 ByteSub Transformation

The *ByteSub* step is the only non-linear transformation of the cipher. ByteSub consists of an S-Box (Substitution Box) denoted by S_{RD} applied to the bytes of the state. The S-Box is a 256-byte table shown in figure 3 , where the value of the upper nibble m and the lower nibble n of an input byte is used to determine the value of the output byte.

		n															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
m	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 1: Tabular Representation of $S_{RD}(mn)$

Basically the S-Box is defined as a composition of two transformations:

1. Taking the multiplicative inverse in GF(2^8).

   Rijndael uses the polynomial representation of GF(2^8)[6] .The elements of GF(2^8) are considered as polynomials whose degree are smaller than 8 over the finite field GF(2). Multiplication of two polynomials is defined as the algebraic product of the polynomials modulo the irreducible polynomial $x^8+x^4+x^3+x+1$, while the addition on polynomials in GF(2^8) corresponds to a simple bitwise EXOR ($\oplus$). This makes the addition and multiplication closed in $GF(2^8)$. Accordingly, the multiplicative inverse a^{-1} is defined by

$$a \times a^{-1} \equiv \; 1 \quad (mod \;\; x^8 + x^4 + x^3 + x + 1),$$

where the value 00 is mapped onto itself.

Only with this multiplicative inverse, the algebraic expression of ByteSub is so simple that it could mount attacks such as interpolation attacks. Therefore, the second transformation is then employed.

2. Employing an invertible affine transformation over GF(2), defined by:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

For decryption, the inverse operation *InvByteSub* applies the inverse S-Box to the bytes of the state. The inverse S-Box is obtained by taking the inverse of the affine mapping and then taking its multiplicative inverse in GF(2^8). The inverse of the affine mapping is defined by:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2.4.2 ShiftRow Transformation

The ShiftRow transformation is a byte transposition that cyclically shifts the rows of the state over different offsets. As specified in AES[1], for the 16-byte block size , row 0 of the state vector is left shifted over $C_0 = 0$ byte , row 1 over $C_1 = 1$ byte, row 2 over $C_2 = 2$ bytes, row 3 over $C_3 = 3$ bytes. Mathematically, the byte at position j in row i $a_{i,j}$ moves to position $a_{i,(j+4-C_i)mod4}$.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \xrightarrow{ShiftRows} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix}$$

The inverse operation *InvShiftRow* is also a cyclic shift of the 3 bottom row bytes by

[1] Different for Rijndael, because different block sizes N_b determine different shift offsets

the same offsets in the opposite direction.

2.4.3 MixColumn Transformation

In MixColumn transformation, Rijndael considers the columns of the State as polynomials with coefficients over GF(2^8), and then multiplies it with a fixed polynomial $c(x) =$ '03'x^3+ '01'x^2+ '01'x+ '02' modulo (x^4+1) . Here, $c(x)$ is invertible because it is coprime to x^4+1.

This modular multiplication with a fixed polynomial can be written as a matrix multiplication. Let the output of a MixColumn transformation $b(x) = c(x) \otimes a(x)$ (mod x^4+1) , where a polynomial $a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x^1 + a_0 \cdot x$ represents individual columns of the state. Then the matrix multiplication is of the form,

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

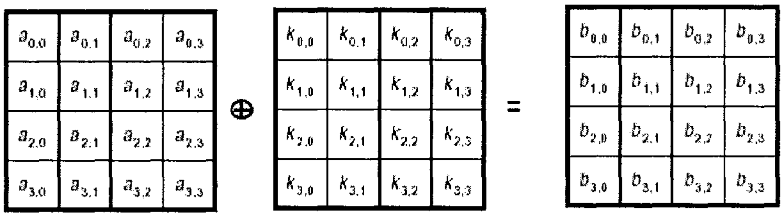
The inverse operation *InvMixColumn* is similar to MixColumn. Every column is multiplied with a fixed multiplication polynomial $d(x)$ defined by $c(x) \otimes d(x) =$ '01' (mod x^4+1). Then, it's given by $d(x) =$ '0B'x^3+ '0D'x^2+ '09'x+ '0E' .

Accordingly, the matrix multiplication form of InvMixColumn is as follows:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

2.4.4 AddRoundKey Transformation

The AddRoundKey transformation modifies the state by applying a simple bitwise EXOR of the state bytes and the round key bytes. The round key is derived from the original cipher key by means of the key schedule. And its length is equal to the block length.



$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

Figure 3: AddRoundKey Transformation

AddRoundKey is its own inverse.

2.5 Key Schedule

The round keys are derived from the cipher key by the key schedule which consists of two components: the key expansion and the round key selection.

1. **Key Expansion**

The key expansion specifies how the cipher key is expended to a total number of bits, equal to the block length multiplied by the number of rounds plus 1, since the cipher requires one round key for the initial key addition and one for each of the rounds. The expanded key can be considered as a linear array of 4-byte words, denoted by $W[N_b(N_r+1)]$. Thus, for the AES with a 128-bit block, the expanded key is denoted by $W[4(N_r+1)]$ and of 1408 bits (128 x (10 + 1)).

There are two versions of the key expansion function: one for N_k equal to or below 6 and one for N_k above 6. For $N_k \leq 6$, a pseudo C notation for the Key Expansion process is given as follows:

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)], Nk)
{
    word temp;
    for(i = 0; i < $N_k$; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++)
    {
        temp = W[i-1];
        if (i % Nk == 0)
            temp = SubByte(RotByte(temp))$\wedge Rcon[i/Nk]$;
        W[i] = W[i - Nk]$\wedge temp$;
    }
}
```

In the above description, the *SubByte(W)* function returns a 4-byte word, where each byte is the result of applying the S-Box to the corresponding byte of the input word. The *Rotbyte(W)* function returns a word wherein the bytes are a cyclic permutation of the input word such that input word (a_0, a_1, a_2, a_3) returns the output word (a_1, a_2, a_3, a_0). The round constant word array *RCon(i)* is independent of N_k, and defined by a recursion rule in $GF(2^8)$:

RCon(1) = ['01','00','00','00']

RCon(i) = [x,'00','00','00']$\cdot$ RCon(i-1) = [x^{i-1},'00','00','00'], with x^{i-1} being powers of x in the field GF(2^8).

The first N_k words are filled with the Cipher Key. The following words are defined recursively in terms of previous defined words. The recursion function depends on the position of the word. If i is not a multiple of N_k, *W[i]* is the EXOR of the previous word *W[i-1]* and the word N_k positions earlier *W[i-N_k]*. Otherwise, *W[i]* is the EXOR of *W[i-N_k]* and a nonlinear function of *W[i-1]*. The nonlinear function is realized by a cyclic shift of bytes of the word (*RotByte*), a non-linear substitution of the bytes of the word (*SubByte*), and then an addition with a round constant (*RCon*).

For the scheme for $N_k \succ 6$, if i mod N_k=4, SubByte also is applied to $W[i-1]$ prior to the EXOR.

2. **Round Key selection**

The Round Key i is given by the Expanded Key words W[4i] to W[4i+3]. This is illustrated in the following figure.

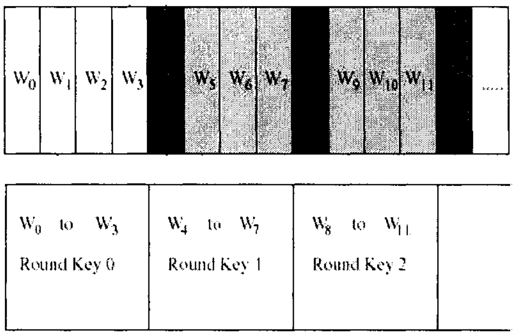


Figure 4: The Round Key Selection for $N_b = 4$ and $N_k = 4$

2.6 The Cipher

The AES cipher thus consists of an initial RoundKey addition, $N_r - 1$ Rounds and a FinalRound. In pseudo C code, the cipher is given below:

```
AES(State,CipherKey)
{
    KeyExpansion(CipherKey,ExpandedKey);
    AddRoundKey(State, ExpandedKey);
    For( i=1 ; i<Nr ; i++)
        Round(State, ExpandedKey + Nb*i);
    FinalRound(State,ExpandedKey + Nb*Nr);
}
```

And Figure 5 gives a flow chart of the cipher:

2.7 The Inverse Cipher

The algorithm for the inverse round can be found in a straightforward way by using the inverse transformations *InvByteSub, InvShiftRow, InvMixColumn* and *AddRoundKey* in a reverse order. Accordingly, in the inverse *InvFinalRound* there is no InvMixColumn transformation. A pseudo C notation is given as follows:

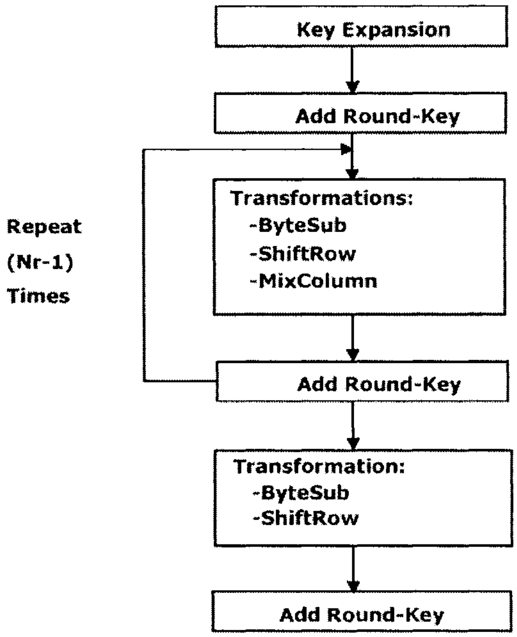


Figure 5: The AES Cipher Flow Chart

```
InvRound(State,RoundKey)

{

    AddRoundKey(State,RoundKey);

    InvMixColumn(State);

    InvShiftRow(State);

    InvByteSub(State);

}

InvFinalRound(State,RoundKey)

{

    AddRoundKey(State,RoundKey);

    InvShiftRow(State);

    InvByteSub(State);

}
```

For implementation reasons, it is often anticipated that the only nonlinear step (S-box substitution) is the first step of the round transformation and the sequence of inverse transformations is equal to that of the cipher itself. Because the sequence of InvShiftRow and InvByteSub is indifferent, these two transformations can easily commute. Moreover, because the order of AddRoundKey and InvmixColumn can be inverted if the round key is adapted accordingly, we can regroup the inverse Round and FinalRound in the same sequence as Round and FinalRound. Then, the inverse algorithm can be represented in pseudo C notation:

```
InvAES(State,CipherKey){
    InvKeyExpansion(CipherKey,InvExpandedKey);
    AddRoundKey(State,InvExpandedKey + $N_b * N_r$);
    For( i=Nr-1 ; i>0 ; i–)
        InvRound(State, InvExpandedKey + Nb*i);
    InvFinalRound(State,InvExpandedKey);
}
InvKeyExpansion(CipherKey,InvExpandedKey){
    KeyExpansion(CipherKey,InvKeyExpandedKey);
    For( i = 1 ; i< $N_r$; i++)
        InvMixColumn(InvExpandedKey + $N_b$*i);
}
InvRound(State,RoundKey){
    InvByteSub(State);
    InvShiftRow(State);
    InvMixColumn(State);
    AddRoundKey(State,RoundKey);}
InvFinalRound(State,RoundKey){
    InvByteSub(State);
    InvShiftRow(State);
    AddRoundKey(State,RoundKey);}
```

Chapter 3

The Specifications of BES

This chapter starts with explaining the common mathematical framework which is required to establish the relationship of the AES and the BES, and then gives a detailed description of the BES structure based on the BES specifications [7] and the steps of its round transformation, and concludes with an integral algebraic representation of a round transformation of BES.

3.1 Embedding the AES into the BES

This section describes the common mathematical framework and the basic techniques required to establish the relationship between the AES and the BES.

3.1.1 Mathematical Framework

1. **State Space**

Here we use the same polynomial representation of GF(2^8)[6]. GF(2^8) is a Galois field, wherein both addition (defined as a simple bitwise EXOR) and multiplication (defined as the algebraic product of the polynomials modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$) are closed. Bytes are regarded as polynomials[1] of this binary field denoted by F.

Both AES and the new cipher BES are defined in terms of a state vector of bytes, which is transformed by the round operations. In both ciphers, the plaintext is the input state vector, and the ciphertext is the output state vector. However, while AES has a 16-byte message and key space, BES is defined to have a 128-byte message and key space. We denote the state space of AES by A, the state space of BES by B, and a subset of the BES space that corresponds to the AES by B_A, so

A	State space of the AES	Vector space F^{16}
B	State space of the BES	Vector space F^{128}
B_A	Subset of B corresponding to A	Subset of F^{128}

Table 2: The State Space

Moreover, instead of describing the state vector by using an array of bytes, here we represent the state as a column vector, so the operations on the state vector are represented by matrix multiplications of such a column vector occurring on the left.

2. Vector Inversion

We can easily define the inversion operation.

For $a \in F$, it is identical to the field multiplicative inversion for non-zero elements

[1] The set of all possible byte values corresponds to the set of all polynomials with degree less than 8 and coefficients in the finite field GF(2).

with $0^{-1} = 0$.

For an n-dimensional vector $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}) \in F^n$, the vector inversion can be viewed as a componentwise operation:

$$\mathbf{a}^{(-1)} = (a_0^{(-1)}, a_1^{(-1)} \ldots\ldots a_{n-1}^{(-1)})$$

3. **Vector Conjugates**

For any element $a \in F$, we also define that the vector conjugate of a consists of the eight GF (2)-conjugates of a, denoted by

$$\tilde{a} = (a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7})$$

with a^{2^k} being powers of a in the field F.

If using a vector conjugate mapping ϕ from F^n to F^{8n} to represent the vector conjugate of $a \in F$, while $n = 1$, we have

$$\tilde{a} = \phi(a) = (a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7})$$

Accordingly, an n-dimensional vector $\mathbf{a} = (a_0, \ldots, a_{n-1}) \in F^n$ is mapped to

$$\tilde{\mathbf{a}} = \phi(\mathbf{a}) = (\phi(a_0), \ldots, \phi(a_{n-1})),$$

while

$$\phi(a_0) = (a_0^{2^0}, a_0^{2^1}, a_0^{2^2}, a_0^{2^3}, a_0^{2^4}, a_0^{2^5}, a_0^{2^6}, a_0^{2^7}), \ldots,$$

$$\phi(a_{n-1}) = (a_{n-1}^{2^0}, a_{n-1}^{2^1}, a_{n-1}^{2^2}, a_{n-1}^{2^3}, a_{n-1}^{2^4}, a_{n-1}^{2^5}, a_{n-1}^{2^6}, a_{n-1}^{2^7})$$

Obviously, this vector conjugate mapping ϕ extends the vector space from F^n to a subset of F^{8n}. Moreover, ϕ is additive and preserves inverses, so

$$\phi(a+a^{'}) = \phi(a)+\phi(a^{'}) \text{ and } \phi(a^{-1}) = \phi(a)^{-1}$$

Also, $\phi^{-1} : F^{8n} \longrightarrow F^n$ can be given to extract the basic vector from a vector conjugate.

4. **Embedding the AES state space in the BES state space.**

   By using the above definitions of the vector inversion and the vector conjugate mapping ϕ, any element[2] of the AES state space A can be embedded into the BES state space B. We define $B_A = \phi(A) \subset B$ to be the AES subset of BES. Therefore, elements of B_A are embedded images of AES states or subkeys in the BES state space. Furthermore, B_A is also additive and preserves inverses.

   The following figure shows the relationship between the AES and the BES based on the mathematical framework we discussed above:

3.1.2 The Basic Cipher Structure of AES

The AES considered here is the basic version of which encrypts a 16-byte block using a 16-byte key within 10 encryption rounds, referring to FIPS 197 [3]. A typical round of

[2] Plaintext, ciphertext, intermediate text, or subkey are the elements of the state space.

$$\begin{tikzcd}
 & A \arrow[r, "\phi"] \arrow[d] & B_A \arrow[d] & \\
k \arrow[r] & \boxed{AES} \arrow[d] & \boxed{BES} \arrow[d] & \phi(k) \arrow[l] \\
 & A & B_A \arrow[l, "\phi^{-1}"'] &
\end{tikzcd}$$

Figure 6: The Relationship between the AES and the BES

AES is defined in terms of the following three significant transformations:

1. **The AES non-linear layer:** The value of each byte in the state vector is substituted according to a S table look-up, denote by *S-Box*. This S-Box consists of three transformations.

   - The input x is transformed by the field multiplicative inverse function, $x \longrightarrow y = x^{(-1)}$, defined by

   $$x^{(-1)} = x^{254} = \begin{cases} x^{(-1)} & x \neq 0 \\ 0 & x = 0 \end{cases}$$

   - The intermediate value y is considered as a 8-dimension GF(2)-vector and multiplied by an (8 × 8) GF (2)-matrix L_A.

   - The result vector the AES S-Box is ($L_A \cdot$ y) + 63, where addition is with respect to GF (2). The output then is regarded as an element of GF(2^8).

     The second and third steps are GF$(2)^8$ transformations.

2. **The AES linear mixing layer:** This layer involves *ShiftRow* and *MixColumn* transformations. For *ShiftRow*, each row of the array is cyclically left shifted by a certain

number of byte positions. For *MixColumn*, each column of the array is viewed as an F-vector and transformed to the column C $\cdot$ y, where C is a (4×4) F-matrix.

3. **The AES key addition layer:** Each byte of the state vector is added to a byte from the corresponding position of a round key, with respect to GF (2).

As described before, instead of viewing the input to the AES round function as an array of bytes , here we can view it as a column vector of bytes. Moreover, the additive constant (63) in the AES S-Box can be removed by combining it within a slightly modified round key. Furthermore, each basic operation in a round of AES describes a bijective mapping on A, so they can be easily replaced with similar operations in the BES.

Therefore, by recasting AES in this way , we ensure that every operation (including the GF(2)-linear map from the AES S-box) can be represented using simple algebraic operations over F, with respecting the design principle of AES. Now, the round transformation of BES can be readily described as consisting of exclusively of inversion in GF(2^8), matrix multiplication in GF (2^8), and key addition in GF (2^8).

3.2 The Big Encryption System

The BES is a 128-byte iterated block cipher, which is composed of only simple algebraic operations over GF(2^8) with fully respecting encryption in the AES. This section further discusses the basic structure of BES.

3.2.1 State Vector

As previously specified, the state vector a of the AES is viewed as an element of A and represented here as a column vector.

$$a = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$
$$= \begin{pmatrix} a_{00}, & a_{10}, & a_{20}, & a_{30}, & a_{01}, & \ldots, & a_{31}, & a_{02}, & \ldots, a_{32}, & a_{03}, & a_{13}, & a_{23}, & a_{33} \end{pmatrix}^T$$

By using the embedding mapping ϕ on each byte of the state vector a, we have

$$\phi(a_{ij}) = (b_{ij0}, \ldots, b_{ij7})$$

Therefore, for the BES, the state vector $b \in B$ can be represented as a column vector:

$$b = \begin{pmatrix} b_{000}, & b_{001}, & \ldots, & b_{007}, & b_{100}, & b_{101}, & \ldots, & b_{107}, & \ldots, & b_{330}, & b_{331}, & \ldots, & b_{337} \end{pmatrix}^T$$

3.2.2 Round Transformation

Described in the previous section, the round transformation of BES is similar to that of AES.

```
Round(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State,RoundKey);
}
FinalRound(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    AddRoundKey(State,RoundKey);
}
```

Since the last round is just slightly different (MixColumn is removed), we only need to concentrate on a typical round transformation.

1. **Roundkey Addition**

   For both the AES and the BES, the round key is viewed as an element of the state space, and then the key addition is to combine the state vector with a round key with the bitwise EXOR operation. Thus, for a round key $(k_A)_i \in A$[3], the AES state vector $a \in A$ is transformed by $a \rightarrow a + (k_A)_i$; for a round key $(k_B)_i \in B$, the BES state vector $b \in B$ is modified by $b \rightarrow b + (k_B)_i$.

[3] i is the number of rounds, $0 \leq i \leq 10$

2. **ShiftRow Transformation**

In AES, the ShiftRow operation is to cyclically shift the rows of the state array over different bytes. Because of considering the state vector as a column vector $a \in A$ here, we also consider this transformation as a transformation of the components of a column vector. Accordingly, it is readily represented by a modular multiplication of the state vector $a \in A$ by a (16×16) matrix R_A. The same holds in BES. The ShiftRow transformation is then represented by a modular multiplication of the state vector $b \in B$ by a (128×128) matrix R_B. Furthermore, in order to deduce R_B from R_A, the vector conjugates should be moved as an integral entity .

3. **MixColumn Transformation**

In the AES MixColumn transformation, each column $y \in F^4$ of the state array is multiplied by a (4×4) matrix C_A. Let $z \in F^4$ be the output column,

$$z = C_A \cdot y = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot y$$

By translating this operation of the AES state array into an operation of the AES state column vector, MixColumn can be viewed as a modular multiplication of the state column vector by (16×16) matrix Mix_A.

In order to replicate the equivalent operation within BES, we define eight versions of the matrix C_A, denoted by $C_B^{(k)}$, for $k = 0, 1, \ldots 7$,

$$C_B^{(k)} = \begin{pmatrix} (02)^{2^k} & (03)^{2^k} & 01 & 01 \\ 01 & (02)^{2^k} & (03)^{2^k} & 01 \\ 01 & 01 & (02)^{2^k} & (03)^{2^k} \\ (03)^{2^k} & 01 & 01 & (02)^{2^k} \end{pmatrix}$$

where $C_B^{(0)} = C_A$. Because C_B is an MDS[4] matrix, it has certain diffusion properties [8]. Thus, if

$$(z_0, z_1, z_2, z_3)^T = C_A \cdot (y_0 \cdot y_1 \cdot y_2 \cdot y_3)^T \text{ then}$$

$$(z_0^{2^k}, z_1^{2^k}, z_2^{2^k}, z_3^{2^k})^T = C_B^{(k)} \cdot (y_0^{2^k} \cdot y_1^{2^k} \cdot y_2^{2^k} \cdot y_3^{2^k})^T$$

Now by using these matrices $C_B^{(k)}$ and the vector conjugate embedding mapping ϕ, the (128 × 128) F-matrix Mix_B can be defined to replicate the MixColumn operation in the AES by a multiplication of the state vector $b \in B$ with Mix_B in the BES.

4. **ByteSub Transformtion**

   As previously described, we consider the ByteSub step (S-Box) as being composed of two transformations.[5]

   (a) S-Box Inversion

[4] MDS stands for Maximal Distance Separable

[5] The additive constant 63 is removed by combing it within a slightly modified round key.

As defined before, the vector inversion is a componentwise operation on bytes, so it is straightforward to describe this S-Box inversion for both the BES and the AES. In the AES, it can be viewed as a componentwise vector inversion of the state vector $a \in A$, defined by $a \longrightarrow a^{(-1)}$. Accordingly, for the state vector $b \in B$, this step is defined by $b \longrightarrow b^{(-1)}$ in the BES.

(b) The S-box GF (2)-linear operation

In the AES specification, this operation operates on each component of the state vector by considering each component $a_{ij} \in$ GF(2^8) consisting of bit $x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7$ as a 8-dimension GF(2)-vector. This transformation, denoted by f from GF(2^8) to GF$(2)^8$, is then represented by a multiplication of each component by an (8×8) GF(2) matrix L_A.

$$f(a_{ij}) = L_A \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

In this transformation, the natural mapping $\psi : GF(2^8) \longrightarrow GF(2)^8$ is involved by default. Therefore, the componentwise AES GF (2)-linear operation $f : F \rightarrow F$ is then defined by $f(a) = \psi^{-1}(L_A \cdot (\psi(a)))$ for $a \in F$, where ψ and ψ^{-1} bring a lot of complexity to the analysis of the AES.

However, by applying *Lagrange interpolation* to $f : F \rightarrow F$, f can be simply represented by a polynomial function over F . Thus, the polynomial representation with coefficients in F is

$$f(a_{ij}) = \sum_{k=0}^{7} \lambda_k(a_{ij}^{2^k}) \quad for \quad a \in F$$

$$where \quad (\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7) = (05, 09, f9, 25, f4, 01, b5, 8f).$$

This polynomial is slightly different from the S-Box interpolation polynomial [6][8], because we separate the inversion out from the rest of the S-Box function.

Now the equivalent operation in BES can be defined. First, by using an (8 × 8) matrix L_B, we can replicate the AES GF(2)-linear map on the first byte of a vector conjugate set, where

[6] The S-Box interpolation polynomial is $f(a_{ij}) = \sum_{k=0}^{7} \lambda_k(a_{ij}^{2^k})^{(-1)} + 63$

$$L_B = \begin{pmatrix} (\lambda_0)^{2^0} & (\lambda_1)^{2^0} & (\lambda_2)^{2^0} & (\lambda_3)^{2^0} & (\lambda_4)^{2^0} & (\lambda_5)^{2^0} & (\lambda_6)^{2^0} & (\lambda_7)^{2^0} \\ (\lambda_7)^{2^1} & (\lambda_0)^{2^1} & (\lambda_1)^{2^1} & (\lambda_2)^{2^1} & (\lambda_3)^{2^1} & (\lambda_4)^{2^1} & (\lambda_5)^{2^1} & (\lambda_6)^{2^1} \\ (\lambda_6)^{2^2} & (\lambda_7)^{2^2} & (\lambda_0)^{2^2} & (\lambda_1)^{2^2} & (\lambda_2)^{2^2} & (\lambda_3)^{2^2} & (\lambda_4)^{2^2} & (\lambda_5)^{2^2} \\ (\lambda_5)^{2^3} & (\lambda_6)^{2^3} & (\lambda_7)^{2^3} & (\lambda_0)^{2^3} & (\lambda_1)^{2^3} & (\lambda_2)^{2^3} & (\lambda_3)^{2^3} & (\lambda_4)^{2^3} \\ (\lambda_4)^{2^4} & (\lambda_5)^{2^4} & (\lambda_6)^{2^4} & (\lambda_7)^{2^4} & (\lambda_0)^{2^4} & (\lambda_1)^{2^4} & (\lambda_2)^{2^4} & (\lambda_3)^{2^4} \\ (\lambda_3)^{2^5} & (\lambda_4)^{2^5} & (\lambda_5)^{2^5} & (\lambda_6)^{2^5} & (\lambda_7)^{2^5} & (\lambda_0)^{2^0} & (\lambda_1)^{2^5} & (\lambda_2)^{2^5} \\ (\lambda_2)^{2^6} & (\lambda_3)^{2^6} & (\lambda_4)^{2^6} & (\lambda_5)^{2^6} & (\lambda_6)^{2^6} & (\lambda_7)^{2^6} & (\lambda_0)^{2^6} & (\lambda_1)^{2^6} \\ (\lambda_1)^{2^7} & (\lambda_2)^{2^7} & (\lambda_3)^{2^7} & (\lambda_4)^{2^7} & (\lambda_5)^{2^7} & (\lambda_6)^{2^7} & (\lambda_7)^{2^7} & (\lambda_0)^{2^7} \end{pmatrix}$$

Then, by applying a (128 × 128) F-matrix, Lin_B, the entire set of GF (2)-linear map in the AES can be represented within the BES.

5. **Key Schedule**

While the AES key schedule produces eleven 16-byte round keys $(k_A)_i \in A$, the key schedule provides eleven 128-byte BES round key $(k_B)_i \in B$.

The key schedule in the AES involves the same operations as the AES round transformation, namely the GF (2)-linear map, componentwise inversion, byte rotation, and addition. Therefore, we can simply apply the techniques discussed in previous sections to describe the key schedule for the BES. Thus the key schedule can also be described using the same simple algebraic operations over F.

3.2.3 Algebraic Representation of the BES Round Function

For the input state vector $b \in B$ and the round key $(k_B)_i \in B$, the BES round function can be given by

$$Round_B(b,(k_B)_i) = Mix_B(R_B(lin_B(b^{(-1)}))) + (k_B)_i = M_B \cdot (b^{(-1)}) + (k_B)_i$$

where M_B is a (128 × 128) matrix performing linear diffusion within the BES.

So far, the round function of BES, and hence essentially the AES can be simply represented by a componentwise inversion and an affine transformation within the same field $GF(2^8)$. Furthermore, both of them are simple algebraic transformations over $B = F^{128}$, so each of them describes an isomorphic algebraic curve on B, which preserves algebraic curves of AES over A with a reasonable probability.

Chapter 4

The Implementation of BES Encryption

Based on the design principles of BES cipher [7], this chapter is an original contribution which detailedly explores the issues related to the implementation of BES encryption. While the section 4.1 deals with the input and the output data structure of the cipher, the section 4.2 is grouped by a detailed description of each operation in a typical BES encryption round.

4.1 State Vector

As specified in the previous chapter, by using the vector conjugate mapping ϕ from F to a subset of F^8,

$$\phi(a) = (a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7}), \quad for \quad a \in F$$

$$where \quad a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7} \quad are \quad GF(2)-conjugates \quad of \quad a,$$

the 16-byte messages and keys in AES can be extended into the 128-byte messages and keys in BES. By considering both the state vector $a \in A$ and the state vector $b \in B$ as column vectors, we have

$$a = \left(a_{00},\ a_{10},\ a_{20},\ a_{30},\ a_{01},\ \ldots,\ a_{31},\ a_{02},\ \ldots, a_{32},\ a_{03},\ a_{13},\ a_{23},\ a_{33} \right)^T$$
$$\xrightarrow{\phi} b = \left(\phi(a_{00}),\ \cdots,\ \phi(a_{30}),\ \phi(a_{01}),\ \cdots,\ \phi(a_{31}),\ \cdots,\ \phi(a_{03}), \cdots\ \phi(a_{33}), \right)^T$$

$$where\quad \phi(a_{ij}) = (a_{ij}^{2^0}, a_{ij}^{2^1}, a_{ij}^{2^2}, a_{ij}^{2^3}, a_{ij}^{2^4}, a_{ij}^{2^5}, a_{ij}^{2^6}, a_{ij}^{2^7}),\quad 0 \leq i, j \leq 3$$

$$b = \left(a_{00},\ a_{00}^2,\ a_{00}^4,\ a_{00}^8,\ a_{00}^{16},\ a_{00}^{32},\ a_{00}^{64},\ a_{00}^{128},\ \cdots,\ a_{33},\ a_{33}^2,\ \cdots,\ a_{33}^{128} \right)^T,$$

where $a_{ij}^{2^0}, a_{ij}^{2^1}, a_{ij}^{2^2}, a_{ij}^{2^3}, a_{ij}^{2^4}, a_{ij}^{2^5}, a_{ij}^{2^6}, a_{ij}^{2^7}$ are the powers of a_{ij} in F.

4.2 Round Function of BES

This section deals with the issues related to the implementation of a typical BES encryption.

4.2.1 Round Transformation

The round transformations of BES has the similar operations to that of AES. Therefore, a pseudo-C notation can be given as follows:

```
Round(State,RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State,RoundKey);
}
```

4.2.2 ByteSub Operation

As described before, this step in AES involves three transformations : a S-Box inversion, a GF(2)-linear operation and a constant addition. However, the additive constant is removed from this step by incorporating it into slightly modified round keys, so here we only focus on the first two transformations.

1. **S-Box Inversion**

   The S-Box inversion in AES is identical to the standard field inversion in F on each byte of the state vector $a_{ij} \in F$: $a_{ij} \rightarrow a_{ij}^{(-1)}$ for non-zero field elements with $0^{(-1)} = 0$.

   Apparently, in the BES, the S-Box inversion on the state vector $b \in A$ can be viewed as a componentwise standard field inversion on each GF(2)-conjugates of $a_{ij} \in F$:

$$b = \phi(a_{ij}) = (a_{ij}, a_{ij}^{2}, a_{ij}^{4}, a_{ij}^{8}, a_{ij}^{16}, a_{ij}^{32}, a_{ij}^{64}, a_{ij}^{128})$$

$$\longrightarrow b^{(-1)} = (\phi(a_{ij}))^{(-1)} = (a_{ij}, a_{ij}^{2}, a_{ij}^{4}, a_{ij}^{8}, a_{ij}^{16}, a_{ij}^{32}, a_{ij}^{64}, a_{ij}^{128})^{(-1)}$$

$$= (a_{ij}^{(-1)}, (a_{ij}^{(-1)})^{2}, (a_{ij}^{(-1)})^{4}, (a_{ij}^{(-1)})^{8}, (a_{ij}^{(-1)})^{16}, (a_{ij}^{(-1)})^{32}, (a_{ij}^{(-1)})^{64}, (a_{ij}^{(-1)})^{128})$$

$$= \phi(a_{ij}^{(-1)}) \quad for \quad 0 \leq i, j \leq 3$$

2. **S-box GF (2)-linear Operation**

In AES, the S-box GF(2)-linear operation $f : F \rightarrow F$ on each byte of the state vector is represented by a matrix multiplication of each byte with a (8×8) GF (2) matrix L_A, wherein each byte $a_{ij} \in F$ consisting of bits $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$ is considered as a GF(2) vector of 8 dimensions.

$$f(a_{ij}) = L_A \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

By using Lagrange interpolation formula to interpolate $f : F \longrightarrow F$, it can be represented by a polynomial function:

$$f(a_{ij}) = \sum_{k=0}^{7} \lambda_k (a_{ij})^{2^k} \quad for \quad a \in F \quad and \quad 0 \leq i, j \leq 3$$

$$where \quad (\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7) = (05, 09, f9, 25, f4, 01, b5, 8f).$$

Now we can define the GF(2)-linear operation by a matrix multiplication on 8-conjugate vectors with the following (8×8) matrix L_B:

$$\begin{pmatrix} 05 & 09 & f9 & 25 & f4 & 01 & b5 & 8f \\ (8f)^2 & (05)^2 & (09)^2 & (f9)^2 & (25)^2 & (f4)^2 & (01)^2 & (b5)^2 \\ (b5)^4 & (8f)^4 & (05)^4 & (09)^4 & (f9)^4 & (25)^4 & (f4)^4 & (01)^4 \\ (01)^8 & (b5)^8 & (8f)^8 & (05)^8 & (09)^8 & (f9)^8 & (25)^8 & (f4)^8 \\ (f4)^{16} & (01)^{16} & (b5)^{16} & (8f)^{16} & (05)^{16} & (09)^{16} & (f9)^{16} & (25)^{16} \\ (25)^{32} & (f4)^{32} & (01)^{32} & (b5)^{32} & (8f)^{32} & (05)^{32} & (09)^{32} & (f9)^{32} \\ (f9)^{64} & (25)^{64} & (f4)^{64} & (01)^{64} & (b5)^{64} & (8f)^{64} & (05)^{64} & (09)^{64} \\ (09)^{128} & (f9)^{128} & (25)^{128} & (f4)^{128} & (01)^{128} & (b5)^{128} & (8f)^{128} & (05)^{128} \end{pmatrix} \begin{pmatrix} a_{ij} \\ a_{ij}^2 \\ a_{ij}^4 \\ a_{ij}^8 \\ a_{ij}^{16} \\ a_{ij}^{32} \\ a_{ij}^{64} \\ a_{ij}^{128} \end{pmatrix}$$

$$= \begin{pmatrix} 05 \cdot a_{ij} + 09 \cdot a_{ij}^2 + f9 \cdot a_{ij}^4 + 25 \cdot a_{ij}^8 + f4 \cdot a_{ij}^{16} + 01 \cdot a_{ij}^{32} + b5 \cdot a_{ij}^{64} + 8f \cdot a_{ij}^{128} \\ (8f)^2 \cdot a_{ij} + (05)^2 \cdot a_{ij}^2 + (09)^2 \cdot a_{ij}^4 + (f9)^2 \cdot a_{ij}^8 + (25)^2 \cdot a_{ij}^{16} + (f4)^2 \cdot a_{ij}^{32} + (01)^2 \cdot a_{ij}^{64} + (b5)^2 \cdot a_{ij}^{128} \\ (b5)^4 \cdot a_{ij} + (8f)^4 \cdot a_{ij}^2 + (05)^4 \cdot a_{ij}^4 + (09)^4 \cdot a_{ij}^8 + (f9)^4 \cdot a_{ij}^{16} + (25)^4 \cdot a_{ij}^{32} + (f4)^4 \cdot a_{ij}^{64} + (01)^4 \cdot a_{ij}^{128} \\ (01)^8 \cdot a_{ij} + (b5)^8 \cdot a_{ij}^2 + (8f)^8 \cdot a_{ij}^4 + (05)^8 \cdot a_{ij}^8 + (09)^8 \cdot a_{ij}^{16} + (f9)^8 \cdot a_{ij}^{32} + (25)^8 \cdot a_{ij}^{64} + (f4)^8 \cdot a_{ij}^{128} \\ (f4)^{16} \cdot a_{ij} + (01)^{16} \cdot a_{ij}^2 + (b5)^{16} \cdot a_{ij}^4 + (8f)^{16} \cdot a_{ij}^8 + (05)^{16} \cdot a_{ij}^{16} + (09)^{16} \cdot a_{ij}^{32} + (f9)^{16} \cdot a_{ij}^{64} + (25)^{16} \cdot a_{ij}^{128} \\ (25)^{32} \cdot a_{ij} + (f4)^{32} \cdot a_{ij}^2 + (01)^{32} \cdot a_{ij}^4 + (b5)^{32} \cdot a_{ij}^8 + (8f)^{32} \cdot a_{ij}^{16} + (05)^{32} \cdot a_{ij}^{32} + (09)^{32} \cdot a_{ij}^{64} + (f9)^{32} \cdot a_{ij}^{128} \\ (f9)^{64} \cdot a_{ij} + (25)^{64} \cdot a_{ij}^2 + (f4)^{64} \cdot a_{ij}^4 + (01)^{64} \cdot a_{ij}^8 + (b5)^{64} \cdot a_{ij}^{16} + (8f)^{64} \cdot a_{ij}^{32} + (05)^{64} \cdot a_{ij}^{64} + (09)^{64} \cdot a_{ij}^{128} \\ (09)^{128} \cdot a_{ij} + (f9)^{128} \cdot a_{ij}^2 + (25)^{128} \cdot a_{ij}^4 + (f4)^{128} \cdot a_{ij}^8 + (01)^{128} \cdot a_{ij}^{16} + (b5)^{128} \cdot a_{ij}^{32} + (8f)^{128} \cdot a_{ij}^{64} + (05)^{128} \cdot a_{ij}^{128} \end{pmatrix}$$

This matrix in obvious manner applies the GF(2)-linear operation of the AES S-Box on the first byte of a vector conjugate set and preserves the conjugacy property on the remaining bytes. Thus, the entire GF(2)-linear operation on BES can be represented by using a (128 $\times$128) F-matrix Lin_B.

From above, we can have the entire representation of ByteSub operation, denoted by S_{RD}, as follows :

$$S_{RD}(b) = Lin_B \cdot (b^{-1})$$

$$
= \left(\begin{array}{ccccccccccccccccc}
05 & 09 & f9 & 25 & f4 & 01 & b5 & 8f & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(8f)^2 & (05)^2 & (09)^2 & (f9)^2 & (25)^2 & (f4)^2 & (01)^2 & (b5)^2 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(b5)^4 & (8f)^4 & (05)^4 & (09)^4 & (f9)^4 & (25)^4 & (f4)^4 & (01)^4 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(01)^8 & (b5)^8 & (8f)^8 & (05)^8 & (09)^8 & (f9)^8 & (25)^8 & (f4)^8 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(f4)^{16} & (01)^{16} & (b5)^{16} & (8f)^{16} & (05)^{16} & (09)^{16} & (f9)^{16} & (25)^{16} & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(25)^{32} & (f4)^{32} & (01)^{32} & (b5)^{32} & (8f)^{32} & (05)^{32} & (09)^{32} & (f9)^{32} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(f9)^{64} & (25)^{64} & (f4)^{64} & (01)^{64} & (b5)^{64} & (8f)^{64} & (05)^{64} & (09)^{64} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(09)^{128} & (f9)^{128} & (25)^{128} & (f4)^{128} & (01)^{128} & (b5)^{128} & (8f)^{128} & (05)^{128} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & \vdots & & & & \ddots & & & & & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 05 & 09 & f9 & 25 & f4 & 01 & b5 & 8f \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (8f)^2 & (05)^2 & (09)^2 & (f9)^2 & (25)^2 & (f4)^2 & (01)^2 & (b5)^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (b5)^4 & (8f)^4 & (05)^4 & (09)^4 & (f9)^4 & (25)^4 & (f4)^4 & (01)^4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (01)^8 & (b5)^8 & (8f)^8 & (05)^8 & (09)^8 & (f9)^8 & (25)^8 & (f4)^8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & (f4)^{16} & (01)^{16} & (b5)^{16} & (8f)^{16} & (05)^{16} & (09)^{16} & (f9)^{16} & (25)^{16} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (25)^{32} & (f4)^{32} & (01)^{32} & (b5)^{32} & (8f)^{32} & (05)^{32} & (09)^{32} & (f9)^{32} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (f9)^{64} & (25)^{64} & (f4)^{64} & (01)^{64} & (b5)^{64} & (8f)^{64} & (05)^{64} & (09)^{64} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (09)^{128} & (f9)^{128} & (25)^{128} & (f4)^{128} & (01)^{128} & (b5)^{128} & (8f)^{128} & (05)^{128}
\end{array}\right)
\left(\begin{array}{c}
a_{00} \\ a_{00}^{2} \\ a_{00}^{4} \\ a_{00}^{8} \\ a_{00}^{16} \\ a_{00}^{32} \\ a_{00}^{64} \\ a_{00}^{128} \\ \vdots \\ a_{33} \\ a_{33}^{2} \\ a_{33}^{4} \\ a_{33}^{8} \\ a_{33}^{16} \\ a_{33}^{32} \\ a_{33}^{64} \\ a_{33}^{128}
\end{array}\right)^{-1}
$$

$$
= \begin{pmatrix}
05 & 09 & f9 & 25 & f4 & 01 & b5 & 8f & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(8f)^{2} & (05)^{2} & (09)^{2} & (f9)^{2} & (25)^{2} & (f4)^{2} & (01)^{2} & (b5)^{2} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(b5)^{4} & (8f)^{4} & (05)^{4} & (09)^{4} & (f9)^{4} & (25)^{4} & (f4)^{4} & (01)^{4} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(01)^{8} & (b5)^{8} & (8f)^{8} & (05)^{8} & (09)^{8} & (f9)^{8} & (25)^{8} & (f4)^{8} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(f4)^{16} & (01)^{16} & (b5)^{16} & (8f)^{16} & (05)^{16} & (09)^{16} & (f9)^{16} & (25)^{16} & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(25)^{32} & (f4)^{32} & (01)^{32} & (b5)^{32} & (8f)^{32} & (05)^{32} & (09)^{32} & (f9)^{32} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(f9)^{64} & (25)^{64} & (f4)^{64} & (01)^{64} & (b5)^{64} & (8f)^{64} & (05)^{64} & (09)^{64} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(09)^{128} & (f9)^{128} & (25)^{128} & (f4)^{128} & (01)^{128} & (b5)^{128} & (8f)^{128} & (05)^{128} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & \vdots & & & & \ddots & & & & & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 05 & 09 & f9 & 25 & f4 & 01 & b5 & 8f \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (8f)^{2} & (05)^{2} & (09)^{2} & (f9)^{2} & (25)^{2} & (f4)^{2} & (01)^{2} & (b5)^{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (b5)^{4} & (8f)^{4} & (05)^{4} & (09)^{4} & (f9)^{4} & (25)^{4} & (f4)^{4} & (01)^{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (01)^{8} & (b5)^{8} & (8f)^{8} & (05)^{8} & (09)^{8} & (f9)^{8} & (25)^{8} & (f4)^{8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & (f4)^{16} & (01)^{16} & (b5)^{16} & (8f)^{16} & (05)^{16} & (09)^{16} & (f9)^{16} & (25)^{16} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (25)^{32} & (f4)^{32} & (01)^{32} & (b5)^{32} & (8f)^{32} & (05)^{32} & (09)^{32} & (f9)^{32} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (f9)^{64} & (25)^{64} & (f4)^{64} & (01)^{64} & (b5)^{64} & (8f)^{64} & (05)^{64} & (09)^{64} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (09)^{128} & (f9)^{128} & (25)^{128} & (f4)^{128} & (01)^{128} & (b5)^{128} & (8f)^{128} & (05)^{128}
\end{pmatrix}
\begin{pmatrix}
(a_{00})^{-1} \\
(a_{00}^{2})^{-1} \\
(a_{00}^{4})^{-1} \\
(a_{00}^{8})^{-1} \\
(a_{00}^{16})^{-1} \\
(a_{00}^{32})^{-1} \\
(a_{00}^{64})^{-1} \\
(a_{00}^{128})^{-1} \\
\vdots \\
(a_{33})^{-1} \\
(a_{33}^{2})^{-1} \\
(a_{33}^{4})^{-1} \\
(a_{33}^{8})^{-1} \\
(a_{33}^{16})^{-1} \\
(a_{33}^{32})^{-1} \\
(a_{33}^{64})^{-1} \\
(a_{33}^{128})^{-1}
\end{pmatrix}
$$

$$= \begin{pmatrix} 05 \cdot a_{00}^{-1} + 09 \cdot (a_{00}^{2})^{-1} + f9 \cdot (a_{00}^{4})^{-1} + 25 \cdot (a_{00}^{8})^{-1} + f4 \cdot (a_{00}^{16})^{-1} + 01 \cdot (a_{00}^{32})^{-1} + b5 \cdot (a_{00}^{64})^{-1} + 8f \cdot (a_{00}^{128})^{-1} \\ (8f)^{2} \cdot (a_{00})^{-1} + (05)^{2} \cdot (a_{00}^{2})^{-1} + (09)^{2} \cdot (a_{00}^{4})^{-1} + (f9)^{2} \cdot (a_{00}^{8})^{-1} + (25)^{2} \cdot (a_{00}^{16})^{-1} + (f4)^{2} \cdot (a_{00}^{32})^{-1} + (01)^{2} \cdot (a_{00}^{64})^{-1} + (b5)^{2} \cdot (a_{00}^{128})^{-1} \\ (b5)^{4} \cdot (a_{00})^{-1} + (8f)^{4} \cdot (a_{00}^{2})^{-1} + (05)^{4} \cdot (a_{00}^{4})^{-1} + (09)^{4} \cdot (a_{00}^{8})^{-1} + (f9)^{4} \cdot (a_{00}^{16})^{-1} + (25)^{4} \cdot (a_{00}^{32})^{-1} + (f4)^{4} \cdot (a_{00}^{64})^{-1} + (01)^{4} \cdot (a_{00}^{128})^{-1} \\ (01)^{8} \cdot (a_{00})^{-1} + (b5)^{8} \cdot (a_{00}^{2})^{-1} + (8f)^{8} \cdot (a_{00}^{4})^{-1} + (05)^{8} \cdot (a_{00}^{8})^{-1} + (09)^{8} \cdot (a_{00}^{16})^{-1} + (f9)^{8} \cdot (a_{00}^{32})^{-1} + (25)^{8} \cdot (a_{00}^{64})^{-1} + (f4)^{8} \cdot (a_{00}^{128})^{-1} \\ (f4)^{16} \cdot (a_{00})^{-1} + (01)^{16} \cdot (a_{00}^{2})^{-1} + (b5)^{16} \cdot (a_{00}^{4})^{-1} + (8f)^{16} \cdot (a_{00}^{8})^{-1} + (05)^{16} \cdot (a_{00}^{16})^{-1} + (09)^{16} \cdot (a_{00}^{32})^{-1} + (f9)^{16} \cdot (a_{00}^{64})^{-1} + (25)^{16} \cdot (a_{00}^{128})^{-1} \\ (25)^{32} \cdot (a_{00})^{-1} + (f4)^{32} \cdot (a_{00}^{2})^{-1} + (01)^{32} \cdot (a_{00}^{4})^{-1} + (b5)^{32} \cdot (a_{00}^{8})^{-1} + (8f)^{32} \cdot (a_{00}^{16})^{-1} + (05)^{32} \cdot (a_{00}^{32})^{-1} + (09)^{32} \cdot (a_{00}^{64})^{-1} + (f9)^{32} \cdot (a_{00}^{128})^{-1} \\ (f9)^{64} \cdot (a_{00})^{-1} + (25)^{64} \cdot (a_{00}^{2})^{-1} + (f4)^{64} \cdot (a_{00}^{4})^{-1} + (01)^{64} \cdot (a_{00}^{8})^{-1} + (b5)^{64} \cdot (a_{00}^{16})^{-1} + (8f)^{64} \cdot (a_{00}^{32})^{-1} + (05)^{64} \cdot (a_{00}^{64})^{-1} + (09)^{64} \cdot (a_{00}^{128})^{-1} \\ (09)^{128} \cdot (a_{00})^{-1} + (f9)^{128} \cdot (a_{00}^{2})^{-1} + (25)^{128} \cdot (a_{00}^{4})^{-1} + (f4)^{128} \cdot (a_{00}^{8})^{-1} + (01)^{128} \cdot (a_{00}^{16})^{-1} + (b5)^{128} \cdot (a_{00}^{32})^{-1} + (8f)^{128} \cdot (a_{00}^{64})^{-1} + (05)^{128} \cdot (a_{00}^{128})^{-1} \\ \vdots \\ 05 \cdot (a_{33})^{-1} + 09 \cdot (a_{33}^{2})^{-1} + f9 \cdot (a_{33}^{4})^{-1} + 25 \cdot (a_{33}^{8})^{-1} + f4 \cdot (a_{33}^{16})^{-1} + 01 \cdot (a_{33}^{32})^{-1} + b5 \cdot (a_{33}^{64})^{-1} + 8f \cdot (a_{33}^{128})^{-1} \\ (8f)^{2} \cdot (a_{33})^{-1} + (05)^{2} \cdot (a_{33}^{2})^{-1} + (09)^{2} \cdot (a_{33}^{4})^{-1} + (f9)^{2} \cdot (a_{33}^{8})^{-1} + (25)^{2} \cdot (a_{33}^{16})^{-1} + (f4)^{2} \cdot (a_{33}^{32})^{-1} + (01)^{2} \cdot (a_{33}^{64})^{-1} + (b5)^{2} \cdot (a_{33}^{128})^{-1} \\ (b5)^{4} \cdot (a_{33})^{-1} + (8f)^{4} \cdot (a_{33}^{2})^{-1} + (05)^{4} \cdot (a_{33}^{4})^{-1} + (09)^{4} \cdot (a_{33}^{8})^{-1} + (f9)^{4} \cdot (a_{33}^{16})^{-1} + (25)^{4} \cdot (a_{33}^{32})^{-1} + (f4)^{4} \cdot (a_{33}^{64})^{-1} + (01)^{4} \cdot (a_{33}^{128})^{-1} \\ (01)^{8} \cdot (a_{33})^{-1} + (b5)^{8} \cdot (a_{33}^{2})^{-1} + (8f)^{8} \cdot (a_{33}^{4})^{-1} + (05)^{8} \cdot (a_{33}^{8})^{-1} + (09)^{8} \cdot (a_{33}^{16})^{-1} + (f9)^{8} \cdot (a_{33}^{32})^{-1} + (25)^{8} \cdot (a_{33}^{64})^{-1} + (f4)^{8} \cdot (a_{33}^{128})^{-1} \\ (f4)^{16} \cdot (a_{33})^{-1} + (01)^{16} \cdot (a_{33}^{2})^{-1} + (b5)^{16} \cdot (a_{33}^{4})^{-1} + (8f)^{16} \cdot (a_{33}^{8})^{-1} + (05)^{16} \cdot (a_{33}^{16})^{-1} + (09)^{16} \cdot (a_{33}^{32})^{-1} + (f9)^{16} \cdot (a_{33}^{64})^{-1} + (25)^{16} \cdot (a_{33}^{128})^{-1} \\ (25)^{32} \cdot (a_{33})^{-1} + (f4)^{32} \cdot (a_{33}^{2})^{-1} + (01)^{32} \cdot (a_{33}^{4})^{-1} + (b5)^{32} \cdot (a_{33}^{8})^{-1} + (8f)^{32} \cdot (a_{33}^{16})^{-1} + (05)^{32} \cdot (a_{33}^{32})^{-1} + (09)^{32} \cdot (a_{33}^{64})^{-1} + (f9)^{32} \cdot (a_{33}^{128})^{-1} \\ (f9)^{64} \cdot (a_{33})^{-1} + (25)^{64} \cdot (a_{33}^{2})^{-1} + (f4)^{64} \cdot (a_{33}^{4})^{-1} + (01)^{64} \cdot (a_{33}^{8})^{-1} + (b5)^{64} \cdot (a_{33}^{16})^{-1} + (8f)^{64} \cdot (a_{33}^{32})^{-1} + (05)^{64} \cdot (a_{33}^{64})^{-1} + (09)^{64} \cdot (a_{33}^{128})^{-1} \\ (09)^{128} \cdot (a_{33})^{-1} + (f9)^{128} \cdot (a_{33}^{2})^{-1} + (25)^{128} \cdot (a_{33}^{4})^{-1} + (f4)^{128} \cdot (a_{33}^{8})^{-1} + (01)^{128} \cdot (a_{33}^{16})^{-1} + (b5)^{128} \cdot (a_{33}^{32})^{-1} + (8f)^{128} \cdot (a_{33}^{64})^{-1} + (05)^{128} \cdot (a_{33}^{128})^{-1} \end{pmatrix}$$

4.2.3 ShiftRow Operation

In AES, the ShiftRow step is to cyclically shift the rows of the state vector over different bytes: row 0 of the state vector is shifted over 0 byte leftwards, row 1 over 1 byte, row 2 over 2 bytes, row 3 over 3 bytes as follows:

$$a = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \xrightarrow{ShiftRow} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix}$$

By considering the state vector as a column vector, the process can be pictured as the transformation of the components of a column vector $a \in A$:

$$a = \begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{pmatrix} \xrightarrow{ShiftRows} \begin{pmatrix} a_{00} \\ a_{11} \\ a_{22} \\ a_{33} \\ a_{01} \\ a_{12} \\ a_{23} \\ a_{30} \\ a_{02} \\ a_{13} \\ a_{20} \\ a_{31} \\ a_{03} \\ a_{10} \\ a_{21} \\ a_{32} \end{pmatrix}$$

Therefore we can represent this transformation by a matrix multiplication of the state vector $a \in A$ by a (16×16) F matrix R_A:

$$R_A \cdot a = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}
=
\begin{pmatrix} a_{00} \\ a_{11} \\ a_{22} \\ a_{33} \\ a_{01} \\ a_{12} \\ a_{23} \\ a_{30} \\ a_{02} \\ a_{13} \\ a_{20} \\ a_{31} \\ a_{03} \\ a_{10} \\ a_{21} \\ a_{32} \end{pmatrix}$$

Accordingly, the ShiftRow step in BES can be represented by a matrix multiplication of the state vector $b \in B$ by a (128×128) F matrix R_B. In order to preserve the vector conjugacy while extending matrix R_A to matrix R_B, the corresponding vector conjugates should be moved as an integral entity. Thus, if the component in R_A equals to 1, then we extend it to the following (8 × 8) F block in R_B;

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

if the component in R_A equals to 0, we extend it to

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

By extending the (16 × 16) F matrix R_A to the (128 × 128) F matrix R_B, now the ShiftRow step can be represented by a matrix multiplication as follows.

$$R_B \cdot b = \left(\begin{array}{ccccccccccccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & & & & & & & & \cdots & & & & & & & & & \cdots & & & & & & & & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
& & & & & & & & & & & & 1 & 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 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & & & & & & & & & & & & & \vdots \\
& & & & & & & & & & & & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
& & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
& & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
& & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
& & & \vdots & & & & & & & & & & & & & & & & & \ddots & & & & & & & & & & & & \vdots \\
& & & & & & & & & & & & & & & & & & & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& & & & & & & & & & & & & & & & & & & & & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
& & & & & & & & & & & & & & & & & & & & & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
& & & 0 & & & & & & & & & & & & \cdots & & & & & & 0 & 0 & 0 & 1 & 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 & 1 & 0 \\
& & & & & & & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\begin{pmatrix}
a_{00} \\ a_{00}^{2} \\ a_{00}^{4} \\ a_{00}^{8} \\ a_{00}^{16} \\ a_{00}^{32} \\ a_{00}^{64} \\ a_{00}^{128} \\
a_{10} \\ a_{10}^{2} \\ a_{10}^{4} \\ a_{10}^{8} \\ a_{10}^{16} \\ a_{10}^{32} \\ a_{10}^{64} \\ a_{10}^{128} \\
\vdots \\
a_{33} \\ a_{33}^{2} \\ a_{33}^{4} \\ a_{33}^{8} \\ a_{33}^{16} \\ a_{33}^{32} \\ a_{33}^{64} \\ a_{33}^{128}
\end{pmatrix}
=
\begin{pmatrix}
a_{00} \\ a_{00}^{2} \\ a_{00}^{4} \\ a_{00}^{8} \\ a_{00}^{16} \\ a_{00}^{32} \\ a_{00}^{64} \\ a_{00}^{128} \\
a_{11} \\ a_{11}^{2} \\ a_{11}^{4} \\ a_{11}^{8} \\ a_{11}^{16} \\ a_{11}^{32} \\ a_{11}^{64} \\ a_{11}^{128} \\
\vdots \\
a_{32} \\ a_{32}^{2} \\ a_{32}^{4} \\ a_{32}^{8} \\ a_{32}^{16} \\ a_{32}^{32} \\ a_{32}^{64} \\ a_{32}^{128}
\end{pmatrix}$$

4.2.4 MixColumn Operation

In AES, this step is a permutation operating on the state vector column by column, and is defined as a matrix multiplication of the state vector $a \in A$ by a (4×4) F matrix C_A, where y_i is a conceptual column of the state vector a, for $0 \leq i \leq 3$.

$$C_A \cdot a = C_A \cdot (y_0, y_1, y_2, y_3)$$

$$= \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$= \begin{pmatrix} 02 \cdot a_{00} + 03 \cdot a_{10} + a_{20} + a_{30} & 02 \cdot a_{01} + 03 \cdot a_{11} + a_{21} + a_{31} & 02 \cdot a_{02} + 03 \cdot a_{12} + a_{22} + a_{32} & 02 \cdot a_{03} + 03 \cdot a_{13} + a_{23} + a_{33} \\ a_{00} + 02 \cdot a_{10} + 03 \cdot a_{20} + a_{30} & a_{01} + 02 \cdot a_{11} + 03 \cdot a_{21} + a_{31} & a_{02} + 02 \cdot a_{12} + 03 \cdot a_{22} + a_{32} & a_{03} + 02 \cdot a_{13} + 03 \cdot a_{23} + a_{33} \\ a_{00} + a_{10} + 02 \cdot a_{20} + 03 \cdot a_{30} & a_{01} + a_{11} + 02 \cdot a_{21} + 03 \cdot a_{31} & a_{02} + a_{12} + 02 \cdot a_{22} + 03 \cdot a_{32} & a_{03} + a_{13} + 02 \cdot a_{23} + 03 \cdot a_{33} \\ 03 \cdot a_{00} + a_{10} + a_{20} + 02 \cdot a_{30} & 03 \cdot a_{01} + a_{11} + a_{21} + 02 \cdot a_{31} & 03 \cdot a_{02} + a_{12} + a_{22} + 02 \cdot a_{32} & 03 \cdot a_{03} + a_{13} + a_{23} + 02 \cdot a_{33} \end{pmatrix}$$

Similarly, we can consider the state vector as a column vector, and then this step can be considered as a matrix multiplication of the state vector $a \in A$ by a (16×16) F matrix Mix_A.

$$C_A \cdot a = \begin{pmatrix} 02 \cdot a_{00} + 03 \cdot a_{10} + a_{20} + a_{30} \\ a_{00} + 02 \cdot a_{10} + 03 \cdot a_{20} + a_{30} \\ a_{00} + a_{10} + 02 \cdot a_{20} + 03 \cdot a_{30} \\ 03 \cdot a_{00} + a_{10} + a_{20} + 02 \cdot a_{30} \\ 02 \cdot a_{01} + 03 \cdot a_{11} + a_{21} + a_{31} \\ a_{01} + 02 \cdot a_{11} + 03 \cdot a_{21} + a_{31} \\ a_{01} + a_{11} + 02 \cdot a_{21} + 03 \cdot a_{31} \\ 03 \cdot a_{01} + a_{11} + a_{21} + 02 \cdot a_{31} \\ 02 \cdot a_{02} + 03 \cdot a_{12} + a_{22} + a_{32} \\ a_{02} + 02 \cdot a_{12} + 03 \cdot a_{22} + a_{32} \\ a_{02} + a_{12} + 02 \cdot a_{22} + 03 \cdot a_{32} \\ 03 \cdot a_{02} + a_{12} + a_{22} + 02 \cdot a_{32} \\ 02 \cdot a_{03} + 03 \cdot a_{13} + a_{23} + a_{33} \\ a_{03} + 02 \cdot a_{13} + 03 \cdot a_{23} + a_{33} \\ a_{03} + a_{13} + 02 \cdot a_{23} + 03 \cdot a_{33} \\ 03 \cdot a_{03} + a_{13} + a_{23} + 02 \cdot a_{33} \end{pmatrix}$$

$$
= \begin{pmatrix}
02 & 03 & 01 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
01 & 02 & 03 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
01 & 01 & 02 & 03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
03 & 01 & 01 & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 02 & 03 & 01 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 01 & 02 & 03 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 01 & 01 & 02 & 03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 03 & 01 & 01 & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 02 & 03 & 01 & 01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 02 & 03 & 01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 01 & 02 & 03 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 03 & 01 & 01 & 02 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 02 & 03 & 01 & 01 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 02 & 03 & 01 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 01 & 02 & 03 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 03 & 01 & 01 & 02
\end{pmatrix}
\begin{pmatrix}
a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33}
\end{pmatrix}
$$

$$
= Mix_A \cdot a
$$

From above deduction, Mix_A is a block diagonal matrix with 4 identical blocks of C_A.

$$Mix_A \cdot a = \begin{pmatrix} C_A & 0 & 0 & 0 \\ 0 & C_A & 0 & 0 \\ 0 & 0 & C_A & 0 \\ 0 & 0 & 0 & C_A \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

In order to define Mix_B to replicate the MixColumn step in BES, these 8 versions of C_A are defined as follows:

$$C_B^{(k)} = \begin{pmatrix} (02)^{2^k} & (03)^{2^k} & 1 & 1 \\ 1 & (02)^{2^k} & (03)^{2^k} & 1 \\ 1 & 1 & (02)^{2^k} & (03)^{2^k} \\ (03)^{2^k} & 1 & 1 & (02)^{2^k} \end{pmatrix}$$

for $k = 0, \cdots, 7$, where $C_B^{(0)} = C_A$.

Furthermore, as previously described, $C_B^{(k)}$ has certain diffusion properties:

if $(z_0, z_1, z_2, z_3)^T = C_A \cdot (y_0 \cdot y_1 \cdot y_2 \cdot y_3)^T$ then

$(z_0^{2^k}, z_1^{2^k}, z_2^{2^k}, z_3^{2^k})^T = C_B^{(k)} \cdot (y_0^{2^k} \cdot y_1^{2^k} \cdot y_2^{2^k} \cdot y_3^{2^k})^T$

where z_i is the output column of $C_A \cdot y_i$, for $0 \leq i \leq 3$.

Therefore, the following is true:

$$\begin{pmatrix} z_0 \\ z_0^2 \\ z_0^4 \\ z_0^8 \\ z_0^{16} \\ z_0^{32} \\ z_0^{64} \\ z_0^{128} \\ z_1 \\ z_1^2 \\ z_1^4 \\ z_1^8 \\ z_1^{16} \\ z_1^{32} \\ z_1^{64} \\ z_1^{128} \\ z_2 \\ z_2^2 \\ z_2^4 \\ z_2^8 \\ z_2^{16} \\ z_2^{32} \\ z_2^{64} \\ z_2^{128} \\ z_3 \\ z_3^2 \\ z_3^4 \\ z_3^8 \\ z_3^{16} \\ z_3^{32} \\ z_3^{64} \\ z_3^{128} \end{pmatrix} = \begin{pmatrix} C_B^{(0)} \cdot y_0 \\ C_B^{(1)} \cdot y_0^2 \\ C_B^{(2)} \cdot y_0^4 \\ C_B^{(3)} \cdot y_0^8 \\ C_B^{(4)} \cdot y_0^{16} \\ C_B^{(5)} \cdot y_0^{32} \\ C_B^{(6)} \cdot y_0^{64} \\ C_B^{(7)} \cdot y_0^{128} \\ C_B^{(0)} \cdot y_1 \\ C_B^{(1)} \cdot y_1^2 \\ C_B^{(2)} \cdot y_1^4 \\ C_B^{(3)} \cdot y_1^8 \\ C_B^{(4)} \cdot y_1^{16} \\ C_B^{(5)} \cdot y_1^{32} \\ C_B^{(6)} \cdot y_1^{64} \\ C_B^{(7)} \cdot y_1^{128} \\ C_B^{(0)} \cdot y_2 \\ C_B^{(1)} \cdot y_2^2 \\ C_B^{(2)} \cdot y_2^4 \\ C_B^{(3)} \cdot y_2^8 \\ C_B^{(4)} \cdot y_2^{16} \\ C_B^{(5)} \cdot y_2^{32} \\ C_B^{(6)} \cdot y_2^{64} \\ C_B^{(7)} \cdot y_2^{128} \\ C_B^{(0)} \cdot y_3 \\ C_B^{(1)} \cdot y_3^2 \\ C_B^{(2)} \cdot y_3^4 \\ C_B^{(3)} \cdot y_3^8 \\ C_B^{(4)} \cdot y_3^{16} \\ C_B^{(5)} \cdot y_3^{32} \\ C_B^{(6)} \cdot y_3^{64} \\ C_B^{(7)} \cdot y_3^{128} \end{pmatrix}$$

$$= \left(\begin{array}{cccc} \begin{matrix} C_B^{(0)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C_B^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_B^{(2)} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_B^{(3)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C_B^{(4)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C_B^{(5)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_B^{(6)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C_B^{(7)} \end{matrix} & \cdots & 0 & \\ \vdots & \vdots & & \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & \begin{matrix} C_B^{(0)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & C_B^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_B^{(2)} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_B^{(3)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C_B^{(4)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C_B^{(5)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_B^{(6)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C_B^{(7)} \end{matrix} \end{array}\right) \begin{pmatrix} y_0 \\ y_0^2 \\ y_0^4 \\ y_0^8 \\ y_0^{16} \\ y_0^{32} \\ y_0^{64} \\ y_0^{128} \\ \vdots \\ \vdots \\ y_3 \\ y_3^2 \\ y_3^4 \\ y_3^8 \\ y_3^{16} \\ y_3^{32} \\ y_3^{64} \\ y_3^{128} \end{pmatrix},$$

where $y_i, y_i^2, y_i^4, y_i^8, y_i^{16}, y_i^{32}, y_i^{64}, y_i^{128}$ are conjugates of y_i, for $0 \leq i \leq 3$.

The above is the basis we use to define Mix_B. For example, for the first column $y_0 = (a_{00}, a_{10}, a_{20}, a_{30})$ of the state vector a in AES, the MixColumn operation on the vector conjugate set of y_0 in BES is described as follows:

$$\begin{pmatrix}
02 & 03 & 01 & 01 & & & & & & & & & & & \\
01 & 02 & 03 & 01 & \cdots & & & & & \cdots & 0 & & & & \\
01 & 01 & 02 & 03 & & & & & & & & & & & \\
03 & 01 & 01 & 02 & & & & & & & & & & & \\
 & & & & & (02)^2 & (03)^2 & 1 & 1 & & & & & & \\
 & & & & \vdots & 1 & (02)^2 & (03)^2 & 1 & & \vdots & & & & \\
 & & & & & 1 & 1 & (02)^2 & (03)^2 & & & & & & \\
 & & & & & (03)^2 & 1 & 1 & (02)^2 & & & & & & \\
 & & & & \vdots & & & & & \ddots & \vdots & & & & \\
 & & & & & & & & & & & (02)^{128} & (03)^{128} & 1 & 1 \\
 & & & & 0 & \cdots & & & & \cdots & & 1 & (02)^{128} & (03)^{128} & 1 \\
 & & & & & & & & & & & 1 & 1 & (02)^{128} & (03)^{128} \\
 & & & & & & & & & & & (03)^{128} & 1 & 1 & (02)^{128}
\end{pmatrix}
\begin{pmatrix}
a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{00}^2 \\ a_{10}^2 \\ a_{20}^2 \\ a_{30}^2 \\ \vdots \\ a_{00}^{128} \\ a_{10}^{128} \\ a_{20}^{128} \\ a_{30}^{128}
\end{pmatrix}$$

$$
= \begin{pmatrix}
02 \cdot a_{00} + 03 \cdot a_{10} + a_{20} + a_{30} \\
a_{00} + 02 \cdot a_{10} + 03 \cdot a_{20} + a_{30} \\
a_{00} + a_{10} + 02 \cdot a_{20} + 03 \cdot a_{30} \\
03 \cdot a_{00} + a_{10} + a_{20} + 02 \cdot a_{30} \\
(02)^2 \cdot a_{00}^2 + (03)^2 \cdot a_{10}^2 + a_{20}^2 + a_{30}^2 \\
a_{00}^2 + (02)^2 \cdot a_{10}^2 + (03)^2 \cdot a_{20}^2 + a_{30}^2 \\
a_{00}^2 + a_{10}^2 + (02)^2 \cdot a_{20}^2 + (03)^2 \cdot a_{30}^2 \\
(03)^2 \cdot a_{00}^2 + a_{10}^2 + a_{20}^2 + (02)^2 \cdot a_{30}^2 \\
(02)^4 \cdot a_{00}^4 + (03)^4 \cdot a_{10}^4 + a_{20}^4 + a_{30}^4 \\
a_{00}^4 + (02)^4 \cdot a_{10}^4 + (03)^4 \cdot a_{20}^4 + a_{30}^4 \\
a_{00}^4 + a_{10}^4 + (02)^4 \cdot a_{20}^4 + (03)^4 \cdot a_{30}^4 \\
(03)^4 \cdot a_{00}^4 + a_{10}^4 + a_{20}^4 + (02)^4 \cdot a_{30}^4 \\
(02)^8 \cdot a_{00}^8 + (03)^8 \cdot a_{10}^8 + a_{20}^8 + a_{30}^8 \\
a_{00}^8 + (02)^8 \cdot a_{10}^8 + (03)^8 \cdot a_{20}^8 + a_{30}^8 \\
a_{00}^8 + a_{10}^8 + (02)^8 \cdot a_{20}^8 + (03)^8 \cdot a_{30}^8 \\
(03)^8 \cdot a_{00}^8 + a_{10}^8 + a_{20}^8 + (02)^8 \cdot a_{30}^8 \\
(02)^{16} \cdot a_{00}^{16} + (03)^{16} \cdot a_{10}^{16} + a_{20}^{16} + a_{30}^{16} \\
a_{00}^{16} + (02)^{16} \cdot a_{10}^{16} + (03)^{16} \cdot a_{20}^{16} + a_{30}^{16} \\
a_{00}^{16} + a_{10}^{16} + (02)^{16} \cdot a_{20}^{16} + (03)^{16} \cdot a_{30}^{16} \\
(03)^{16} \cdot a_{00}^{16} + a_{10}^{16} + a_{20}^{16} + (02)^{16} \cdot a_{30}^{16} \\
(02)^{32} \cdot a_{00}^{32} + (03)^{32} \cdot a_{10}^{32} + a_{20}^{32} + a_{30}^{32} \\
a_{00}^{32} + (02)^{32} \cdot a_{10}^{32} + (03)^{32} \cdot a_{20}^{32} + a_{30}^{32} \\
a_{00}^{32} + a_{10}^{32} + (02)^{32} \cdot a_{20}^{32} + (03)^{32} \cdot a_{30}^{32} \\
(03)^{32} \cdot a_{00}^{32} + a_{10}^{32} + a_{20}^{32} + (02)^{32} \cdot a_{30}^{32} \\
(02)^{64} \cdot a_{00}^{64} + (03)^{64} \cdot a_{10}^{64} + a_{20}^{64} + a_{30}^{64} \\
a_{00}^{64} + (02)^{64} \cdot a_{10}^{64} + (03)^{64} \cdot a_{20}^{64} + a_{30}^{64} \\
a_{00}^{64} + a_{10}^{64} + (02)^{64} \cdot a_{20}^{64} + (03)^{64} \cdot a_{30}^{64} \\
(03)^{64} \cdot a_{00}^{64} + a_{10}^{64} + a_{20}^{64} + (02)^{64} \cdot a_{30}^{64} \\
(02)^{128} \cdot a_{00}^{128} + (03)^{128} \cdot a_{10}^{128} + a_{20}^{128} + a_{30}^{128} \\
a_{00}^{128} + (02)^{128} \cdot a_{10}^{128} + (03)^{128} \cdot a_{20}^{128} + a_{30}^{128} \\
a_{00}^{128} + a_{10}^{128} + (02)^{128} \cdot a_{20}^{128} + (03)^{128} \cdot a_{30}^{128} \\
(03)^{128} \cdot a_{00}^{128} + a_{10}^{128} + a_{20}^{128} + (02)^{128} \cdot a_{30}^{128}
\end{pmatrix}
$$

Because the MixColumn step in BES respects the vector conjugate mapping, the output vector conjugate set of $C_B^{(k)} \cdot y_0^{2^k}$ is derived from the above column vector by some reordering, as follows:

$$\begin{pmatrix}
02 \cdot a_{00} + 03 \cdot a_{10} + a_{20} + a_{30} \\
(02)^2 \cdot a_{00}^2 + (03)^2 \cdot a_{10}^2 + a_{20}^2 + a_{30}^2 \\
(02)^4 \cdot a_{00}^4 + (03)^4 \cdot a_{10}^4 + a_{20}^4 + a_{30}^4 \\
(02)^8 \cdot a_{00}^8 + (03)^8 \cdot a_{10}^8 + a_{20}^8 + a_{30}^8 \\
(02)^{16} \cdot a_{00}^{16} + (03)^{16} \cdot a_{10}^{16} + a_{20}^{16} + a_{30}^{16} \\
(02)^{32} \cdot a_{00}^{32} + (03)^{32} \cdot a_{10}^{32} + a_{20}^{32} + a_{30}^{32} \\
(02)^{64} \cdot a_{00}^{64} + (03)^{64} \cdot a_{10}^{64} + a_{20}^{64} + a_{30}^{64} \\
(02)^{128} \cdot a_{00}^{128} + (03)^{128} \cdot a_{10}^{128} + a_{20}^{128} + a_{30}^{128} \\
a_{00} + 02 \cdot a_{10} + 03 \cdot a_{20} + a_{30} \\
a_{00}^2 + (02)^2 \cdot a_{10}^2 + (03)^2 \cdot a_{20}^2 + a_{30}^2 \\
a_{00}^4 + (02)^4 \cdot a_{10}^4 + (03)^4 \cdot a_{20}^4 + a_{30}^4 \\
a_{00}^8 + (02)^8 \cdot a_{10}^8 + (03)^8 \cdot a_{20}^8 + a_{30}^8 \\
a_{00}^{16} + (02)^{16} \cdot a_{10}^{16} + (03)^{16} \cdot a_{20}^{16} + a_{30}^{16} \\
a_{00}^{32} + (02)^{32} \cdot a_{10}^{32} + (03)^{32} \cdot a_{20}^{32} + a_{30}^{32} \\
a_{00}^{64} + (02)^{64} \cdot a_{10}^{64} + (03)^{64} \cdot a_{20}^{64} + a_{30}^{64} \\
a_{00}^{128} + (02)^{128} \cdot a_{10}^{128} + (03)^{128} \cdot a_{20}^{128} + a_{30}^{128} \\
a_{00} + a_{10} + 02 \cdot a_{20} + 03 \cdot a_{30} \\
a_{00}^2 + a_{10}^2 + (02)^2 \cdot a_{20}^2 + (03)^2 \cdot a_{30}^2 \\
a_{00}^4 + a_{10}^4 + (02)^4 \cdot a_{20}^4 + (03)^4 \cdot a_{30}^4 \\
a_{00}^8 + a_{10}^8 + (02)^8 \cdot a_{20}^8 + (03)^8 \cdot a_{30}^8 \\
a_{00}^{16} + a_{10}^{16} + (02)^{16} \cdot a_{20}^{16} + (03)^{16} \cdot a_{30}^{16} \\
a_{00}^{32} + a_{10}^{32} + (02)^{32} \cdot a_{20}^{32} + (03)^{32} \cdot a_{30}^{32} \\
a_{00}^{64} + a_{10}^{64} + (02)^{64} \cdot a_{20}^{64} + (03)^{64} \cdot a_{30}^{64} \\
a_{00}^{128} + a_{10}^{128} + (02)^{128} \cdot a_{20}^{128} + (03)^{128} \cdot a_{30}^{128} \\
03 \cdot a_{00} + a_{10} + a_{20} + 02 \cdot a_{30} \\
(03)^2 \cdot a_{00}^2 + a_{10}^2 + a_{20}^2 + (02)^2 \cdot a_{30}^2 \\
(03)^4 \cdot a_{00}^4 + a_{10}^4 + a_{20}^4 + (02)^4 \cdot a_{30}^4 \\
(03)^8 \cdot a_{00}^8 + a_{10}^8 + a_{20}^8 + (02)^8 \cdot a_{30}^8 \\
(03)^{16} \cdot a_{00}^{16} + a_{10}^{16} + a_{20}^{16} + (02)^{16} \cdot a_{30}^{16} \\
(03)^{32} \cdot a_{00}^{32} + a_{10}^{32} + a_{20}^{32} + (02)^{32} \cdot a_{30}^{32} \\
(03)^{64} \cdot a_{00}^{64} + a_{10}^{64} + a_{20}^{64} + (02)^{64} \cdot a_{30}^{64} \\
(03)^{128} \cdot a_{00}^{128} + a_{10}^{128} + a_{20}^{128} + (02)^{128} \cdot a_{30}^{128}
\end{pmatrix}$$

Accordingly, the matrix multiplication of the MixColumn on the vector conjugate set of y_0 can be represented by $Mix_B^{(y_0)} =$

$$= \begin{pmatrix}
02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (02)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (02)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (02)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (02)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (02)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (02)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 \\
01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 \\
01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๔} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๘} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๖} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๓๒} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๖๔} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๒๘} \\
03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (03)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๒} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (03)^{๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๔} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (03)^{๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๘} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (03)^{๑๖} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๖} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (03)^{๓๒} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๓๒} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (03)^{๖๔} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๖๔} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (03)^{๑๒๘} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (02)^{๑๒๘}
\end{pmatrix}
\begin{pmatrix}
a_{๐} \\ a_{๐}^{๒} \\ a_{๐}^{๔} \\ a_{๐}^{๘} \\ a_{๐}^{๑๖} \\ a_{๐}^{๓๒} \\ a_{๐}^{๖๔} \\ a_{๐}^{๑๒๘} \\
a_{๑} \\ a_{๑}^{๒} \\ a_{๑}^{๔} \\ a_{๑}^{๘} \\ a_{๑}^{๑๖} \\ a_{๑}^{๓๒} \\ a_{๑}^{๖๔} \\ a_{๑}^{๑๒๘} \\
a_{๒} \\ a_{๒}^{๒} \\ a_{๒}^{๔} \\ a_{๒}^{๘} \\ a_{๒}^{๑๖} \\ a_{๒}^{๓๒} \\ a_{๒}^{๖๔} \\ a_{๒}^{๑๒๘} \\
a_{๓} \\ a_{๓}^{๒} \\ a_{๓}^{๔} \\ a_{๓}^{๘} \\ a_{๓}^{๑๖} \\ a_{๓}^{๓๒} \\ a_{๓}^{๖๔} \\ a_{๓}^{๑๒๘}
\end{pmatrix}$$

Apparently, the same deduction is true for the vector conjugate set of column y_1, y_2, y_3. Furthermore, $Mix_B^{y_1}, Mix_B^{y_2}, Mix_B^{y_3}$ are identical to $Mix_B^{y_0}$.

Therefore, the entire matrix multiplication representation of the MixColumn on the state vector $b \in B$ is given by:

$$Mix_B \cdot b = Diag_4(Mix_B^{(y_0)}) \cdot b$$

4.2.5 AddRoundKey Operation

While in the AES, this steps modifies the state vector $a \in A$ by combining a round key $(k_A)_i \in A$ with the bitwise EXOR operation: $a \rightarrow a + (k_A)_i$, in the BES the similar bitwise EXOR operation with a round key $(k_B)_i \in B$ has applied to the state vector $b \in B$: $b \rightarrow b + (k_B)_i$, where $0 \leq i \leq N_{round} = 10$.

Furthermore, a constant $63 \in A$ addition on each byte of the state vector $a \in A$, which is removed from ByteSub operation, is conceptually combined into this step in AES. Thus, in BES, the similar operation is realized by a bitwise EXOR operation on the state vector $b \in B$ with an extended 128-byte constant column vector as follows:

$$\begin{pmatrix} 63, & (63)^2, & (63)^4, & (63)^8, & (63)^{16}, & (63)^{32}, & (63)^{64}, & (63)^{128}, & \cdots, & 63, & 63^2, & \cdots, & (63)^{128} \end{pmatrix}^T$$

In next section, I further consider how the round keys are generated.

4.3 Key Schedule

In AES, round keys are also elements of the state space, so are represented in the same way as the state vector. Here a round key is considered as a column vector:

$$k = \begin{pmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{pmatrix}$$

$$= \begin{pmatrix} k_{00}, & k_{10}, & k_{20}, & k_{30}, & k_{01}, & \cdots, & k_{31}, & \cdots, & k_{03}, & k_{13}, & k_{23}, & k_{33} \end{pmatrix}^T$$

Accordingly, by applying the vector conjugate embedding mapping ϕ, we can extend a 16-byte round key k_A to a 128-byte round key k_B.

$$k_A = \begin{pmatrix} k_{00}, & \cdots, & k_{30}, & k_{01}, & \cdots, & k_{03}, & \cdots & k_{33} \end{pmatrix}^T$$

$$\xrightarrow{\phi} k_B = \begin{pmatrix} \phi(k_{00}), & \cdots, & \phi(k_{30}), & \phi(k_{01}), & \cdots, & \phi(k_{03}), & \cdots & \phi(k_{33}) \end{pmatrix}^T$$

$$= \begin{pmatrix} k_{00}, & k_{00}^2, & k_{00}^4, & k_{00}^8, & k_{00}^{16}, & k_{00}^{32}, & k_{00}^{64}, & k_{00}^{128}, & \cdots, & k_{33}, & \cdots, & k_{33}^{128} \end{pmatrix}^T$$

In order to get 11 round keys for the 10-round encryption, AES specifies a key schedule to extend the original cipher key. First, the first round key is filled with the cipher key. The following round keys are defined recursively by using previously defined sub key.

For the i^{th} round of AES , the round key is given by $(k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3})$. There are two versions of the recursion function.

1. For $n = 4i$ is a multiple of 4, $k_n = k_{n-4} + f(k_{n-1})$. The non-linear function f includes

a S-box substitution, a byte rotation and a round constant RC addition.

$$k_A(i) = \begin{pmatrix} k_{00} \\ k_{10} \\ k_{20} \\ k_{30} \\ k_{01} \\ k_{11} \\ k_{21} \\ k_{31} \\ k_{02} \\ k_{12} \\ k_{22} \\ k_{32} \\ k_{03} \\ k_{13} \\ k_{23} \\ k_{33} \end{pmatrix} \to k_A(i+1) = \begin{pmatrix} k_{00} + S_{RD}(k_{13}) + RC \\ k_{10} + S_{RD}(k_{23}) \\ k_{20} + S_{RD}(k_{33}) \\ k_{30} + S_{RD}(k_{03}) \\ k_{01} + S_{RD}(k_{10} + S_{RD}(k_{23})) + RC \\ k_{11} + S_{RD}(k_{20} + S_{RD}(k_{33})) \\ k_{21} + S_{RD}(k_{30} + S_{RD}(k_{03})) \\ k_{31} + S_{RD}(k_{00} + S_{RD}(k_{13}) + RC) \\ k_{02} + S_{RD}(k_{11} + S_{RD}(k_{20} + S_{RD}(k_{33}))) + RC \\ k_{12} + S_{RD}(k_{21} + S_{RD}(k_{30} + S_{RD}(k_{03}))) \\ k_{22} + S_{RD}(k_{31} + S_{RD}(k_{00} + S_{RD}(k_{13}) + RC)) \\ k_{32} + S_{RD}(k_{01} + S_{RD}(k_{10} + S_{RD}(k_{23})) + RC) \\ k_{03} + S_{RD}(k_{12} + S_{RD}(k_{21} + S_{RD}(k_{30} + S_{RD}(k_{03})))) + RC \\ k_{13} + S_{RD}(k_{22} + S_{RD}(k_{31} + S_{RD}(k_{00} + S_{RD}(k_{13}) + RC))) \\ k_{23} + S_{RD}(k_{32} + S_{RD}(k_{01} + S_{RD}(k_{10} + S_{RD}(k_{23})) + RC)) \\ k_{33} + S_{RD}(k_{02} + S_{RD}(k_{11} + S_{RD}(k_{20} + S_{RD}(k_{33}))) + RC) \end{pmatrix}$$

2. For $n = 4i+1, 4i+2, 4i+3$ is not a multiple of 4, $k_n = k_{n-4} + k_{n-1}$.

$$k_A(i) = \begin{pmatrix} k_{00} \\ k_{10} \\ k_{20} \\ k_{30} \\ k_{01} \\ k_{11} \\ k_{21} \\ k_{31} \\ k_{02} \\ k_{12} \\ k_{22} \\ k_{32} \\ k_{03} \\ k_{13} \\ k_{23} \\ k_{33} \end{pmatrix} \to k_A(i+1) = \begin{pmatrix} k_{00}+k_{03} \\ k_{10}+k_{13} \\ k_{20}+k_{23} \\ k_{30}+k_{33} \\ k_{01}+(k_{00}+k_{03}) \\ k_{11}+(k_{10}+k_{13}) \\ k_{21}+(k_{20}+k_{23}) \\ k_{31}+(k_{30}+k_{33}) \\ k_{02}+(k_{01}+(k_{00}+k_{03})) \\ k_{12}+(k_{11}+(k_{10}+k_{13})) \\ k_{22}+(k_{21}+(k_{20}+k_{23})) \\ k_{32}+(k_{31}+(k_{30}+k_{33})) \\ k_{03}+(k_{02}+(k_{01}+(k_{00}+k_{03}))) \\ k_{13}+(k_{12}+(k_{11}+(k_{10}+k_{13}))) \\ k_{23}+(k_{22}+(k_{21}+(k_{20}+k_{23}))) \\ k_{33}+(k_{32}+(k_{31}+(k_{30}+k_{33}))) \end{pmatrix}$$

Unlike the description of BES key schedule in previous chapter based on the BES specification [7], here in my approach, I'm not using the same techniques from the previous sections to describe and implement the key schedule, because of the following two considerations.

First of all, from the matrix representation of $k(i+1)$ for $n = 4i$, we can see the recursion function brings a lot of complexity to translate the transformation into just a simple matrix multiplication of the state vector. Second of all, the key schedule is relatively independent of the round function of the cipher. The way of the generations of round keys will not reflect on the algebraic representation of the BES round function.

Therefore, in my implementation, the BES ExpandedKey array is directly derived from the AES ExpandedKey array by using the vector conjugate mapping.

Chapter 5

The Implementation of BES Decryption

In the BES specification [7], the inverse BES cipher is not described. The following chapter is an original contribution and further explores the BES decryption process with full respect to the design concepts of BES.

5.1 Inverse Round Function

As previously described , the inverse round function of BES is similar to that of AES:

```
InvRound(State,RoundKey)
{
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
AddRoundKey(State, RoundKey);
}
```

5.2 InvByteSub Operation

In AES, the InvByteSub applies the inverse S-Box denoted by S_{RD}^{-1} to the state vector $a \in A$. It consists of three transformation: the S-Box inversion denoted by g, the inverse GF(2)-linear operation denoted by $f^{-1}$and a constant $'05'$ addition. Since g is self-inverse, we have

$$S_{RD}^{-1}(a) = g^{-1}(f^{-1}(a) + 05) = g(f^{-1}(a) + 05)$$

1. Inverse S-box GF (2)-linear Operation

   In inverse AES cipher, the inverse GF(2)-linear function $f^{-1} : F \rightarrow F$, operating on each byte of the state vector $a \in A$, is represented by a matrix multiplication of each byte with a (8×8) F matrix L_A^{-1}, wherein each byte $a_{ij} \in F$ consisting of bits $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$ is considered as a GF (2) vector of 8 dimensions.

$$f^{-1}(a_{ij}) = L_A^{-1} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

   The tabular representation of f^{-1} is given below.

		n															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
m	0	00	4a	94	de	29	63	bd	f7	52	18	c6	8c	7b	31	ef	a5
	1	a4	ee	30	7a	8d	c7	19	53	f6	bc	62	28	df	95	4b	01
	2	49	03	dd	97	60	2a	f4	be	1b	51	8f	c5	32	78	a6	ec
	3	ed	a7	79	33	c4	8e	50	1a	bf	f5	2b	61	96	dc	02	48
	4	92	d8	06	4c	bb	f1	2f	65	c0	8a	54	1e	e9	a3	7d	37
	5	36	7c	a2	e8	1f	55	8b	c1	64	2e	f0	ba	4d	07	d9	93
	6	db	91	4f	05	f2	b8	66	2c	89	c3	1d	57	a0	ea	34	7e
	7	7f	35	eb	a1	56	1c	c2	88	2d	67	b9	f3	04	4e	90	da
	8	25	6f	b1	fb	0c	46	98	d2	77	3d	e3	a9	5e	14	ca	80
	9	81	cb	15	5f	a8	e2	3c	76	d3	99	47	0d	fa	b0	6e	24
	a	6c	26	f8	b2	45	0f	d1	9b	3e	74	aa	e0	17	5d	83	c9
	b	c8	82	5c	16	e1	ab	75	3f	9a	d0	0e	44	b3	f9	27	6d
	c	b7	fd	23	69	9e	d4	0a	40	e5	af	71	3b	cc	86	58	12
	d	13	59	87	cd	3a	70	ae	e4	41	0b	d5	9f	68	22	fc	b6
	e	fe	b4	6a	20	d7	9d	43	09	ac	e6	38	72	85	cf	11	5b
	f	5a	10	ce	84	73	39	e7	ad	08	42	9c	d6	21	6b	b5	ff

Table 3: Tabular Representation of $f^{-1}(mn)$

The Lagrange interpolating polynomial of f^{-1} is the polynomial of degree 255 that passes through the 256 points $y_0 = f^{-1}(x_{00}), y_1 = f^{-1}(x_{01}), \cdots, y_{ff} = f^{-1}(x_{ff})$, given by:

$$f^{-1}(x) = \sum_{j=0}^{ff} (y_j \prod_{\substack{k=0 \\ k \neq j}}^{ff} \frac{x - x_k}{x_j - x_k})$$

Written explicitly,

$$f^{-1}(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{ff})}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_{ff})} \cdot y_0 + \frac{(x-x_0)(x-x_2)\cdots(x-x_{ff})}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_{ff})} \cdot y_1$$

$$+ \cdots + \frac{(x-x_0)(x-x_1)\cdots(x-x_{fe})}{(x_{ff}-x_0)(x_{ff}-x_1)\cdots(x_{ff}-x_{fe})} \cdot y_{ff}$$

Substituting the values in Table 3 for x_i,

$$= \frac{(x-1)(x-2)\cdots(x-ff)}{(0-1)(0-2)\cdots(0-ff)} \cdot 0 + \frac{(x-0)(x-2)\cdots(x-ff)}{(1-0)(1-2)\cdots(1-ff)} \cdot 4a$$

$$+ \cdots + \frac{(x-0)(x-1)\cdots(x-fe)}{(ff-0)(ff-1)\cdots(ff-fe)} \cdot ff$$

The coefficients in the polynomial are considered as GF (2^8) vectors, thus the addition and multiplication of two coefficients are the addition and modular multiplication over F. Accordingly, the division can be represented by the multiplicative inverse a^{-1} over F . Therefore, f^{-1} can be interpolated by Lagrange formula and represented by a polynomial function as follows:

$$f^{-1}(a_{ij}) = \sum_{k=0}^{7} \gamma_k (a_{ij})^{2^k} \quad for \quad a \in F \quad and \quad 0 \leq i,j \leq 3$$

$$where(\gamma_0,\gamma_1,\gamma_2,\gamma_3,\gamma_4,\gamma_5,\gamma_6,\gamma_7) = (05,fe,7f,5a,78,59,db,6e).$$

Now we can define the GF(2)-linear operation in the inverse BES by a matrix multiplication on 8-conjugate vectors with the following (8×8) matrix L_B^{-1}:

$$\begin{pmatrix} 05 & fe & 7f & 5a & 78 & 59 & db & 6e \\ (6e)^2 & (05)^2 & (fe)^2 & (7f)^2 & (5a)^2 & (78)^2 & (59)^2 & (db)^2 \\ (db)^4 & (6e)^4 & (05)^4 & (fe)^4 & (7f)^4 & (5a)^4 & (78)^4 & (59)^4 \\ (59)^8 & (db)^8 & (6e)^8 & (05)^8 & (fe)^8 & (7f)^8 & (5a)^8 & (78)^8 \\ (78)^{16} & (59)^{16} & (db)^{16} & (6e)^{16} & (05)^{16} & (fe)^{16} & (7f)^{16} & (5a)^{16} \\ (5a)^{32} & (78)^{32} & (59)^{32} & (db)^{32} & (6e)^{32} & (05)^{32} & (fe)^{32} & (7f)^{32} \\ (7f)^{64} & (5a)^{64} & (78)^{64} & (59)^{64} & (db)^{64} & (6e)^{64} & (05)^{64} & (fe)^{64} \\ (fe)^{128} & (7f)^{128} & (5a)^{128} & (78)^{128} & (59)^{128} & (db)^{128} & (6e)^{128} & (05)^{128} \end{pmatrix} \begin{pmatrix} a_{ij} \\ a_{ij}^2 \\ a_{ij}^4 \\ a_{ij}^8 \\ a_{ij}^{16} \\ a_{ij}^{32} \\ a_{ij}^{64} \\ a_{ij}^{128} \end{pmatrix}$$

$$= \begin{pmatrix} 05\cdot a_{ij}+fe\cdot a_{ij}^{2}+7f\cdot a_{ij}^{4}+5a\cdot a_{ij}^{8}+78\cdot a_{ij}^{16}+59\cdot a_{ij}^{32}+db\cdot a_{ij}^{64}+6e\cdot a_{ij}^{128} \\ (6e)^2\cdot a_{ij}+(05)^2\cdot a_{ij}^{2}+(fe)^2\cdot a_{ij}^{4}+(7f)^2\cdot a_{ij}^{8}+(5a)^2\cdot a_{ij}^{16}+(78)^2\cdot a_{ij}^{32}+(59)^2\cdot a_{ij}^{64}+(db)^2\cdot a_{ij}^{128} \\ (db)^4\cdot a_{ij}+(6e)^4\cdot a_{ij}^{2}+(05)^4\cdot a_{ij}^{4}+(fe)^4\cdot a_{ij}^{8}+(7f)^4\cdot a_{ij}^{16}+(5a)^4\cdot a_{ij}^{32}+(78)^4\cdot a_{ij}^{64}+(59)^4\cdot a_{ij}^{128} \\ (59)^8\cdot a_{ij}+(db)^8\cdot a_{ij}^{2}+(6e)^8\cdot a_{ij}^{4}+(05)^8\cdot a_{ij}^{8}+(fe)^8\cdot a_{ij}^{16}+(7f)^8\cdot a_{ij}^{32}+(5a)^8\cdot a_{ij}^{64}+(78)^8\cdot a_{ij}^{128} \\ (78)^{16}\cdot a_{ij}+(59)^{16}\cdot a_{ij}^{2}+(db)^{16}\cdot a_{ij}^{4}+(6e)^{16}\cdot a_{ij}^{8}+(05)^{16}\cdot a_{ij}^{16}+(fe)^{16}\cdot a_{ij}^{32}+(7f)^{16}\cdot a_{ij}^{64}+(5a)^{16}\cdot a_{ij}^{128} \\ (5a)^{32}\cdot a_{ij}+(78)^{32}\cdot a_{ij}^{2}+(59)^{32}\cdot a_{ij}^{4}+(db)^{32}\cdot a_{ij}^{8}+(6e)^{32}\cdot a_{ij}^{16}+(05)^{32}\cdot a_{ij}^{32}+(fe)^{32}\cdot a_{ij}^{64}+(7f)^{32}\cdot a_{ij}^{128} \\ (7f)^{64}\cdot a_{ij}+(5a)^{64}\cdot a_{ij}^{2}+(78)^{64}\cdot a_{ij}^{4}+(59)^{64}\cdot a_{ij}^{8}+(db)^{64}\cdot a_{ij}^{16}+(6e)^{64}\cdot a_{ij}^{32}+(05)^{64}\cdot a_{ij}^{64}+(fe)^{64}\cdot a_{ij}^{128} \\ (fe)^{128}\cdot a_{ij}+(7f)^{128}\cdot a_{ij}^{2}+(5a)^{128}\cdot a_{ij}^{4}+(78)^{128}\cdot a_{ij}^{8}+(59)^{128}\cdot a_{ij}^{16}+(db)^{128}\cdot a_{ij}^{32}+(6e)^{128}\cdot a_{ij}^{64}+(05)^{128}\cdot a_{ij}^{128} \end{pmatrix}$$

Therefore, L_B^{-1} applies the AES-action of the inverse GF(2)-linear map on the first byte of a 8-conjugate vector, and preserves the conjugacy property on the remaining bytes. By using L_B^{-1} in a similar way, we can define a (128×128) F-matrix Lin_B^{-1} to represent the entire inverse GF(2)-linear operation on BES.

2. S-box Inversion

As described before, this step is self-inverse, for the state vector $b \in B$, we have

$$b = \phi(a_{ij}) = (a_{ij}, a_{ij}^{2}, a_{ij}^{4}, a_{ij}^{8}, a_{ij}^{16}, a_{ij}^{32}, a_{ij}^{64}, a_{ij}^{128})$$

$$\longrightarrow b^{(-1)} = (\phi(a_{ij}))^{(-1)} = (a_{ij}, a_{ij}^{2}, a_{ij}^{4}, a_{ij}^{8}, a_{ij}^{16}, a_{ij}^{32}, a_{ij}^{64}, a_{ij}^{128})^{(-1)}$$

$$= (a_{ij}^{(-1)}, (a_{ij}^{(-1)})^{2}, (a_{ij}^{(-1)})^{4}, (a_{ij}^{(-1)})^{8}, (a_{ij}^{(-1)})^{16}, (a_{ij}^{(-1)})^{32}, (a_{ij}^{(-1)})^{64}, (a_{ij}^{(-1)})^{128})$$

$$= \phi(a_{ij}^{-1}) \quad for \quad 0 \le i, j \le 3$$

Now we can have the representation of inverse S-Box substitution:

$$S_{RD}^{-1}(b) = (Lin_B^{-1} \cdot (b) + 05)^{-1}$$

$$
= \left(\left(\begin{array}{ccccccccccccccccc}
05 & fe & 7f & 5a & 78 & 59 & db & 6e & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(6e)^{2} & (05)^{2} & (fe)^{2} & (7f)^{2} & (5a)^{2} & (78)^{2} & (59)^{2} & (db)^{2} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(db)^{4} & (6e)^{4} & (05)^{4} & (fe)^{4} & (7f)^{4} & (5a)^{4} & (78)^{4} & (59)^{4} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(59)^{8} & (db)^{8} & (6e)^{8} & (05)^{8} & (fe)^{8} & (7f)^{8} & (5a)^{8} & (78)^{8} & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(78)^{16} & (59)^{16} & (db)^{16} & (6e)^{16} & (05)^{16} & (fe)^{16} & (7f)^{16} & (5a)^{16} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(5a)^{32} & (78)^{32} & (59)^{32} & (db)^{32} & (6e)^{32} & (05)^{32} & (fe)^{32} & (7f)^{32} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(7f)^{64} & (5a)^{64} & (78)^{64} & (59)^{64} & (db)^{64} & (6e)^{64} & (05)^{64} & (fe)^{64} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(fe)^{128} & (7f)^{128} & (5a)^{128} & (78)^{128} & (59)^{128} & (db)^{128} & (6e)^{128} & (05)^{128} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & \vdots & & & & \ddots & & & & & \vdots & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 05 & fe & 7f & 5a & 78 & 59 & db & 6e \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (6e)^{2} & (05)^{2} & (fe)^{2} & (7f)^{2} & (5a)^{2} & (78)^{2} & (59)^{2} & (db)^{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (db)^{4} & (6e)^{4} & (05)^{4} & (fe)^{4} & (7f)^{4} & (5a)^{4} & (78)^{4} & (59)^{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & (59)^{8} & (db)^{8} & (6e)^{8} & (05)^{8} & (fe)^{8} & (7f)^{8} & (5a)^{8} & (78)^{8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (78)^{16} & (59)^{16} & (db)^{16} & (6e)^{16} & (05)^{16} & (fe)^{16} & (7f)^{16} & (5a)^{16} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (5a)^{32} & (78)^{32} & (59)^{32} & (db)^{32} & (6e)^{32} & (05)^{32} & (fe)^{32} & (7f)^{32} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (7f)^{64} & (5a)^{64} & (78)^{64} & (59)^{64} & (db)^{64} & (6e)^{64} & (05)^{64} & (fe)^{64} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & (fe)^{128} & (7f)^{128} & (5a)^{128} & (78)^{128} & (59)^{128} & (db)^{128} & (6e)^{128} & (05)^{128}
\end{array} \right)
\begin{pmatrix}
a_{00} \\ a_{00}^{2} \\ a_{00}^{4} \\ a_{00}^{8} \\ a_{00}^{16} \\ a_{00}^{32} \\ a_{00}^{64} \\ a_{00}^{128} \\ \vdots \\ a_{33} \\ a_{33}^{2} \\ a_{33}^{4} \\ a_{33}^{8} \\ a_{33}^{16} \\ a_{33}^{32} \\ a_{33}^{64} \\ a_{33}^{128}
\end{pmatrix}
+
\begin{pmatrix}
05 \\ (05)^{2} \\ (05)^{4} \\ (05)^{8} \\ (05)^{16} \\ (05)^{32} \\ (05)^{64} \\ (05)^{128} \\ \vdots \\ 05 \\ (05)^{2} \\ (05)^{4} \\ (05)^{8} \\ (05)^{16} \\ (05)^{32} \\ (05)^{64} \\ (05)^{128}
\end{pmatrix} \right)^{-1}
$$

$$= \begin{pmatrix} 05 \cdot a_{00} + fe \cdot a_{00}^{2} + 7f \cdot a_{00}^{4} + 5a \cdot a_{00}^{8} + 78 \cdot a_{00}^{16} + 59 \cdot a_{00}^{32} + db \cdot a_{00}^{64} + 6e \cdot a_{00}^{128} + 05 \\ (6e)^{2} \cdot a_{00} + (05)^{2} \cdot a_{00}^{2} + (fe)^{2} \cdot a_{00}^{4} + (7f)^{2} \cdot a_{00}^{8} + (5a)^{2} \cdot a_{00}^{16} + (78)^{2} \cdot a_{00}^{32} + (59)^{2} \cdot a_{00}^{64} + (db)^{2} \cdot a_{00}^{128} + (05)^{2} \\ (db)^{4} \cdot a_{00} + (6e)^{4} \cdot a_{00}^{2} + (05)^{4} \cdot a_{00}^{4} + (fe)^{4} \cdot a_{00}^{8} + (7f)^{4} \cdot a_{00}^{16} + (5a)^{4} \cdot a_{00}^{32} + (78)^{4} \cdot a_{00}^{64} + (59)^{4} \cdot a_{00}^{128} + (05)^{4} \\ (59)^{8} \cdot a_{00} + (db)^{8} \cdot a_{00}^{2} + (6e)^{8} \cdot a_{00}^{4} + (05)^{8} \cdot a_{00}^{8} + (fe)^{8} \cdot a_{00}^{16} + (7f)^{8} \cdot a_{00}^{32} + (5a)^{8} \cdot a_{00}^{64} + (78)^{8} \cdot a_{00}^{128} + (05)^{8} \\ (78)^{16} \cdot a_{00} + (59)^{16} \cdot a_{00}^{2} + (db)^{16} \cdot a_{00}^{4} + (6e)^{16} \cdot a_{00}^{8} + (05)^{16} \cdot a_{00}^{16} + (fe)^{16} \cdot a_{00}^{32} + (7f)^{16} \cdot a_{00}^{64} + (5a)^{16} \cdot a_{00}^{128} + (05)^{16} \\ (5a)^{32} \cdot a_{00} + (78)^{32} \cdot a_{00}^{2} + (59)^{32} \cdot a_{00}^{4} + (db)^{32} \cdot a_{00}^{8} + (6e)^{32} \cdot a_{00}^{16} + (05)^{32} \cdot a_{00}^{32} + (fe)^{32} \cdot a_{00}^{64} + (7f)^{32} \cdot a_{00}^{128} + (05)^{32} \\ (7f)^{64} \cdot a_{00} + (5a)^{64} \cdot a_{00}^{2} + (78)^{64} \cdot a_{00}^{4} + (59)^{64} \cdot a_{00}^{8} + (db)^{64} \cdot a_{00}^{16} + (6e)^{64} \cdot a_{00}^{32} + (05)^{64} \cdot a_{00}^{64} + (fe)^{64} \cdot a_{00}^{128} + (05)^{64} \\ (fe)^{128} \cdot a_{00} + (7f)^{128} \cdot a_{00}^{2} + (5a)^{128} \cdot a_{00}^{4} + (78)^{128} \cdot a_{00}^{8} + (59)^{128} \cdot a_{00}^{16} + (db)^{128} \cdot a_{00}^{32} + (6e)^{128} \cdot a_{00}^{64} + (05)^{128} \cdot a_{00}^{128} + (05)^{128} \\ \vdots \\ 05 \cdot a_{33} + fe \cdot a_{33}^{2} + 7f \cdot a_{33}^{4} + 5a \cdot a_{33}^{8} + 78 \cdot a_{33}^{16} + 59 \cdot a_{33}^{32} + db \cdot a_{33}^{64} + 6e \cdot a_{33}^{128} + 05 \\ (6e)^{2} \cdot a_{33} + (05)^{2} \cdot a_{33}^{2} + (fe)^{2} \cdot a_{33}^{4} + (7f)^{2} \cdot a_{33}^{8} + (5a)^{2} \cdot a_{33}^{16} + (78)^{2} \cdot a_{33}^{32} + (59)^{2} \cdot a_{33}^{64} + (db)^{2} \cdot a_{33}^{128} + (05)^{2} \\ (db)^{4} \cdot a_{33} + (6e)^{4} \cdot a_{33}^{2} + (05)^{4} \cdot a_{33}^{4} + (fe)^{4} \cdot a_{33}^{8} + (7f)^{4} \cdot a_{33}^{16} + (5a)^{4} \cdot a_{33}^{32} + (78)^{4} \cdot a_{33}^{64} + (59)^{4} \cdot a_{33}^{128} + (05)^{4} \\ (5a)^{8} \cdot a_{33} + (db)^{8} \cdot a_{33}^{2} + (6e)^{8} \cdot a_{33}^{4} + (05)^{8} \cdot a_{33}^{8} + (fe)^{8} \cdot a_{33}^{16} + (7f)^{8} \cdot a_{33}^{32} + (5a)^{8} \cdot a_{33}^{64} + (78)^{8} \cdot a_{33}^{128} + (05)^{8} \\ (78)^{16} \cdot a_{33} + (5a)^{16} \cdot a_{33}^{2} + (db)^{16} \cdot a_{33}^{4} + (6e)^{16} \cdot a_{33}^{8} + (05)^{16} \cdot a_{33}^{16} + (fe)^{16} \cdot a_{33}^{32} + (7f)^{16} \cdot a_{33}^{64} + (5a)^{16} \cdot a_{33}^{128} + (05)^{16} \\ (5a)^{32} \cdot a_{33} + (78)^{32} \cdot a_{33}^{2} + (59)^{32} \cdot a_{33}^{4} + (db)^{32} \cdot a_{33}^{8} + (6e)^{32} \cdot a_{33}^{16} + (05)^{32} \cdot a_{33}^{32} + (fe)^{32} \cdot a_{33}^{64} + (7f)^{32} \cdot a_{33}^{128} + (05)^{32} \\ (7f)^{64} \cdot a_{33} + (5a)^{64} \cdot a_{33}^{2} + (78)^{64} \cdot a_{33}^{4} + (59)^{64} \cdot a_{33}^{8} + (db)^{64} \cdot a_{33}^{16} + (6e)^{64} \cdot a_{33}^{32} + (05)^{64} \cdot a_{33}^{64} + (fe)^{64} \cdot a_{33}^{128} + (05)^{64} \\ (fe)^{128} \cdot a_{33} + (7f)^{128} \cdot a_{33}^{2} + (5a)^{128} \cdot a_{33}^{4} + (78)^{128} \cdot a_{33}^{8} + (59)^{128} \cdot a_{33}^{16} + (db)^{128} \cdot a_{33}^{32} + (6e)^{128} \cdot a_{33}^{64} + (05)^{128} \cdot a_{33}^{128} + (05)^{128} \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} (05\cdot a_{00} + fe\cdot a_{00}^{2} + 7f\cdot a_{00}^{4} + 5a\cdot a_{00}^{8} + 78\cdot a_{00}^{16} + 59\cdot a_{00}^{32} + db\cdot a_{00}^{64} + 6e\cdot a_{00}^{128} + 05)^{-1} \\ \left((6e)^{2}\cdot a_{00} + (05)^{2}\cdot a_{00}^{2} + (fe)^{2}\cdot a_{00}^{4} + (7f)^{2}\cdot a_{00}^{8} + (5a)^{2}\cdot a_{00}^{16} + (78)^{2}\cdot a_{00}^{32} + (59)^{2}\cdot a_{00}^{64} + (db)^{2}\cdot a_{00}^{128} + (05)^{2}\right)^{-1} \\ \left((db)^{4}\cdot a_{00} + (6e)^{4}\cdot a_{00}^{2} + (05)^{4}\cdot a_{00}^{4} + (fe)^{4}\cdot a_{00}^{8} + (7f)^{4}\cdot a_{00}^{16} + (5a)^{4}\cdot a_{00}^{32} + (78)^{4}\cdot a_{00}^{64} + (59)^{4}\cdot a_{00}^{128} + (05)^{4}\right)^{-1} \\ \left((59)^{8}\cdot a_{00} + (db)^{8}\cdot a_{00}^{2} + (6e)^{8}\cdot a_{00}^{4} + (05)^{8}\cdot a_{00}^{8} + (fe)^{8}\cdot a_{00}^{16} + (7f)^{8}\cdot a_{00}^{32} + (5a)^{8}\cdot a_{00}^{64} + (78)^{8}\cdot a_{00}^{128} + (05)^{8}\right)^{-1} \\ \left((78)^{16}\cdot a_{00} + (59)^{16}\cdot a_{00}^{2} + (db)^{16}\cdot a_{00}^{4} + (6e)^{16}\cdot a_{00}^{8} + (05)^{16}\cdot a_{00}^{16} + (fe)^{16}\cdot a_{00}^{32} + (7f)^{16}\cdot a_{00}^{64} + (5a)^{16}\cdot a_{00}^{128} + (05)^{16}\right)^{-1} \\ \left((5a)^{32}\cdot a_{00} + (78)^{32}\cdot a_{00}^{2} + (59)^{32}\cdot a_{00}^{4} + (db)^{32}\cdot a_{00}^{8} + (6e)^{32}\cdot a_{00}^{16} + (05)^{32}\cdot a_{00}^{32} + (fe)^{32}\cdot a_{00}^{64} + (7f)^{32}\cdot a_{00}^{128} + (05)^{32}\right)^{-1} \\ \left((7f)^{64}\cdot a_{00} + (5a)^{64}\cdot a_{00}^{2} + (78)^{64}\cdot a_{00}^{4} + (59)^{64}\cdot a_{00}^{8} + (db)^{64}\cdot a_{00}^{16} + (6e)^{64}\cdot a_{00}^{32} + (05)^{64}\cdot a_{00}^{64} + (fe)^{64}\cdot a_{00}^{128} + (05)^{64}\right)^{-1} \\ \left((fe)^{128}\cdot a_{00} + (7f)^{128}\cdot a_{00}^{2} + (5a)^{128}\cdot a_{00}^{4} + (78)^{128}\cdot a_{00}^{8} + (59)^{128}\cdot a_{00}^{16} + (db)^{128}\cdot a_{00}^{32} + (6e)^{128}\cdot a_{00}^{64} + (05)^{128}\cdot a_{00}^{128} + (05)^{128}\right)^{-1} \\ \vdots \\ (05\cdot a_{33} + fe\cdot a_{33}^{2} + 7f\cdot a_{33}^{4} + 5a\cdot a_{33}^{8} + 78\cdot a_{33}^{16} + 59\cdot a_{33}^{32} + db\cdot a_{33}^{64} + 6e\cdot a_{33}^{128} + 05)^{-1} \\ \left((6e)^{2}\cdot a_{33} + (05)^{2}\cdot a_{33}^{2} + (fe)^{2}\cdot a_{33}^{4} + (7f)^{2}\cdot a_{33}^{8} + (5a)^{2}\cdot a_{33}^{16} + (78)^{2}\cdot a_{33}^{32} + (59)^{2}\cdot a_{33}^{64} + (db)^{2}\cdot a_{33}^{128} + (05)^{2}\right)^{-1} \\ \left((db)^{4}\cdot a_{33} + (6e)^{4}\cdot a_{33}^{2} + (05)^{4}\cdot a_{33}^{4} + (fe)^{4}\cdot a_{33}^{8} + (7f)^{4}\cdot a_{33}^{16} + (5a)^{4}\cdot a_{33}^{32} + (78)^{4}\cdot a_{33}^{64} + (59)^{4}\cdot a_{33}^{128} + (05)^{4}\right)^{-1} \\ \left((5a)^{8}\cdot a_{33} + (db)^{8}\cdot a_{33}^{2} + (6e)^{8}\cdot a_{33}^{4} + (05)^{8}\cdot a_{33}^{8} + (fe)^{8}\cdot a_{33}^{16} + (7f)^{8}\cdot a_{33}^{32} + (5a)^{8}\cdot a_{33}^{64} + (78)^{8}\cdot a_{33}^{128} + (05)^{8}\right)^{-1} \\ \left((78)^{16}\cdot a_{33} + (5a)^{16}\cdot a_{33}^{2} + (db)^{16}\cdot a_{33}^{4} + (6e)^{16}\cdot a_{33}^{8} + (05)^{16}\cdot a_{33}^{16} + (fe)^{16}\cdot a_{33}^{32} + (7f)^{16}\cdot a_{33}^{64} + (5a)^{16}\cdot a_{33}^{128} + (05)^{16}\right)^{-1} \\ \left((5a)^{32}\cdot a_{33} + (78)^{32}\cdot a_{33}^{2} + (59)^{32}\cdot a_{33}^{4} + (db)^{32}\cdot a_{33}^{8} + (6e)^{32}\cdot a_{33}^{16} + (05)^{32}\cdot a_{33}^{32} + (fe)^{32}\cdot a_{33}^{64} + (7f)^{32}\cdot a_{33}^{128} + (05)^{32}\right)^{-1} \\ \left((7f)^{64}\cdot a_{33} + (5a)^{64}\cdot a_{33}^{2} + (78)^{64}\cdot a_{33}^{4} + (59)^{64}\cdot a_{33}^{8} + (db)^{64}\cdot a_{33}^{16} + (6e)^{64}\cdot a_{33}^{32} + (05)^{64}\cdot a_{33}^{64} + (fe)^{64}\cdot a_{33}^{128} + (05)^{64}\right)^{-1} \\ \left((fe)^{128}\cdot a_{33} + (7f)^{128}\cdot a_{33}^{2} + (5a)^{128}\cdot a_{33}^{4} + (78)^{128}\cdot a_{33}^{8} + (59)^{128}\cdot a_{33}^{16} + (db)^{128}\cdot a_{33}^{32} + (6e)^{128}\cdot a_{33}^{64} + (05)^{128}\cdot a_{33}^{128} + (05)^{128}\right)^{-1} \end{pmatrix}$$

5.3 InvShiftRow Operation

In inverse AES cipher, the InvShiftRow step is also a cyclic shift of the rows of the state $a \in A$ over different offsets: row 0 of the state vector is shifted over 0 byte, row 1 over 3 bytes, row 2 over 2 bytes, row 3 over 1 byte leftwards, as follows:

$$a = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \xrightarrow{InvShiftRows} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{13} & a_{10} & a_{11} & a_{12} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{31} & a_{32} & a_{33} & a_{30} \end{pmatrix}$$

By considering the state vector $a \in A$ as a column vector, the process can be viewed as the transformation of the components of a column vector as follows:

$$a = \begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{pmatrix} \xrightarrow{InvShiftRows} \begin{pmatrix} a_{00} \\ a_{13} \\ a_{22} \\ a_{31} \\ a_{01} \\ a_{10} \\ a_{23} \\ a_{32} \\ a_{02} \\ a_{11} \\ a_{20} \\ a_{33} \\ a_{03} \\ a_{12} \\ a_{21} \\ a_{30} \end{pmatrix}.$$

Then we can further represent this transformation by a matrix multiplication of the state vector $a \in A$ by a (16×16) F matrix R_A^{-1}.

$$R_A^{-1} \cdot a = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{pmatrix} = \begin{pmatrix} a_{00} \\ a_{13} \\ a_{22} \\ a_{31} \\ a_{01} \\ a_{10} \\ a_{23} \\ a_{32} \\ a_{02} \\ a_{11} \\ a_{20} \\ a_{33} \\ a_{03} \\ a_{12} \\ a_{21} \\ a_{30} \end{pmatrix}$$

We can also extend a (16×16) F matrix R_A^{-1} into a (128×128) F matrixR_B^{-1} in a similar way, in order to represent this transformation by a matrix multiplication of the state vector $b \in B$ with R_B^{-1}. Therefore, we extend each $'1'$ component in R_A^{-1} to the following (8×8) F block

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

and each $'0'$ component to

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

Then the InvShiftRow in BES can be represented as follows:

$$R_B^{-1}\cdot b=\begin{pmatrix}
1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&&&&&\cdots&&&&&&&&&&\cdots&&&&0\\
0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1\\
&&&&&&&&&&&&&&&&&&1&0&0&0&0&0&0&0\\
&&&&&&&&&&&&&&&&&&0&1&0&0&0&0&0&0\\
&&&&&&&&&&&&&&&&&&0&0&1&0&0&0&0&0\\
&&&&&&&&&&&&&&&&&&0&0&0&1&0&0&0&0\\
&&&0&&&&&&&&&\cdots&&&&&&0&0&0&0&1&0&0&0&0\;\;0\\
&&&&&&&&&&&&&&&&&&0&0&0&0&0&1&0&0\\
&&&&&&&&&&&&&&&&&&0&0&0&0&0&0&1&0\\
&&&&&&&&&&&&&&&&&&0&0&0&0&0&0&0&1\\
&&&\vdots&&&&&&&&&\ddots&&&&&&&&&&&&&&\vdots\\
&&&&&&&&&1&0&0&0&0&0&0&0\\
&&&&&&&&&0&1&0&0&0&0&0&0\\
&&&&&&&&&0&0&1&0&0&0&0&0\\
&&&0&&&&&0\;\;0&0&0&0&1&0&0&0&0&\cdots&&&&&\cdots&&&&0\\
&&&&&&&&&0&0&0&0&1&0&0&0\\
&&&&&&&&&0&0&0&0&0&1&0&0\\
&&&&&&&&&0&0&0&0&0&0&1&0\\
&&&&&&&&&0&0&0&0&0&0&0&1
\end{pmatrix}\begin{pmatrix}
a_{00}\\ a_{00}^{2}\\ a_{00}^{4}\\ a_{00}^{8}\\ a_{00}^{16}\\ a_{00}^{32}\\ a_{00}^{64}\\ a_{00}^{128}\\
a_{10}\\ a_{10}^{2}\\ a_{10}^{4}\\ a_{10}^{8}\\ a_{10}^{16}\\ a_{10}^{32}\\ a_{10}^{64}\\ a_{10}^{128}\\
\vdots\\
a_{33}\\ a_{33}^{2}\\ a_{33}^{4}\\ a_{33}^{8}\\ a_{33}^{16}\\ a_{33}^{32}\\ a_{33}^{64}\\ a_{33}^{128}
\end{pmatrix}=\begin{pmatrix}
a_{00}\\ a_{00}^{2}\\ a_{00}^{4}\\ a_{00}^{8}\\ a_{00}^{16}\\ a_{00}^{32}\\ a_{00}^{64}\\ a_{00}^{128}\\
a_{11}\\ a_{11}^{2}\\ a_{11}^{4}\\ a_{11}^{8}\\ a_{11}^{16}\\ a_{11}^{32}\\ a_{11}^{64}\\ a_{11}^{128}\\
\vdots\\
a32\\ a32^{2}\\ a32^{4}\\ a32^{8}\\ a32^{16}\\ a32^{32}\\ a32^{64}\\ a32^{128}
\end{pmatrix}$$

5.4 InvMixColumn Operation

In AES, this step is also defined to modify each column of the state vector $a \in A$ by multiplying it with a fixed polynomial, and the matrix multiplication of this step is represented as follows:

$$C_A^{-1} \cdot a = C_A^{-1} \cdot (y_0, y_1, y_2, y_3) = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Considering this step as a matrix multiplication of the state vector $a \in A$ by a (16×16) F matrix Mix_A^{-1}, we have

$$Mix_A^{-1} \cdot a = \begin{pmatrix} C_A^{-1} & 0 & 0 & 0 \\ 0 & C_A^{-1} & 0 & 0 \\ 0 & 0 & C_A^{-1} & 0 \\ 0 & 0 & 0 & C_A^{-1} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$= \left(\begin{array}{cccccccccccccccc}
0E & 0B & 0D & 09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
09 & 0E & 0B & 0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0D & 09 & 0E & 0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0B & 0D & 09 & 0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0E & 0B & 0D & 09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 09 & 0E & 0B & 0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0D & 09 & 0E & 0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0B & 0D & 09 & 0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0E & 0B & 0D & 09 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 09 & 0E & 0B & 0D & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0D & 09 & 0E & 0B & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0B & 0D & 09 & 0E & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0E & 0B & 0D & 09 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 09 & 0E & 0B & 0D \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0D & 09 & 0E & 0B \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0B & 0D & 09 & 0E
\end{array}\right)
\left(\begin{array}{c}
a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33}
\end{array}\right)$$

Similarly, then we consider 8 versions of C_A^{-1}.

$$\left(C_B^{(k)}\right)^{-1} = \begin{pmatrix}
(0E)^{2^k} & (0B)^{2^k} & (0D)^{2^k} & (09)^{2^k} \\
(09)^{2^k} & (0E)^{2^k} & (0B)^{2^k} & (0D)^{2^k} \\
(0D)^{2^k} & (09)^{2^k} & (0E)^{2^k} & (0B)^{2^k} \\
(0B)^{2^k} & (0D)^{2^k} & (09)^{2^k} & (0E)^{2^k}
\end{pmatrix} \begin{array}{l} for \\ k = 0, \cdots 7, \end{array}$$

In particular, $(C_B^{(0)})^{-1} = C_A^{-1}$. Furthermore, $\left(C_B^{(k)}\right)^{-1}$ has diffusion properties in a similar way.

$$(C_B^{(k)})^{-1} \cdot y_i^{2^k} = \begin{pmatrix} \left(C_B^{(0)}\right)^{-1} \cdot y_0 \\ \left(C_B^{(1)}\right)^{-1} \cdot y_0^2 \\ \left(C_B^{(2)}\right)^{-1} \cdot y_0^4 \\ \left(C_B^{(3)}\right)^{-1} \cdot y_0^8 \\ \left(C_B^{(4)}\right)^{-1} \cdot y_0^{16} \\ \left(C_B^{(5)}\right)^{-1} \cdot y_0^{32} \\ \left(C_B^{(6)}\right)^{-1} \cdot y_0^{64} \\ \left(C_B^{(7)}\right)^{-1} \cdot y_0^{128} \\ \left(C_B^{(0)}\right)^{-1} \cdot y_1 \\ \left(C_B^{(1)}\right)^{-1} \cdot y_1^2 \\ \left(C_B^{(2)}\right)^{-1} \cdot y_1^4 \\ \left(C_B^{(3)}\right)^{-1} \cdot y_1^8 \\ \left(C_B^{(4)}\right)^{-1} \cdot y_1^{16} \\ \left(C_B^{(5)}\right)^{-1} \cdot y_1^{32} \\ \left(C_B^{(6)}\right)^{-1} \cdot y_1^{64} \\ \left(C_B^{(7)}\right)^{-1} \cdot y_1^{128} \\ \left(C_B^{(0)}\right)^{-1} \cdot y_2 \\ \left(C_B^{(1)}\right)^{-1} \cdot y_2^2 \\ \left(C_B^{(2)}\right)^{-1} \cdot y_2^4 \\ \left(C_B^{(3)}\right)^{-1} \cdot y_2^8 \\ \left(C_B^{(4)}\right)^{-1} \cdot y_2^{16} \\ \left(C_B^{(5)}\right)^{-1} \cdot y_2^{32} \\ \left(C_B^{(6)}\right)^{-1} \cdot y_2^{64} \\ \left(C_B^{(7)}\right)^{-1} \cdot y_2^{128} \\ \left(C_B^{(0)}\right)^{-1} \cdot y_3 \\ \left(C_B^{(1)}\right)^{-1} \cdot y_3^2 \\ \left(C_B^{(2)}\right)^{-1} \cdot y_3^4 \\ \left(C_B^{(3)}\right)^{-1} \cdot y_3^8 \\ \left(C_B^{(4)}\right)^{-1} \cdot y_3^{16} \\ \left(C_B^{(5)}\right)^{-1} \cdot y_3^{32} \\ \left(C_B^{(6)}\right)^{-1} \cdot y_3^{64} \\ \left(C_B^{(7)}\right)^{-1} \cdot y_3^{128} \end{pmatrix} \quad \begin{matrix} \mathit{for} \\ 0 \le i \le 3 \end{matrix}$$

$$
= \left(\begin{array}{cccccccccccccccccc}
\left(C_B^{(0)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & \\
0 & \left(C_B^{(1)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & \\
0 & 0 & \left(C_B^{(2)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 & & & & & & & & & & \\
0 & 0 & 0 & \left(C_B^{(3)}\right)^{-1} & 0 & 0 & 0 & 0 & \cdots & \cdots & & & & 0 & & & & \\
0 & 0 & 0 & 0 & \left(C_B^{(4)}\right)^{-1} & 0 & 0 & 0 & & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & \left(C_B^{(5)}\right)^{-1} & 0 & 0 & & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & \left(C_B^{(6)}\right)^{-1} & 0 & & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \left(C_B^{(7)}\right)^{-1} & & & & & & & & & & \\
 & & & \vdots & & & & & \ddots & & & & & \vdots & & & & \\
 & & & \vdots & & & & & & \ddots & & & & \vdots & & & & \\
 & & & & & & & & & & \left(C_B^{(0)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & \left(C_B^{(1)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & 0 & \left(C_B^{(2)}\right)^{-1} & 0 & 0 & 0 & 0 & 0 \\
 & & & 0 & & & & & \cdots & \cdots & 0 & 0 & 0 & \left(C_B^{(3)}\right)^{-1} & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & 0 & 0 & 0 & \left(C_B^{(4)}\right)^{-1} & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & \left(C_B^{(5)}\right)^{-1} & 0 & 0 \\
 & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & \left(C_B^{(6)}\right)^{-1} & 0 \\
 & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \left(C_B^{(7)}\right)^{-1}
\end{array}\right)
\begin{pmatrix} y_0 \\ y_0^2 \\ y_0^4 \\ y_0^8 \\ y_0^{16} \\ y_0^{32} \\ y_0^{64} \\ y_0^{128} \\ \vdots \\ \vdots \\ y_3 \\ y_3^2 \\ y_3^4 \\ y_3^8 \\ y_3^{16} \\ y_3^{32} \\ y_3^{64} \\ y_3^{128} \end{pmatrix}
$$

Then we use this to define Mix_B^{-1}. Take the first column $y_0 = (a_{00}, a_{10}, a_{20}, a_{30})$ of the state vector a in AES as an example. Thus, the InvMixColumn operation on the vector conjugate set of y_0 in BES is described as follows:

$$\left(\begin{array}{cccccccccccccc}
0E & 0B & 0D & 09 & & & & & & & & & & \\
09 & 0E & 0B & 0D & \cdots & & & & & 0 & & & & \\
0D & 09 & 0E & 0B & & & & & & & & & & \\
0B & 0D & 09 & 0E & & & & & & & & & & \\
 & & & & & (0E)^2 & (0B)^2 & (0D)^2 & (09)^2 & & & & & \\
\vdots & & & & & (09)^2 & (0E)^2 & (0B)^2 & (0D)^2 & \vdots & & & & \\
\vdots & & & & & (0D)^2 & (09)^2 & (0E)^2 & (0B)^2 & \vdots & & & & \\
 & & & & & (0B)^2 & (0D)^2 & (09)^2 & (0E)^2 & & & & & \\
 & & & & \ddots & & & & & & & & & \\
 & & & & & & & & & (0E)^{128} & (0B)^{128} & (0D)^{128} & (09)^{128} & \\
0 & & & & \cdots & & & & & (09)^{128} & (0E)^{128} & (0B)^{128} & (0D)^{128} & \\
 & & & & & & & & & (0D)^{128} & (09)^{128} & (0E)^{128} & (0B)^{128} & \\
 & & & & & & & & & (0B)^{128} & (0D)^{128} & (09)^{128} & (0E)^{128} &
\end{array}\right)
\begin{pmatrix}
a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{00}^2 \\ a_{10}^2 \\ a_{20}^2 \\ a_{30}^2 \\ \vdots \\ a_{00}^{128} \\ a_{10}^{128} \\ a_{20}^{128} \\ a_{30}^{128}
\end{pmatrix}$$

$$
= \begin{pmatrix}
0E \cdot a_{00} + 0B \cdot a_{10} + 0D \cdot a_{20} + 09 \cdot a_{30} \\
09 \cdot a_{00} + 0E \cdot a_{10} + 0B \cdot a_{20} + 0D \cdot a_{30} \\
0D \cdot a_{00} + 09 \cdot a_{10} + 0E \cdot a_{20} + 0B \cdot a_{30} \\
0B \cdot a_{00} + 0D \cdot a_{10} + 09 \cdot a_{20} + 0E \cdot a_{30} \\
(0E)^2 \cdot a_{00}^2 + (0B)^2 \cdot a_{10}^2 + (0D)^2 \cdot a_{20}^2 + (09)^2 \cdot a_{30}^2 \\
(09)^2 \cdot a_{00}^2 + (0E)^2 \cdot a_{10}^2 + (0B)^2 \cdot a_{20}^2 + (0D)^2 \cdot a_{30}^2 \\
(0D)^2 \cdot a_{00}^2 + (09)^2 \cdot a_{10}^2 + (0E)^2 \cdot a_{20}^2 + (0B)^2 \cdot a_{30}^2 \\
(0B)^2 \cdot a_{00}^2 + (0D)^2 \cdot a_{10}^2 + (09)^2 \cdot a_{20}^2 + (0E)^2 \cdot a_{30}^2 \\
\vdots \\
\vdots \\
(0E)^{128} \cdot a_{00}^{128} + (0B)^{128} \cdot a_{10}^{128} + (0D)^{128} \cdot a_{20}^{128} + (09)^{128} \cdot a_{30}^{128} \\
(09)^{128} \cdot a_{00}^{128} + (0E)^{128} \cdot a_{10}^{128} + (0B)^{128} \cdot a_{20}^{128} + (0D)^{128} \cdot a_{30}^{128} \\
(0D)^{128} \cdot a_{00}^{128} + (09)^{128} \cdot a_{10}^{128} + (0E)^{128} \cdot a_{20}^{128} + (0B)^{128} \cdot a_{30}^{128} \\
(0B)^{128} \cdot a_{00}^{128} + (0D)^{128} \cdot a_{10}^{128} + (09)^{128} \cdot a_{20}^{128} + (0E)^{128} \cdot a_{30}^{128}
\end{pmatrix}
$$

Then we reorder the output of $(C_B^{(k)})^{-1} \cdot y_0^{2^k}$, with the purpose of reserving the vector conjugacy, to derive $(Mix_B^{(y_0)})^{-1}$ from it. Therefore, InvMixColumn step in BES on the vector conjugate set of column y_0 can be represented by a matrix multiplication with $(Mix_B^{(y_0)})^{-1}$, where

$$(Mix_B^{(y_0)})^{-1} =$$

$$\left(\begin{array}{cccccccccccccccccccccccccccccccc}
0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (0E) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)''' \\
09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (09) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)''' \\
0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (0D) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)''' \\
0B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0E & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (0B) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (0B)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)'' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'' & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (0B)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)' & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (0B)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0D)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (09)''' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (0E)'''
\end{array}\right)
\begin{pmatrix}
a \\ a' \\ a \\ a' \\ a'' \\ a'' \\ a' \\ a''' \\
a \\ a' \\ a \\ a' \\ a'' \\ a'' \\ a' \\ a''' \\
a \\ a' \\ a \\ a' \\ a'' \\ a'' \\ a' \\ a''' \\
a \\ a' \\ a \\ a' \\ a'' \\ a'' \\ a' \\ a'''
\end{pmatrix}$$

Similarly, we can deduce $(Mix_B^{y_1})^{-1}, (Mix_B^{y_2})^{-1}, (Mix_B^{y_3})^{-1}$ for the vector conjugate set of column y_1, y_2, y_3. Each of these is identical to $(Mix_B^{y_0})^{-1}$.

Therefore, the entire matrix multiplication representation of the InvMixColumn on the state vector $b \in B$ is given by:

$$Mix_B^{-1} \cdot b = Diag_4((Mix_B^{y_0})^{-1}) \cdot b$$

5.5 AddRoundKey Operation

AddRoundKey is its own inverse. In the AES, this steps modifies the state vector $a \in A$ by combining a round key $(k_A)_i \in A$ with the bitwise EXOR operation: $a \longrightarrow a + (k_A)_i$. Accordingly, in the BES, the similar bitwise EXOR operation round with a round key $(k_B)_i \in B$ has applied to the state vector $b \in B : b \longrightarrow b + (k_B)_i$, where $0 \leq i \leq N_{round} = 10$.

5.6 Inverse Key Schedule

In AES, the only difference between the key schedule and the inverse one is that InvMixColumn is applied in the inverse one after the key expansion. Since I use the vector conjugate set of the AES ExpandedKey as the BES ExpandedKey instead, here the inverse key schedule of BES is derived from the AES inverse key schedule in a similar way.

Chapter 6

Testing Analysis of the BES Implementation

The following chapter is an original contribution and analyzes the testing results derived from the BES implementation.

6.1 Testing Results

Although the following testing data may be considered preliminary in that it does not involve a large volume of data, the testing is based on a random plaintext selection, which gives the testing analysis great generality.

In an effort to explore how the BES reserves the AES property, here the 128-byte state vector b of BES cipher is represented by a table with 16 rows and 8 columns, wherein each row represents the vector conjugate set of the corresponding byte a_{ij} in the state vector a

of AES. Thereby, the first column a_{ij} represents the state vector a.

Randomly, take a plaintext $a = (7b, 17, 17, 15, 2d, 4c, 59, 17, 38, 4d, 4d, 2d, de, 7b, ea, 03)$ of AES with the original key= $(d, 7b, 1c, 7a, f5, 7, bd, cb, a6, 4b, 7, 91, 16, 9c, ea, 20)$ as an example.

Thus, the 10-round AES encrypted data is

$$E_A(a) = (9f, d6, d4, 21, 8e, c8, 6a, a8, 57, bd, 99, 4, b6, 1e, 6, db)$$

and the corresponding decrypted data is

$$D_A(E_A(a)) = (7b, 17, 17, 15, 2d, 4c, 59, 17, 38, 4d, 4d, 2d, de, 7b, ea, 3) = a.$$

Accordingly, the tabular representation of the corresponding plaintext state b in BES is given in Table 4, wherein the first column represents a.

	a_{ij}	a_{ij}^2	a_{ij}^4	a_{ij}^8	a_{ij}^{16}	a_{ij}^{32}	a_{ij}^{64}	a_{ij}^{128}
a_{00}	7b	99	c0	31	76	c8	71	dd
a_{10}	17	0e	54	a0	f6	52	b4	fd
a_{20}	17	0e	54	a0	f6	52	b4	fd
a_{30}	15	0a	44	bb	a8	b6	f9	07
a_{01}	2d	3d	26	78	9c	d1	2b	29
a_{11}	4c	fb	03	05	11	1a	5f	e5
a_{21}	59	f1	47	be	b9	ac	a6	e2
a_{31}	17	0e	54	a0	f6	52	b4	fd
a_{02}	38	37	62	c3	34	67	d2	2e
a_{12}	4d	fa	02	04	10	1b	5e	e4
a_{22}	4d	fa	02	04	10	1b	5e	e4
a_{32}	2d	3d	26	78	9c	d1	2b	29
a_{03}	de	7e	88	da	6e	93	84	8a
a_{13}	7b	99	c0	31	76	c8	71	dd
a_{23}	ea	19	5a	f4	56	a4	e6	49
a_{33}	03	05	11	1a	5f	e5	4c	fb

Table 4: A BES Test Case: Plaintext b

And the tabular representation of the corresponding original key of BES is given in Table

5.

	k_{ij}	k_{ij}^2	k_{ij}^4	k_{ij}^8	k_{ij}^{16}	k_{ij}^{32}	k_{ij}^{64}	k_{ij}^{128}
k_{00}	0d	51	b1	ec	0d	51	b1	ec
k_{10}	7b	99	c0	31	76	c8	71	dd
k_{20}	1c	4b	ee	09	41	aa	b2	e9
k_{30}	7a	98	c1	30	77	c9	70	dc
k_{01}	f5	57	a5	e7	48	eb	18	5b
k_{11}	07	15	0a	44	bb	a8	b6	f9
k_{21}	bd	bc	bd	bc	bd	bc	bd	bc
k_{31}	cb	74	cc	61	c6	25	7d	8d
k_{02}	a6	e2	59	f1	47	be	b9	ac
k_{12}	4b	ee	09	41	aa	b2	e9	1c
k_{22}	07	15	0a	44	bb	a8	b6	f9
k_{23}	91	80	9a	c5	20	6c	97	94
k_{03}	16	0f	55	a1	f7	53	b5	fc
k_{13}	9c	d1	2b	29	2d	3d	26	78
k_{23}	ea	19	5a	f4	56	a4	e6	49
k_{33}	20	6c	97	94	91	80	9a	c5

Table 5: A BES Test Case: Original Key

After a 10-round encryption of BES, the ciphertext state $E_B(b)$ is given in Table 6, wherein the first column equals to the AES ciphertext $(E_A(a))$.

	$E_B(a_{ij})$	$E_B(a_{ij}^2)$	$E_B(a_{ij}^4)$	$E_B(a_{ij}^8)$	$E_B(a_{ij}^{16})$	$E_B(a_{ij}^{32})$	$E_B(a_{ij}^{64})$	$E_B(a_{ij}^{128})$
a_{00}	9f	7d	84	71	d4	cc	88	c9
a_{10}	d6	33	a2	0a	cc	c7	5b	1e
a_{20}	d4	7a	12	0a	1b	c5	1f	ba
a_{30}	21	3b	ad	31	61	d9	e1	6f
a_{01}	8e	75	10	22	dc	96	5c	7d
a_{11}	c8	63	16	cd	dd	61	e6	ce
a_{21}	6a	df	ed	5b	de	2b	15	4f
a_{31}	a8	80	ef	bd	f7	10	2e	df
a_{02}	57	a8	66	2b	a1	51	ce	8e
a_{12}	bd	62	a9	f0	5e	d7	be	19
a_{22}	99	0e	6a	62	22	2e	a7	b0
a_{32}	04	00	54	06	ed	79	83	6c
a_{03}	b6	a7	f1	74	a3	c9	72	e9
a_{13}	1e	7e	b8	0d	b3	ef	c5	6f
a_{23}	6f	41	10	a4	76	5d	1f	05
a_{33}	db	ce	e1	02	75	1e	2e	20

Table 6: A BES Test Case: Ciphertext $E_B(b)$

Then after a 10-round decryption on the ciphertext state $E_B(b)$, the result $D_B(E_B(b))$ is given in Table 7, wherein the first column equals to the AES result of $D_A(E_A(a))$.

	$D_B(E_B(a_{ij}))$	$D_B(E_B(a_{ij}^2))$	$D_B(E_B(a_{ij}^4))$	$D_B(E_B(a_{ij}^8))$	$D_B(E_B(a_{ij}^{16}))$	$D_B(E_B(a_{ij}^{32}))$	$D_B(E_B(a_{ij}^{64}))$	$D_B(E_B(a_{ij}^{128}))$
a_{00}	7b	38	f9	a7	92	0e	16	90
a_{10}	17	34	2d	18	67	38	9c	a5
a_{20}	17	39	71	76	1e	8a	32	79
a_{30}	15	a3	a3	a2	36	d0	3e	0f
a_{01}	2d	7c	2d	4e	9b	02	0e	69
a_{11}	4c	aa	b8	e6	2b	8d	8b	2d
a_{21}	59	a6	dd	8d	11	08	21	af
a_{31}	17	b9	0c	d8	94	5d	82	54
a_{02}	38	cc	85	23	fe	bb	0e	18
a_{12}	4d	01	fb	59	ab	d0	aa	69
a_{22}	4d	3c	f4	91	f7	d4	c7	a0
a_{23}	2d	d4	b7	d7	1b	97	1b	49
a_{03}	de	a5	fb	34	31	7e	2a	53
a_{13}	7b	c7	d7	d6	2f	a8	da	32
a_{23}	ea	94	cb	85	ad	4a	a1	32
a_{33}	03	fd	fb	b5	7e	1f	e8	57

Table 7: A BES Test Case : Decipher(Encipher(*b*))

Obviously, the plaintext in Table 4 and the Decipher(Encipher(plaintext)) in Table 7 is only partially matched, namely that only the first columns of the tables match. This means that BES can fully replicate the AES actions, not only encryption but also decryption, on the first bytes of the vector conjugate sets. Therefore, the conclusion that BES fully respects the encryption and decryption of AES is true.

However, the inverse BES cipher, derived from the original design principles, can not convert the remaining bytes in the BES encrypted data into clear data. Since the BES is not intended to be used for encryption, this decryption issue is acceptable. Next, the following section will further analysis on this decryption issue.

6.2 Testing Analysis

The issue of BES decryption, described in the previous section, results from the fact of "originally"[1] implementing the InvByteSub and InvMixColumn steps in BES decryption.

[1] "originally" refers to the inverse BES cipher derived from the design principles of BES cipher

1. **InvByteSub Operation**

In AES, the InvByteSub operation denoted by S_{RD}^{-1} can be represented by:

$$S_{RD}^{-1}(a) = g^{-1}(f^{-1}(a) + 05) = g(f^{-1}(a) + 05)$$

Since g is self-inverse, f^{-1} is essential to describe the InvByteSub step. Therefore, for $S_{RD}(a) = f(g(a)) + 63$ and

$$f = 05 \cdot a_{ij} + 09 \cdot a_{ij}^2 + f9 \cdot a_{ij}^4 + 25 \cdot a_{ij}^8 + f4 \cdot a_{ij}^{16} + 01 \cdot a_{ij}^{32} + b5 \cdot a_{ij}^{64} + 8f \cdot a_{ij}^{128},$$

by applying Lagrange interpolation, there exists

$$f^{-1} = 05 \cdot a_{ij} + fe \cdot a_{ij}^2 + 7f \cdot a_{ij}^4 + 5a \cdot a_{ij}^8 + 78 \cdot a_{ij}^{16} + 59 \cdot a_{ij}^{32} + db \cdot a_{ij}^{64} + 6e \cdot a_{ij}^{128}$$

which can let $a = S_{RD}^{-1}(S_{RD}(a))$.

BES replicates this transformation on the first bytes of the vector conjugate sets, so the first byte of the plaintext state will match the corresponding first byte of $S_{RD}^{-1}(S_{RD}(plaintext))$. However, the cases for the remaining bytes of the vector conjugate sets are different. In ByteSub step, the $2^{nd}, 3^{rd}, \cdots, 7^{th}$ bytes are modified respectively by:

$$f_1 = (8f)^2 \cdot a_{ij} + (05)^2 \cdot a_{ij}^2 + (09)^2 \cdot a_{ij}^4 + (f9)^2 \cdot a_{ij}^8 + (25)^2 \cdot a_{ij}^{16} + (f4)^2 \cdot a_{ij}^{32} + (01)^2 \cdot a_{ij}^{64} + (b5)^2 \cdot a_{ij}^{128}$$
$$f_2 = (b5)^4 \cdot a_{ij} + (8f)^4 \cdot a_{ij}^2 + (05)^4 \cdot a_{ij}^4 + (09)^4 \cdot a_{ij}^8 + (f9)^4 \cdot a_{ij}^{16} + (25)^4 \cdot a_{ij}^{32} + (f4)^4 \cdot a_{ij}^{64} + (01)^4 \cdot a_{ij}^{128}$$
$$f_3 = (01)^8 \cdot a_{ij} + (b5)^8 \cdot a_{ij}^2 + (8f)^8 \cdot a_{ij}^4 + (05)^8 \cdot a_{ij}^8 + (09)^8 \cdot a_{ij}^{16} + (f9)^8 \cdot a_{ij}^{32} + (25)^8 \cdot a_{ij}^{64} + (f4)^8 \cdot a_{ij}^{128}$$
$$f_4 = (f4)^{16} \cdot a_{ij} + (01)^{16} \cdot a_{ij}^2 + (b5)^{16} \cdot a_{ij}^4 + (8f)^{16} \cdot a_{ij}^8 + (05)^{16} \cdot a_{ij}^{16} + (09)^{16} \cdot a_{ij}^{32} + (f9)^{16} \cdot a_{ij}^{64} + (25)^{16} \cdot a_{ij}^{128}$$
$$f_5 = (25)^{32} \cdot a_{ij} + (f4)^{32} \cdot a_{ij}^2 + (01)^{32} \cdot a_{ij}^4 + (b5)^{32} \cdot a_{ij}^8 + (8f)^{32} \cdot a_{ij}^{16} + (05)^{32} \cdot a_{ij}^{32} + (09)^{32} \cdot a_{ij}^{64} + (f9)^{32} \cdot a_{ij}^{128}$$
$$f_6 = (f9)^{64} \cdot a_{ij} + (25)^{64} \cdot a_{ij}^2 + (f4)^{64} \cdot a_{ij}^4 + (01)^{64} \cdot a_{ij}^8 + (b5)^{64} \cdot a_{ij}^{16} + (8f)^{64} \cdot a_{ij}^{32} + (05)^{64} \cdot a_{ij}^{64} + (09)^{64} \cdot a_{ij}^{128}$$
$$f_7 = (09)^{128} \cdot a_{ij} + (f9)^{128} \cdot a_{ij}^2 + (25)^{128} \cdot a_{ij}^4 + (f4)^{128} \cdot a_{ij}^8 + (01)^{128} \cdot a_{ij}^{16} + (b5)^{128} \cdot a_{ij}^{32} + (8f)^{128} \cdot a_{ij}^{64} + (05)^{128} \cdot a_{ij}^{128}$$

But the inverse $f_1^{-1}, f_3^{-1}, \cdots, f_7^{-1}$ are not simply the following polynomials used in the "original" inverse BES cipher:

$$\begin{aligned}
f_1^{-1} &\neq (6e)^2 \cdot a_{ij} + (05)^2 \cdot a_{ij}^2 + (fe)^2 \cdot a_{ij}^4 + (7f)^2 \cdot a_{ij}^8 + (5a)^2 \cdot a_{ij}^{16} + (78)^2 \cdot a_{ij}^{32} + (59)^2 \cdot a_{ij}^{64} + (db)^2 \cdot a_{ij}^{128} \\
f_2^{-1} &\neq (db)^4 \cdot a_{ij} + (6e)^4 \cdot a_{ij}^2 + (05)^4 \cdot a_{ij}^4 + (fe)^4 \cdot a_{ij}^8 + (7f)^4 \cdot a_{ij}^{16} + (5a)^4 \cdot a_{ij}^{32} + (78)^4 \cdot a_{ij}^{64} + (59)^4 \cdot a_{ij}^{128} \\
f_3^{-1} &\neq (59)^8 \cdot a_{ij} + (db)^8 \cdot a_{ij}^2 + (6e)^8 \cdot a_{ij}^4 + (05)^8 \cdot a_{ij}^8 + (fe)^8 \cdot a_{ij}^{16} + (7f)^8 \cdot a_{ij}^{32} + (5a)^8 \cdot a_{ij}^{64} + (78)^8 \cdot a_{ij}^{128} \\
f_4^{-1} &\neq (78)^{16} \cdot a_{ij} + (59)^{16} \cdot a_{ij}^2 + (db)^{16} \cdot a_{ij}^4 + (6e)^{16} \cdot a_{ij}^8 + (05)^{16} \cdot a_{ij}^{16} + (fe)^{16} \cdot a_{ij}^{32} + (7f)^{16} \cdot a_{ij}^{64} + (5a)^{16} \cdot a_{ij}^{128} \\
f_5^{-1} &\neq (5a)^{32} \cdot a_{ij} + (78)^{32} \cdot a_{ij}^2 + (59)^{32} \cdot a_{ij}^4 + (db)^{32} \cdot a_{ij}^8 + (6e)^{32} \cdot a_{ij}^{16} + (05)^{32} \cdot a_{ij}^{32} + (fe)^{32} \cdot a_{ij}^{64} + (7f)^{32} \cdot a_{ij}^{128} \\
f_6^{-1} &\neq (7f)^{64} \cdot a_{ij} + (5a)^{64} \cdot a_{ij}^2 + (78)^{64} \cdot a_{ij}^4 + (59)^{64} \cdot a_{ij}^8 + (db)^{64} \cdot a_{ij}^{16} + (6e)^{64} \cdot a_{ij}^{32} + (05)^{64} \cdot a_{ij}^{64} + (fe)^{64} \cdot a_{ij}^{128} \\
f_7^{-1} &\neq (fe)^{128} \cdot a_{ij} + (7f)^{128} \cdot a_{ij}^2 + (5a)^{128} \cdot a_{ij}^4 + (78)^{128} \cdot a_{ij}^8 + (59)^{128} \cdot a_{ij}^{16} + (db)^{128} \cdot a_{ij}^{32} + (6e)^{128} \cdot a_{ij}^{64} + (05)^{128} \cdot a_{ij}^{128}
\end{aligned}$$

Therefore, by using such an inverse cipher, the remaining bytes of the ciphertext state can not be efficiently decrypted. One possible solution can be found by enlarging the BES state space to a certain degree, wherein the vector conjugate set of BES is considered as a component of the enlarged new space, thereby possibly enabling the BES decryption in the enlarged new space efficient to the whole vector conjugate set.

2. **InvMixColumn Operation**

Similarly, the $(C_B^{(k)})^{-1}$, the inverse of $C_B^{(k)}$ in the previous defined inverse BES cipher, is also improper, where $k = 0, \cdots 7$ and

$$\left(C_B^{(k)}\right)^{-1} = \begin{pmatrix} (0E)^{2^k} & (0B)^{2^k} & (0D)^{2^k} & (09)^{2^k} \\ (09)^{2^k} & (0E)^{2^k} & (0B)^{2^k} & (0D)^{2^k} \\ (0D)^{2^k} & (09)^{2^k} & (0E)^{2^k} & (0B)^{2^k} \\ (0B)^{2^k} & (0D)^{2^k} & (09)^{2^k} & (0E)^{2^k} \end{pmatrix}, C_B^{(k)} = \begin{pmatrix} (02)^{2^k} & (03)^{2^k} & 1 & 1 \\ 1 & (02)^{2^k} & (03)^{2^k} & 1 \\ 1 & 1 & (02)^{2^k} & (03)^{2^k} \\ (03)^{2^k} & 1 & 1 & (02)^{2^k} \end{pmatrix}.$$

Here another version of $(C_B^{(k)})^{-1}$ can be given as follows:

$$(C_B^{(0)})^{-1} = \begin{pmatrix} (0E) & (0B) & (0D) & (09) \\ (09) & (0E) & (0B) & (0D) \\ (0D) & (09) & (0E) & (0B) \\ (0B) & (0D) & (09) & (0E) \end{pmatrix}, (C_B^{(1)})^{-1} = \begin{pmatrix} (54)^2 & (45)^2 & (51)^2 & (41)^2 \\ (41)^2 & (54)^2 & (45)^2 & (51)^2 \\ (51)^2 & (41)^2 & (54)^2 & (45)^2 \\ (45)^2 & (51)^2 & (41)^2 & (54)^2 \end{pmatrix}$$

$$(C_B^{(2)})^{-1} = \begin{pmatrix} (a0)^4 & (ba)^4 & (b1)^4 & (aa)^4 \\ (aa)^4 & (a0)^4 & (ba)^4 & (b1)^4 \\ (b1)^4 & (aa)^4 & (a0)^4 & (ba)^4 \\ (ba)^4 & (b1)^4 & (aa)^4 & (a0)^4 \end{pmatrix}, (C_B^{(3)})^{-1} = \begin{pmatrix} (f6)^8 & (a9)^8 & (ec)^8 & (b2)^8 \\ (b2)^8 & (f6)^8 & (a9)^8 & (ec)^8 \\ (ec)^8 & (b2)^8 & (f6)^8 & (a9)^8 \\ (a9)^8 & (ec)^8 & (b2)^8 & (f6)^8 \end{pmatrix}$$

$$(C_B^{(4)})^{-1} = \begin{pmatrix} (52)^{16} & (b7)^{16} & (0d)^{16} & (e9)^{16} \\ (e9)^{16} & (52)^{16} & (b7)^{16} & (0d)^{16} \\ (0d)^{16} & (e9)^{16} & (52)^{16} & (b7)^{16} \\ (b7)^{16} & (0d)^{16} & (e9)^{16} & (52)^{16} \end{pmatrix}, (C_B^{(5)})^{-1} = \begin{pmatrix} (b4)^{32} & (f8)^{32} & (51)^{32} & (1c)^{32} \\ (1c)^{32} & (b4)^{32} & (f8)^{32} & (51)^{32} \\ (51)^{32} & (1c)^{32} & (b4)^{32} & (f8)^{32} \\ (f8)^{32} & (51)^{32} & (1c)^{32} & (b4)^{32} \end{pmatrix}$$

$$(C_B^{(6)})^{-1} = \begin{pmatrix} (fd)^{64} & (06)^{64} & (b1)^{64} & (4b)^{64} \\ (4b)^{64} & (fd)^{64} & (06)^{64} & (b1)^{64} \\ (b1)^{64} & (4b)^{64} & (fd)^{64} & (06)^{64} \\ (06)^{64} & (b1)^{64} & (4b)^{64} & (fd)^{64} \end{pmatrix}, (C_B^{(7)})^{-1} = \begin{pmatrix} (17)^{128} & (14)^{128} & (ec)^{128} & (ee)^{128} \\ (ee)^{128} & (17)^{128} & (14)^{128} & (ec)^{128} \\ (ec)^{128} & (ee)^{128} & (17)^{128} & (14)^{128} \\ (14)^{128} & (ec)^{128} & (ee)^{128} & (17)^{128} \end{pmatrix}.$$

Using these 8 new matrices, the new InvMixColumn of BES will decrypt the intermediate state modified by MixColumn for one round correctly. However, this step is byte-position related, which involves reordering the result vector, thereby making this new InvMixColumn only suitable for one round transformation. So this only gives an idea to find the more general InvMixColumn.

From the above testing analysis, conclusions can be drawn that although the encrypted and decrypted data of BES cipher can only be partially matched, BES fully respects the encryption and decryption of AES, which achieves the original design purpose.

Chapter 7

Conclusion and Future Work

There are three important issues related to the BES implementation discussed in the thesis.

First of all, the thesis gives a detailed exploration of the implementation of BES encryption. The key data structure of BES, a 128-byte state vector, is derived from a 16-byte state vector of AES by employing a vector conjugate mapping, where all conjugates of $a_{ij} \in F$ are the powers of a_{ij} in F. Then efforts have been made to replicate the AES actions of a typical round function operating on the state vector within the BES. Moreover, by further translating the the MixColumn, ShiftRow and ByteSub steps of BES into a matrix representation, the round function can be represented in a simple algebraic form: $Round_B(b, (k_B)_i) = Mix_B(R_B(Lin_B(b^{(-1)}))) + (k_B)_i$, where the MixColumn, ShiftRow, ByteSub steps are represented by a matrix multiplication with a (128×128) F-matrix Mix_B, R_B, Lin_B respectively.

Because the decryption of BES is not described the BES specification [7], the next issue was to propose the inverse BES cipher with full respect to the design principles of BES, in order to support the BES implementation. In an effort to explore the inverse round function of BES, the InvMixColumn, InvShifRow and InvByteSub steps are respectively considered as being a matrix multiplication with a (128×128) F-matrix $Mix_B^{-1}, R_B^{-1}, Lin_B^{-1}$ in a similar way. In particular, in order to describe Lin_B^{-1}, Lagrange Formula has been implemented to interpolate the inverse GF(2) linear mapping, where the addition, multiplication and division of two coefficients in Lagrange Formula are respectively the addition, modular multiplication and multiplicative inversion over F.

The thesis further analyzes the testing results of the BES implementation based on a random plaintext selection. The testing data demonstrate that the BES can fully respect the encryption and decryption of the AES, which achieves the original design goal. However, the inverse BES cipher, derived from the original design principles, can not convert the BES encrypted data into a clear data. This decryption issue results from the original idea of implementing InvByteSub and InvMixColumn steps. Furthermore, possible solutions are proposed regarding the decryption issue.

The implementation of BES here demonstrates that BES gives an alternative description of AES by using only simple algebraic operations over F. Moreover, the BES can offer an easier cryptanalysis, which encourages its wide practical use by the cryptanalytic community, with the purpose of getting additional insights into the AES cryptanalysis.

Therefore, more future work is expected to be done. Firstly, the research work to find the suitable InvByteSub and InvMixColumn expects a further step toward solving the problem of BES decryption. Secondly, efforts also need to be made to incorporate the key schedule of BES into the BES implementation in the future.

Bibliography

[1] National Bureau of Standards, Washington D.C. The Data Encryption Standard, FIPS-Pub.46., January 1977. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[2] DES Cracker, 1998. http://www.eff.org/DEScracker/

[3] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[4] John Daemen and Vincent Rijmen. AES Proposal: Rijndael, September 1999. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[5] John Daemen and Vincent Rijmen. Answer to New Observations on Rijndael.Available at http://www.esat.kuleuven.ac.be/ rijmen/rijndael/answer.pdf,11 August 2000.

[6] R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge University Press, 1984.

[7] S. Murphy and M. Robshaw. Essential Algebraic Structure within the AES. In CRYPTO 2002, pages 1C16, 2002.

[8] J. Daemen and V. Rijmen. The Design of Rijndael: AES – The Advanced Encryption Standard. SpringerVerlag, 2002.

[9] N. Ferguson, R. Shroeppel, and D. Whiting. A simple algebraic representation of Rijndael. In S. Vaudenay and A. Youssef, editors, Proceedings of Selected Areas in Cryptography, LNCS, pages 103111, Springer-Verlag, 2001.

[10] S. Murphy and M.J.B. Robshaw. New observations on Rijndael. NIST AES website csrc.nist.gov/encryption/aes, August 2000.

[11] T. Jakobsen and L.R. Knudsen. The interpolation attack on block ciphers. In E. Biham, editor, Proceedings of Fast Software Encryption 1997, LNCS 1267, pages 2840, Springer-Verlag, 1997.

[12] S. Murphy and M.J.B. Robshaw. Further comments on the structure of Rijndael. NIST AES website csrc.nist.gov/encryption/aes, August 2000.

[13] http://csrc.nist.gov/cryptotoolkit/aes/

[14] M. Smid and D. Branstad. Contemporary Cryptology, chapter The Data Encryption Standard: Past and Future. IEEE Press, 1991.

[15] http://www.minrank.org/aes/

[16] S. Lucks. Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In Proceedings of Third AES Conference and also via NIST AES website csrc.nist.gov/encryption/aes, April 2000.

[17] H. Gilbert and M. Minier. A collision attack on seven rounds of Rijndael. Third AES Conference, NIST AES website csrc.nist.gov/encryption/aes, April 2000.

[18] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. IACR eprint server www.iacr.org, April 2002.

[19] N. Courtois, L. Goubin, W. Meier, and J. Tacier. Solving underdefined systems of multivariate quadratic equations. In D. Paillier, editor, Proceedings of Public Key Cryptography 2002, LNCS 2274, pages 211-227, Springer-Verlag, 2002.

[20] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. E.cient algorithms for solving overdefined systems of multivariate polynomial equations. In B. Preneel, editor, Proceedings of Eurocrypt 2000, LNCS 1807, pages 392407, Springer-Verlag, 2000.

Appendix A

Source Code

```
#include <sys/types.h>
#include <sys/timeb.h>
#include <Windows.h.>
#include <stdio.h>
typedef unsigned char bit8;
typedef unsigned int bit32;
FILE* fp;
bit8 shifts[4] = {0,40,80,120};
bit8 invshifts[4] = {0,104,80,56};
int rounds=10;
bit8 A8[256][8];
bit8 A1[256];
int ROOT = 0x11B;
bit8 log[256];
bit8 alog[256];
int prc;
bit32 RC[30];
bit8 ExpandedKey[11][4][4];
bit8 MixColumnBox[4][4] = {0x02,0x03,0x01,0x01,
                           0x01,0x02,0x03,0x01,
                                  0x01,0x01,0x02,0x03,
                                  0x03,0x01,0x01,0x02};
bit8 seed[7][4] = {0x45,0x51,0x41,0x54,
                   0xba,0xb1,0xaa,0xa0,
                   0xa9,0xec,0xb2,0xf6,
                   0xb7,0xd,0xe9,0x52,
                   0xf8,0x51,0x1c,0xb4,
                   0x6,0xb1,0x4b,0xfd,
                   0x14,0xec,0xee,0x17};
bit8 InvMixColumnBox[8][4][4] = {0x0e,0x0b,0x0d,0x09,
                                 0x09,0x0e,0x0b,0x0d,
                                 0x0d,0x09,0x0e,0x0b,
                                 0x0b,0x0d,0x09,0x0e};
bit8 SubByteBox[8][8] = {0x5,0x9,0xf9,0x25,0xf4,0x01,0xb5,0x8f};
bit8 InvSubByteBox[8][8] = {0x05,0xfe,0x7f,0x5a,0x78,0x59,0xdb,0x6e};
bit8 stab[256]=
        {0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x1,0x67,0x2b,0xfe,0xd7,0xab,0x76,
        0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
        0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
        0x4,0xc7,0x23,0xc3,0x18,0x96,0x5,0x9a,0x7,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,0x9,
        0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,0x53,
        0xd1,0x0,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,0xd0,
        0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x2,0x7f,0x50,0x3c,0x9f,0xa8,0x51,
        0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,0xcd,
        0xc,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,0x60,
        0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0xb,0xdb,0xe0,
        0x32,0x3a,0xa,0x49,0x6,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,0xe7,
        0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x8,0xba,
        0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,0x70,
        0x3e,0xb5,0x66,0x48,0x3,0xf6,0xe,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,0xe1,
        0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,0x8c,
        0xa1,0x89,0xd,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0xf,0xb0,0x54,0xbb,0x16};
//function: ShowState
```

```
//Function ShowState stores the intermediate state which is a table with 16 rows and 8 columns

void ShowState(bit8 state[128]){

int i,j;

for (i=0;i<16;i++){

        for (j=0;j<8;j++)

                fprintf(fp, "%x\t",state[i*8+j]);

                        fprintf(fp,"\n");

}

fprintf(fp,"\n");

}

//function: mul is the multiplication on polynomials in GF(2)

bit8 mul(bit8 a, bit8 b){

if (a && b)

                          return alog[(log[a]+log[b])%255];

return 0;

}

//function: BuildLogTables is to initialize the logarithm table and vlog table

void BuildLogTables(){

int i,j;

alog[0] = 1;

for (i = 1; i < 256; i++){

        j = (alog[i-1] << 1) ^ alog[i-1];

if ((j & 0x100) != 0) j ^= ROOT;

        alog[i] = j;

}

for (i = 1; i < 255; i++) log[alog[i]] = i;

}

//function: BuildBoxs is to initialize some vectors for BES calculation

void BuildBoxs(){

int i,j,t;

//the vector A8 are the conjugate vector and the vector A1 is the inverse vector

for (i=0;i<256;i++){

        A8[i][0] = i;
```

```
    for (j=1;j<8;j++){
        A8[i][j] = mul(A8[i][j-1],A8[i][j-1]);
        for (t=0;t<256;t++)
            if (mul(i,t) == 1){
            A1[i] = t;
            break;
        }
}
}
//initialize the InvMixColumnBox
for (i =0;i<7;i++)
for (j=0;j<4;j++){
    InvMixColumnBox[i+1][j][0] = seed[i][3-j];
    InvMixColumnBox[i+1][j][1] = seed[i][j!=0?4-j:j];
    InvMixColumnBox[i+1][j][2] = seed[i][j!=2?abs(j-1):3];
    InvMixColumnBox[i+1][j][3] = seed[i][j!=3?2-j:3];
}
//initialize the constant for key expansion
RC[0] = 0; RC[1] = 1; i = 1;
for (t = 2; t < 30;)
    RC[t++] = (bit8)(i = mul(2, i));
}
//function: BuildSubByteBox is to initialize the vectors for SubByte and InvSubByte
void BuildSubByteBox(){
int i,j;
for (i=1;i<8;i++)
    for (j= 0;j<8;j++){
    //implement the left shift for the vector
        SubByteBox[i][j] = A8[SubByteBox[0][(j-i)<0?j-i+8:j-i]][j];
        InvSubByteBox[i][j] = A8[InvSubByteBox[0][(j-i)<0?i-j:8-j+i]][j];
    }
}
//function: Expansion is to extend the input to its vector conjugate
```

```
void Expansion(bit8 a[16],bit8 b[128]){

for (int i = 0; i < 16; i++)

        for (int j=0; j<8; j++)

                b[i*8+j] = A8[a[i]][j];

}

//function: SubByte is ByteSub in the BES cipher

void SubByte(bit8 b[128]){

bit8 temp, bb[128];

int i,j,t,x;

for (i=0;i<16;i++)

        for (j=0;j<8;j++){

                x = (8-j)%8;

                in temp = mul(SubByteBox[j][x],A1[b[i*8]]);

                in for (t=1;t<8;t++)

                        temp ^= mul(SubByteBox[j][(x+t)%8],A1[A8[b[i*8]][t]]);

//do something to  the different of BES and AES

                bb[i*8+j]=temp^A8[0x63][j];

        }

for (i = 0; i<128;i++)

        b[i] = bb[i];

}

//function: InvSubByte is the InvByteSub in the inverse BES cipher

void InvSubByte(bit8 b[128]){

bit8 temp, bb[128];

int i,j,t,x;

for (i=0;i<16;i++)

        for (j=0;j<8;j++){

                x = (8-j)%8;

                temp = mul(InvSubByteBox[j][x],b[i*8]);

                for (t=1;t<8;t++)

                        temp ^= mul(InvSubByteBox[j][(x+t)%8],A8[b[i*8]][t]);

//do something to  the different of BES and AES

                bb[i*8+j]=A1[temp^A8[0x05][j]];
        }
for (i = 0; i<128;i++)
        b[i] = bb[i];
}
//function: ShiftRows combines ShiftRow and InvShiftRow of BES
//direct: 0—means encryption and 1—means decryption
void ShiftRows(bit8 state[128],bool direct){
bit8 temp[128];
int i,j,t,u; u=0;
if (!direct)
for (i=0; i<4; i++)
        for (j=0; j<4; j++)
                for(t=0; t<8; t++)
                temp[u++] = state[i*32+shifts[j]+t<128?i*32+shifts[j]+t:i*32+shifts[j]+t-128];
else
for (i=0; i<4; i++)
        for (j=0; j<4; j++)
                for(t=0; t<8; t++)
                temp[u++]=state[i*32+invshifts[j]+t<128?i*32+invshifts[j]+t:i*32+invshifts[j]+t-128];
for (i=0;i<128;i++)
        state[i] = temp[i];
}
//function: MixColumns combines MixColumn and InvMixColumn in BES
//direct: 0—means encryption and 1—means decryption
void MixColumns(bit8 state[128],bool direct){
bit8 temp[128];
int i,j,t;
if (!direct)
        for (i=0;i<4;i++)
                for (j=0;j<8;j++)
                        for (t=0;t<4;t++)
                                //do something to adjust the position
                                temp[i*32+j+t*8]= mul(A8[MixColumnBox[t][0]][j],A8[state[i*32+j]][j])
```

```
                              ^mul(A8[MixColumnBox[t][1]][j],A8[state[i*32+j+8]][j])
                              ^mul(A8[MixColumnBox[t][2]][j],A8[state[i*32+j+16]][j])
                              ^mul(A8[MixColumnBox[t][3]][j],A8[state[i*32+j+24]][j]);
else
    for (i=0;i<4;i++)
        for (j=0;j<8;j++)
            for (t=0;t<4;t++)
                //do something to adjust the position
                temp[i*32+j+t*8]= mul(A8[InvMixColumnBox[j][t][0]][j],A8[state[i*32+j]][j])
                          ^mul(A8[InvMixColumnBox[j][t][1]][j],A8[state[i*32+j+8]][j])
                          ^mul(A8[InvMixColumnBox[j][t][2]][j],A8[state[i*32+j+16]][j])
                          ^mul(A8[InvMixColumnBox[j][t][3]][j],A8[state[i*32+j+24]][j]);
for (i=0;i<128;i++)
    state[i] = temp[i];
}
//function: KeyExpansion is to expand the original cipher key
void KeyExpansion(bit8 k[16]){
int i,j,t,prc = 1;
bit8 tk[4][4];
for (j=0;j<4;j++)
    for (i=0;i<4;i++)
        tk[i][j] = k[i*4+j];
t = 0;
for (j=0;(j < 4)&&(t < (rounds+1)*4);j++,t++)
    for (i=0;i<4;i++)
        ExpandedKey[t/4][i][t%4] = tk[i][j];
while (t<(rounds +1)*4){
    for (i =0;i<4;i++)
        tk[i][0] ^= stab[tk[(i+1)%4][4-1]];
tk[0][0] ^= RC[prc];
for(j=1;j<4;j++)
    for (i=0;i<4;i++) tk[i][j] ^= tk[i][j-1];
for (j=0;(j<4)&& (t<(rounds+1)*4);j++,t++)
```

```
          for (i = 0;i<4;i++)
                    ExpandedKey[t/4][i][t%4] = tk[i][j];
}
}
//function: AddRoundKey is the AddRoundKey in BES
void AddRoundKey(bit8 state[128],int index){
int i,j,t;
t =0;
bit8 temp[16],tk[128];
for (i=0;i<4;i++)
          for (j=0;j<4;j++)
                    temp[t++] = ExpandedKey[index][i][j];
//do vector conjugate on the subkey of each round
Expansion(temp,tk);
fprintf(fp,"the subkey of %d round\n",index);
ShowState(tk);
for (i = 0;i<128;i++)
          state[i] ^= tk[i];
}
//function: Encrypt is the BES encryption process
int Encrypt(bit8 a[128]){
int r;
AddRoundKey(a,0);
for (r = 1; r<rounds; r++){
          SubByte(a);
          ShiftRows(a,0);
          MixColumns(a,0);
          AddRoundKey(a,r);
}
SubByte(a);
ShiftRows(a,0);
AddRoundKey(a,rounds);
return 0;
```

```
}
//function: Decrypt is the BES decryption process
int Decrypt(bit8 a[128]){
int r;
AddRoundKey(a,rounds);
InvSubByte(a);
ShiftRows(a,1);
for (r = rounds-1; r>0; r–){
        AddRoundKey(a,r);
        MixColumns(a,1);
        InvSubByte(a);
        1in ShiftRows(a,1);
}
AddRoundKey(a,0);
return 0;
}
//function: coefficient is to calculate the coefficients
//of Lagrange interpolating polynomial with respect to F
void coefficient(int co[256],int k){
int i,j,t;
co[0] = 0;
co[1] = 1;
co[2] = 1;
if (k == 0){
        co[0] = 1;
        co[1] = 1;
        co[2] = 0;
};
if (k == 1){
        co[0] = 0;
        co[1] = 1;
        co[2] = 0;
};
```

```
for (i = 2;i<256 ;i++){
        if (i == k)
                continue;
        t=255;
        while(co[t] == 0)
                t–;
        for (j = t+1; j>0;j–){
                if (j >255) continue;
        co[j] = co[j-1];
        }
co[0] = 0;
for (j = 0; j<t+1;j++)
        co[j] ^= mul(co[j+1],i);
}
}
//function: lagrange is to implement the Lagrange formula with coefficients in F
void lagrange(bit8 result[256]){
int i ,j ,t,x;
int co[256];
int w256[256];
for (i = 0;i<256;i++)
        w256[i] = -1;
for (i=0;i<256;i++){
        for (j=0;j<256;j++)
                co[j] = 0;
        coefficient(co,i);
        t = 1;
        for (j =0; j<256;j++){
                if (j == i)
                        continue;
                x = j^i;
                t = mul(t,x);
        }
t = mul(A1[t],result[i]);
for (j=0;j<256;j++)
        co[j] = mul(co[j],t);
for (j=0;j<256;j++)
        if (w256[j] == -1)
                w256[j] = co[j];
        else
                w256[j] ^= co[j];
}
}
int main(int argc, char* argv[])
{
bit8 a[16]={123,23,23,21,45,76,89,23,56,77,77,45,222,123,234,3};
bit8 ea[128];
bit8 key[16]={13,123,28,122,245,7,189,203,166,75,7,145,22,156,234,32};
fp = fopen("result.txt", "w");
BuildLogTables();
BuildBoxs();
BuildSubByteBox();
Expansion(a,ea);
fprintf(fp,"the expanded plaintext\n");
ShowState(ea);
KeyExpansion(key);
Encrypt(ea);
fprintf(fp,"the encrypted text\n");
ShowState(ea);
Decrypt(ea);
fprintf(fp,"the decrypted text\n");
ShowState(ea);
fclose(fp);
return 0;
}
```