NOTE TO USERS

A DIFFERENTIAL FAULT ATTACK, KEY REPRESENTATION AND MAPPING TABLES FOR THE ADVANCED ENCRYPTION STANDARD

TEJAS S. VYAS

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

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

DECEMBER 2004

Your file Votre référence
ISBN: 0-494-04455-1
Our file Notre référence
ISBN: 0-494-04455-1

Abstract

A Differential Fault Attack, Key Representation and Mapping Tables for the Advanced Encryption Standard

Tejas S. Vyas

In 1997, the National Institute of Standards and Technology (NIST) announced an open competition for an Advanced Encryption Standard (AES), a symmetric-key cryptographic algorithm, to replace the ageing DES and to be able to run on 128-bit data blocks, using 128-bit, 192-bit or 256-bit keys. After five years of intense peer review, attacks and discussions, the Rijndael algorithm was declared the AES. Being used by the entire world for most of the encryption, it has to be subjected to every possible analysis to find new vulnerabilities.

This thesis discusses three new features of the cipher. A powerful attack is initially formulated, which exploits cipher properties in its final and penultimate rounds under applications of precise bit/byte tweaking mechanisms. Next, the Key Schedule of the cipher is simplistically represented and a few interesting properties are noted. Finally, the AES is represented as a set of simple mapping tables and the relevance of such a representation is addressed.

Acknowledgments

I would begin by dedicating this thesis to my 'Pops', Sanjeev, and 'Mom', Padma, who someday are likely to get a Nobel Prize for being the Best Parents. Also, a special mention of my brother, Daish, whose avid passion and dedication in music inspired and evoked similar feelings in me for research.

A big thanks to George W. Bush, whose policies made me choose Canada as a prospective higher education destination. Turns out that, it was a blessing in disguise.

My solitary student life was enlivened by the presence of Anup - the Red and his poor jokes, Gurudath - the Kid and his gossips, Israat - the Coffeemaker and her everyday problems, Kruthi - the Cookie and her sportiness, Ram - the Confused and his gamer talks and Sabeel - the Burp and his procrastinations. Each page of this thesis exudes the fragrance of their moral support and contribution and there are no words to explain its worth.

A huge thanks to my present and previous roomies - Aniket, Rohan, Aman and Anand, with whose help the 'apartment' was revamped to a nice 'home'.

A huge credit goes to my supervisor, Dr. Harutyunyan, who other than being a dedicated, intelligent and a passionate researcher, is one of the nicest human beings I have known.

Finally, a big thank-you to Montreal, the second greatest city on planet Earth, after Bombay. Je me souviens...

Tejas

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptography, derived from the Greek words *kryptôs* meaning "hidden" and *gráphein* meaning "to write", is traditionally an art or a science to convert a normal message into an unintelligible form, so that only a recipient with the shared secret knowledge is able to read the original message. The process of rendering such a message (plaintext) incomprehensible is called encyption, while the reverse process of extracting the original message from its unreadable form (ciphertext) is called decryption. The shared secret knowledge is generally called as the key. Using the same key to encrypt and decrypt messages gives us a symmetric cryptosystem, whereas using different keys gives us an asymmetric or a public key cryptosystem. Trying to obtain the meaning of any encrypted information without the knowledge of the key is called cryptanalysis.

Cryptography is an interdisciplinary science involving linguistics, information theory, number theory, computational complexity, statistics, finite mathematics and in this modern era, even quantum physics.

1.1 Advent of the Advanced Encryption Standard

Dating as far back as 1900 BC [Kah96], cryptography has come a long way. Its importance was exemplified during the World Wars, when monumental efforts in code making and code breaking influenced the outcome of the war.

In the mid-70s, the United States under the auspices of the National Security Agency (NSA) and the National Bureau of Standards (NBS, renamed later as National Institute of

Standards and Technology or NIST) recognized the need to provide a standard for protecting unclassified and sensitive commercial data. This led to a call, requesting a proposal for an algorithm to encrypt 64-bit blocks of data. IBM responded with Lucifer, which with a few modifications, was approved as the Data Encryption Standard (DES) in 1977 [SB91].

DES is a sixteen round Feistel block cipher, which means that a block of input data is operated upon by a repeated application of similar transformations, each of which is called a round function. It uses a 56-bit secret key, beginning with a permutation P to distribute the data followed by the 16 rounds and ending with a reverse permutation P^{-1}. A single round involves splitting the 64 bits into two parts and subjecting one of the 32 bit parts to a Feistel function. The function will further involve an expansion E from 32 bits to 48 bits, a key EXOR with 48 selected bits out of the available 56 key bits, a substitution using eight parallel 6x4 bit S-boxes and a permutation P to reorder the bits. For a detailed account, see [Nat77].

DES lasted for two decades, despite being under intense cryptanalysis. A few shortcomings were discovered and are listed below.

- The *complementation property* (for every secret key K and an encryption $E_K(P)$, if $E_K(P) = C$: $E_{\overline{K}}(\overline{P}) = \overline{C}$) [HMS+76],
- The *four weak keys property* (for a secret key K: $E_K(E_K(m)) = m$) and
- The *six semi weak keys property* (for secret key pairs K_1 and K_2: $E_K(E_K(m)) = m$) [MS87]

They did not present any serious weaknesses. However, in the early 90s, three theoretical attacks undermined its strength: *Differential cryptanalysis* [BS93], *Linear cryptanalysis* [Mat93] and the *improved Davies' attack* [DM95] [BB97]. The 2^{47}, 2^{43} and the 2^{50} encryptions required for cryptanalysing DES differentially, linearly and through the Davies' attack respectively made it impractical then; however a *brute force* attack through an exhaustive 56-bit key search was always a future possibility. In 1977, a US$20 million key search machine design was proposed by Diffie and Hellman [DH77] which could cryptanalyse DES in a day. Wiener proposed a 7 hour cryptanalysis using a US$1 million similar Application Specific Integrated Circuit (ASIC) design machine in 1993 [Wie93]. Finally, DES became vulnerable in 1998, when the custom-made US$250,000 EFF DES cracker [EFF98] brute-forced a key in 56 hours. The time was brought down to twenty-two

hours when the cracker collaborated with 100,000 networked PCs. DES had to be upgraded or replaced.

On September 27, 1997, the National Institute of Standards and Technology (NIST) announced a competition for an Advanced Encryption Standard (AES) to replace the DES and to be able to run on 128-bit data blocks, using 128-bit, 192-bit or 256-bit keys. Being an open process, researchers from 12 different countries worked on developing an advanced encoding method, which also meant that their proposals would be reviewed and attacked by the worldwide cryptographic community. Fifteen algorithms met the requirements, which after a lot of peer review, attacks and worldwide discussions resulted in narrowing it down to a final five: Mars, RC6, Rijndael, Serpent and Twofish. Selection depended a lot on factors like security, speed and versatility.

In October 2000, NIST declared Rijndael, developed by two Belgian cryptographers John Daemen of Proton World International and Vincent Rijmen of Katholieke Universiteit Leuven, as the best overall algorithm for the AES based on the algorithm's combination of security, performance, efficiency, implementability and flexibility. Finally, after another year of evaluation, the U.S. Department of Commerce officially declared Rijndael the Advanced Encryption Standard. The National Security Agency also endorsed Rijndael and the public evaluation effort led to widespread acceptance. Importantly, Rijndael or any of its implementations is not subject to any patents. For a detailed account of the AES process, see [NIS01].

1.2 AES Algorithm

Rijndael is an iterated block cipher and its round transformation differs from the generally used Feistel structure. It has a variable block length and a variable key length, either of which can be independently specified to 128 bits, 192 bits or 256 bits. However, the primary difference between Rijndael and the Advanced Encryption Standard (AES) is the AES's mandatory 128-bit block length irrespective of the 128-bits, 192-bits or the 256-bit key lengths.

The following section explains the cipher structure of the AES. It has been derived from the second version of the original Rijndael documentation [DR99] submitted for the AES competition. For a Federal Information Processing Standards (FIPS) specification of the same, see [AES01]. The naming convention used is similar to the one used for Rijndael.

1.2.1 State and the Cipher Key Representation

An intermediate cipher result, termed as the *State*, can be represented as a rectangular array of bytes. Each array consists of four rows and four columns for a 128 bit key.

Similarly, the cipher key can be represented as an array of four row bytes, but with N_k number of columns depending on the size of the key (N_k = keysize/32). Thus, for a 128-bit, 192-bit or a 256-bit key, N_k will be 4, 6 or 8 respectively.

A two dimensional representation of both the State and the cipher key is given below. Both the state and the cipher key can be also thought of as one-dimensional arrays of 4-byte vectors or words. Thus, the order to map State bytes onto the array is $a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, a_{2,1}, a_{3,1}, a_{0,2}...$, while the order to map cipher key bytes onto the array is $k_{0,0}, k_{1,0}, k_{2,0}, k_{3,0}, k_{0,1}, k_{1,1}, k_{2,1}, k_{3,1}, k_{0,2}...$

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

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

Figure 1: The State and the Cipher Key($N_k = 4$) layout

In the end, the ciphertext is extracted from the state bytes in the same order. Naturally, the same holds true during the decryption process.

1.2.2 Number of Rounds

The length of the cipher key determines the number of rounds, N_r that the AES will undergo. For a 128-bit key ($N_k = 4$), $N_r = 10$. In contrast, $N_r = 12$ and $N_r = 14$ for key sizes of 192-bits ($N_k = 6$) and 256-bits ($N_r = 8$) respectively.

1.2.3 Round Transformation

The round transformation is basically composed of four transformations: the ByteSub[1], the ShiftRow[2], the MixColumn[3] and the AddRoundKey.

[1] SubBytes in the AES specifi cation
[2] ShiftRows in the AES specifi cation
[3] MixColumns in the AES specifi cation

1. **ByteSub Transformation**

In this case, each of the State bytes are operated upon independently using an S-box (Substitution Box), resulting in a non-linear byte substitution. The S-Box is a 256-byte table, wherein the value of the upper and the lower nibble of an input byte determines the value of the output byte. It is basically a composition of two transformations:

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

A field consists of a set of elements, which when combined by either addition or multiplication, produces other elements of the same set. A Galois field (GF) is one such finite field of p^n elements, where p is a prime [Cam03].

A polynomial in the Galois field GF(2^8) for a byte b consisting of bits $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$, is represented as $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x$. Polynomial addition in GF(2^8) corresponds to a simple bit-wise EXOR ($\oplus$), while polynomial multiplication in GF(2^8) corresponds to the multiplication of polynomials modulo an irreducible binary polynomial. An irreducible polynomial for the AES is $m(x) = x^8 + x^4 + x^3 + x + 1$. This ensures that any multiplication result will be a binary polynomial of degree less than 8.

By the extended Euclid algorithm, for any binary polynomial b(x) of degree below 8, one can compute polynomials a(x) and c(x) such that

$$b(x)a(x) + m(x)c(x) = 1$$

This deduces $a(x).b(x)\ mod\ m(x) = 1$ or $b^{-1}(x) = a(x)\ mod\ m(x)$. The multiplicative inverse in GF(2^8) is thus calculated.

(b) Applying an affine transformation over GF(2), defined by:

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

The only exception is that '00' is mapped into itself.

An inverse S-Box table is needed for decryption and that is obtained by taking the inverse of the affine mapping and then taking its multiplicative inverse in GF(2^8)

2. **ShiftRow Transformation**

The ShiftRow transformation involves cyclically shifting the rows of the State with different offsets. Row 0 remains as it is, while rows 1, 2 and 3 left shift over C1, C2 and C3 bytes respectively. Typically, for the AES, $C1 = 1, C2 = 2$ and $C3 = 3$[1]. Mathematically, a byte in row i at position j, $a_{i,j}$ is shifted to $a_{i,(j+4-C_i)mod4}$

The inverse ShiftRow is a cyclic shift of the row bytes by the same offsets in the opposite direction.

3. **MixColumn Transformation**

Here, the columns of the State, as vector bytes or words, are considered as polynomials with coefficients over GF(2^8). Polynomial coefficient addition in GF(2^8) corresponds to a simple bitwise EXOR of two vectors. However, polynomial coefficient multiplication in GF(2^8) requires reducing the product modulo a polynomial of degree 4, so that the result can have a degree less than 4 and be represented as a 4-byte vector. For the AES, one such polynomial is $M(x) = x^4 + 1$.

Thus, the modular product of two polynomials, c(x) and a(x) over GF(2^8): $c(x) = c_3x^3 + c_2x^2 + c_1x^1 + c_0x$ and $a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0x$ is

$$b(x) = c(x) \otimes a(x)$$

The modular product uses M(x) and can be represented in the matrix form as

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The output of a MixColumn transformation is b(x), when individual columns of the State, considered as a polynomial a(x), is multiplied modulo $x^4 + 1$ with a fixed polynomial:

[1]Different for Rijndael, because of different block sizes N_b possible

$$c(x) = \text{'03'}x^3 + \text{'01'}x^2 + \text{'01'}x + \text{'02'}$$

This polynomial is invertible and co-prime to $x^4 + 1$. 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 of MixColumn involves finding the inverse of c(x); that is to find d(x) given $c(x) \otimes d(x) = \text{'01'}$ and applying that to every column. It is calculated as

$$d(x) = \text{'0B'}x^3 + \text{'0D'}x^2 + \text{'09'}x + \text{'0E'}$$

4. **AddRoundKey Transformation**

The AddRoundKey transformation involves a simple bitwise EXOR of the State bytes with the Round Key bytes. The Round Key is derived from the original Cipher Key by a key schedule and its length is equal to the block length.

AddRoundKey is its own inverse.

A pseudo C notation for $(N_r - 1)$ round transformations is given below:

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

For the final round, the MixColumn step is removed. A pseudo C notation for the final round transformations is given below:

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

Figure 2 illustrates the changes affecting the State in a single round transformation. The same is run continuously $N_r - 1$ times.

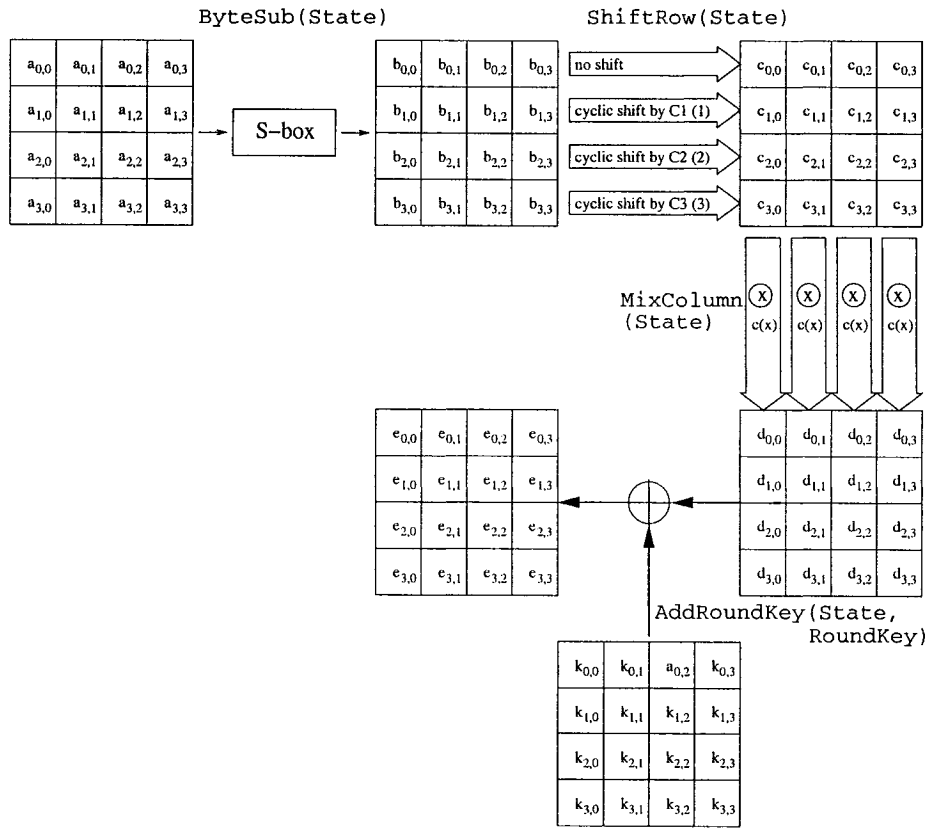

Figure 2: The Round Transformation

1.2.4 Key Schedule

The Round Keys are derived from the Cipher Key by means of the Key schedule. It has the following components: Key Expansion and Round Key Selection.

1. **Key Expansion**

This stage involves expanding the Cipher Key to a length, equal to the block size multiplied by number of rounds plus one. Thus for the AES 128-bit block, the Expanded key will be of 1408 bits (128 x (10 + 1)). Considering it as a linear array of 4-byte words, it can be denoted as $[W_i]$, with i in the range $0 \leq i < N_b(N_r + 1)$, where $N_b = 4$ for the AES.

A pseudo C notation for the Key Expansion process is given.

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)], Nk)
{
    word temp
    for(i = 0; i < Nk; 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)) ∧ Rcon[i / Nk];
        W[i] = W[i - Nk] ∧ temp;
    }
}
```

In the above description, *Rotbyte(W)* is a function which cyclically permutes an input word (a_0, a_1, a_2, a_3) to return the output word (a_1, a_2, a_3, a_0). The function *SubByte(W)* returns a word, where each byte is the result of applying the S-Box to the corresponding input byte of the word. The round constant word array *RCon(i)* is defined by $[x^{i-1}$,'00','00','00'], with x^{i-1} being powers of x in the field GF(2^8). This means that,

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

RCon(i) = [x,'00','00','00'] . RCon(i-1)

The first N_k words make up the Cipher Key. Every following word *W[i]* is the EXOR of *W[i-1]* and *W[i-N_k]*; the EXOR of the previous word and the word N_k positions earlier. This rule applies to all the words, except that a transformation

is performed on *W[i-1]* for words in positions *i* which are multiples of N_k. The transformation is in the form of a cyclic shift of bytes of the word (*RotByte*), followed by a non-linear substitution of the bytes of the word (*SubByte*). This transformation is then EXORed with *W[i-N_k]* and with a round constant (*RCon*).

It is worthwhile to note that, for a 256-bit key (N_k = 8), if i - 4 is a multiple of N_k, then *SubByte()* is applied to *W[i-1]* prior to the EXOR.

2. **Round Key selection**

The Round Key *i* is derived from the Expanded Key words **W[Nb*i]** to **W[Nb*(i+1)]**.

Figure 3 illustrates the calculation of W_4 and W_5 during Key Expansion. Also the Round Key selection for N_k = 4 is depicted.

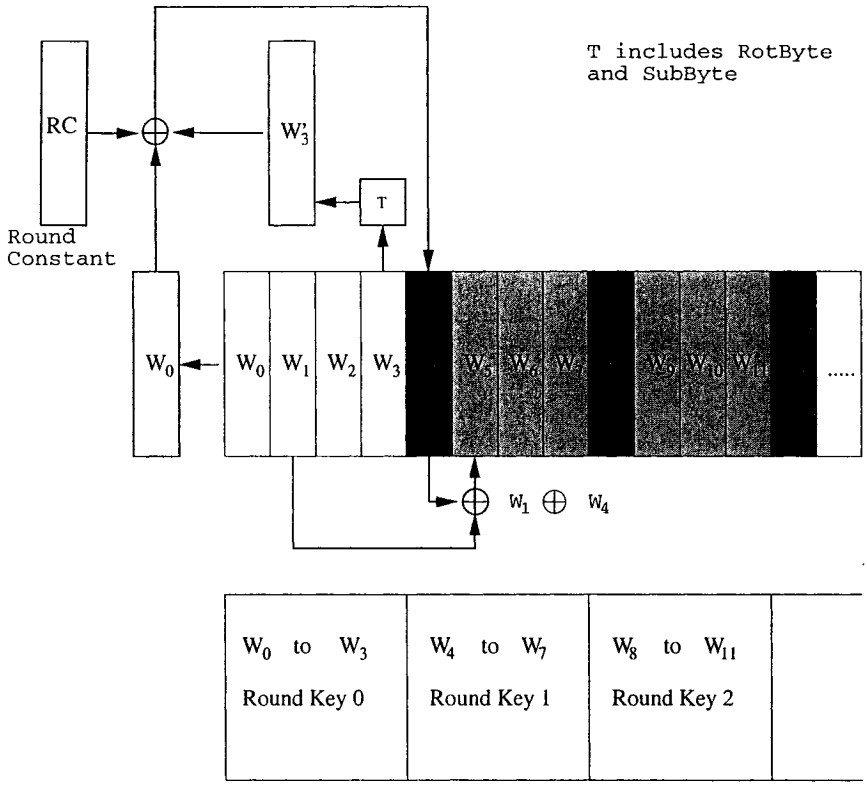

Figure 3: The Key Expansion with the Round Key selection

1.2.5 The Cipher

The AES cipher is thus composed of an initial RoundKey addition, $N_r - 1$ rounds and a final round composed of the ByteSub, the Shiftrow and the AddRoundKey transformations.

A pseudo C notation for 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);
}
```

1.2.6 The Inverse Cipher

The inverse round of the cipher for decryption in pseudo C notation can be similarly thought of as:

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

A pseudo C notation for the final round is given below:

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

The non-linear step (ByteSub) ends up being the last step in the inverse round, which makes its implementation tough with a table lookup [DR99]. Such aspects were anticipated

in the design and hence the structure is designed such that the sequence of transformations of its inverse is equal to that of that of the cipher itself. These transformations are marked by a replacment to their inverses and a change in the key schedule.

From the above inverse round description, one can safely swap the *InvShiftRow* and the *InvByteSub*, because their order does not make any difference to the structure. One is a transposition of bytes, while the other is a non-linear byte substitution.

Similarly, one can do the same for the *AddRoundKey* and the *InvMixColumn* based on the fact that for a linear transformation A, $A(x+k) = A(x) + A(k)$. The Round Key will thus be subjected to an *InvMixColumn* after its inverse expansion and then applied to the cipher under *AddRoundKey*.

The inverse algorithm in pseudo C notation:

```
InvAES(State,CipherKey)
{
    InvKeyExpansion(CipherKey,InvExpandedKey);
    AddRoundKey(State,InvExpandedKey + Nb*Nr);
    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<Nr; i++)
        InvMixColumn(InvExpandedKey + Nb*i);
}
```

1.3 Thesis Contributions

The thesis primarily endeavors to contribute on four different aspects of the AES.

An effort is initially made to assimilate the entire spectrum of research work performed on the AES till date and is explained briefly with appropriate references.

Next, an attempt is made to cryptanalyse the AES, by exploiting the attributes of the cipher in the final round. This very powerful attack requires precise single bit/byte fault

mechanisms. The attack is demonstrated to work if extended by a complete round (penultimate round).

There has been limited research work on the generation of the Key Schedule in both the forward and the inverse cipher. The thesis looks at an approach to represent the same using simple terms which helps derive interesting observations and properties.

Finally, a radical approach is taken in an attempt to represent the AES through simple mapping tables. Using differential analysis, a trail is followed and a concrete way of mapping hexadecimal values is conceptualized.

1.4 Thesis Organization

The thesis is organized in the following order:

- The latest/state-of-the-art literature review of the AES is presented in Chapter 2.
- Chapter 3 explores the Last Round Differential Attack which is similar to the Differential Fault Analysis (DFA) attack and requires precise bit/byte fault mechanisms. An extension to the same is investigated.
- Chapter 4 represents the Round Keys and a few interesting properties are noted.
- Chapter 5 attempts to represent the AES through simple mapping tables.
- The final chapter lists the conclusions derived and suggests a direction for future research.
- A thorough list of bibliographic references is presented next.
- An appendix is added to assist the analysis seen in Chapter 5 by using a test case.

Chapter 2

Literature review of the AES

Being subjected to a review by the worldwide community and subsequently being crowned as a FIPS cryptographic standard, a lot has been written and researched about the AES. This chapter essentially deals with all the work done in the "making" and in the (yet) unsuccessful "breaking" of the AES.

2.1 The Design Rationale of the AES

The authors of the AES designed it primarily with three purposes in mind [DR99]:

1. Resistance against all known attacks.
2. Speed and code compactness on a wide range of platforms.
3. Design simplicity.

A design method, known as the Wide Trail Strategy [Dae95], was developed to satisfy these parameters, primarily to provide resistance against the known nemeses - linear and differential cryptanalysis. In this method, a round transformation should be composed of three distinct invertible uniform "layers":

Linear mixing layer (λ) **:** provides high diffusion[1] over multiple rounds.

Non-linear layer (γ) **:** parallel application of optimum worst case nonlinear S-boxes.

Key addition layer ($\sigma[k^{(r)}]$) **:** Simple Exor of the Round Key to the intermediate State.

A fuller discussion of the wide trail strategy appears in [RD02]. Depending on this strategy, we now consider the motivation for such design choices:

[1] Defi ned by Shannon as a property: Each bit of the input should influence many bits of the output

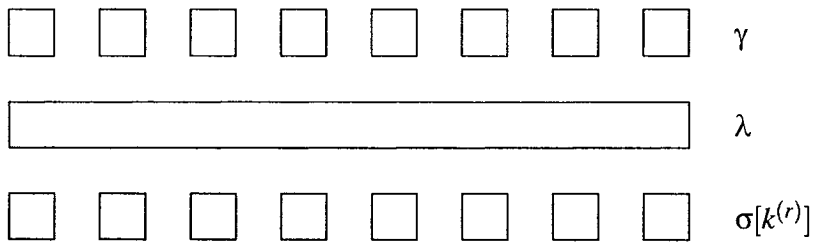

Figure 4: The Wide Trail Strategy - Bricklaying

2.1.1 Choice of the reduction polynomial m(x)

For the multiplication in GF(2^8), the polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ is the first one on the list of irreducible polynomials of degree 8, given in [LN86].

2.1.2 Choice of the S-Box

After the linear and the differential cryptanalytic attacks on the DES, it was necessary to develop an S-box, which can thwart those attacks. Also, algebraic manipulative attacks like the interpolation attack [JK97] had to be prevented. For that, it was imperative that the following criteria had to be kept in mind:

1. Invertibility.

2. Minimisation of the maximum correlation between linear combinations of input and output bits. Correlation is a bivariate measure of association (strength) of the relationship between two variables and can be expressed by a *correlation coefficient*. A minimal value for it denotes the possibility of a random or a non-linear relationship and according to [Nyb94], maximum input/output correlation can be made as low as 2^{-3} for invertible S-boxes.

3. Minimisation of the maximum probability of any differential trail. The probability of a differential trial is defined to be the probability that a pair with input difference α_1 propagates to the intermediate difference α_{i+1} after the i^{th} round for i = 1,...,r. According to [Nyb94], for invertible S-boxes, the maximum probability can be as low as 2^{-6}. This will enable small input changes to produce large output changes and vice versa.

4. Complexity of its algebraic expression in GF(2^8).

5. Simplicity of description.

Among the few classes of non-linear substitution boxes discussed in [Nyb94], the S-box defined by the mapping $x \Rightarrow x^{-1}$ was chosen in GF(2^8). This was primarily on the basis of the properties of x^{-1} such as high non-linearity, high non-linear order, resistance against differential attacks and efficient construction and computability. Such an S-box is made up of a very simple algebraic expression and basically does not satisfy the fourth criterion listed above. To add sufficient algebraic complexity, an invertible simple affine transformation is combined with this "inverse" mapping. Thus, if $a(x)$ is the output of the inverse mapping $x \Rightarrow x^{-1}$ reduced modulo $m(x)$, then the affine is of the type:

$$b(x) = (x^7 + x^6 + x^2 + x) + a(x)(x^7 + x^6 + x^5 + x^4 + 1) mod (x^8 + 1)$$

The modulus $(x^8 + 1)$ was chosen because it was the simplest possible. The multiplication polynomial $(x^7 + x^6 + x^5 + x^4 + 1)$ was the simplest of all polynomials co-prime to the modulus. The constant $(x^7 + x^6 + x^2 + x)$ was chosen to avoid fixed points ($Sbox(a) = a$) and 'opposite' fixed points ($Sbox(a) = \overline{a}$).

2.1.3 Choice of the ShiftRow offsets

The ShiftRow is an important linear step to allow diffusion and its choice of offset was based on the following criteria:

1. The four offsets are different and the first row offset $C0 = 0$.
2. Resistance to attacks using truncated differentials [Knu95] (discussed later).
3. Resistance to the Square attack [DKR97] (discussed later).
4. Simplicity.

The simplest combination of all the possible ones, satisfying the above criteria was selected.

2.1.4 Choice of the MixColumn transformation

The design criteria for such a 4-byte to 4-byte linear transformation was:

1. Invertibility.

2. Linearity in GF(2^8).

3. Symmetry.

4. Simplicity.

5. Speed on 8-bit processors.

6. Relevant diffusion power.

Criteria 2, 3 and 4 allow us to use simple modulus for polynomial coefficient multiplication given by x^4+1. Criteria 5 imposes that for a higher speed and a quicker computability, the coefficients should be small. '01', '02' and '03' are simple coefficients, which when multiplied, require no processing, a single shift and a simple EXOR with the shift respectively.

For the final criteria, one has to understand what diffusion power means. Consider M as the linear Mixcolumn transformation matrix acting on byte vectors, and $W(a)$ be the byte weight of the vector, which is the number of non-zero bytes in the vector. It is intuitive that a cryptanalyst, through linear and differential trails, would benefit from a multiplication polynomial, which reduces the number of non-zero bytes in input and output difference polynomials [DKR97]. This is what we are trying to avoid by selecting a polynomial with high diffusion power expressed by the *branch number*. The branch number of a linear transformation is

$$min_{a\neq 0}(W(a)+W(M^T.a))$$

A non-zero byte is called an *active byte*. The worst-case diffusion occurs with a single active byte, making it easier for cryptanalysis. To avoid it, the upper bound for branch numbers should be $n+1$, where n is the number of bytes within the word. Thus, for the MixColumn, if a state is applied with a single active byte, the output can have at most 4 active bytes. Hence, the upper bound for the branch number is 5. The coefficients were chosen such that the upper bound is reached.

2.1.5 Choice of the Key Expansion

The following design criteria was considered for the key expansion algorithm:

1. Invertibility. Knowledge of N_k consecutive words should be enough to generate the whole Expanded Key.

2. Use of appropriate round constants to eliminate symmetry. In the absence of such round dependent constants, symmetry exists in the round transformation. Similar treatment of State bytes and a similar round transformation applied to all the rounds deems it necessary to introduce some asymmetry.

3. Diffusion of Cipher Key differences into the Round Keys. This is mainly to resist related key attacks [Bih93] [KSW96] (discussed later), whereby two different Cipher Keys, bound by a relation, might have a large set of Round Key bits in common.

4. Enough non-linearity to avoid determination of Round Key differences from Cipher Key differences. The application of ByteSub ensures a high level of non-linearity.

5. No possibility in knowing many Round Key bits based on the knowledge of a part of the Cipher Key or a few Round Key bits.

6. Speed on 8-bit processors.

7. Simplicity.

A light weight, byte oriented expansion scheme with the use of appropriate round constants and ByteSub was thus chosen satisfying all the above constraints.

2.1.6 Choice of the number of rounds

If an attack's expected workload is below that required for an exhaustive key search, it is termed as a shortcut attack. For a block length and a key length of 128 bits, no shortcut attacks have been found for AES with rounds greater than 6. Therefore, 4 more rounds have been added as a security measure. Such an addition is based on strengthening AES's high diffusion round and thwarting the propagation trails, which are needed for linear, differential and truncated differential cryptanalytic attacks.

Also noteworthy is the fact that, for higher key lengths, the number of rounds is raised by one for every increment of 32 bits in the Cipher Key.

2.2 Expected Strength of the AES

The most efficient key-recovery attack for the AES is the exhaustive key search. Available information from given plaintext-ciphertext pairs will yield other plaintext-ciphertext pairs

only by an exhaustive key search and by no other efficient means. The effort required to do so is:

For a 128 bit key$\Rightarrow$ 2^{127} applications of the AES.
For a 192 bit key$\Rightarrow$ 2^{191} applications of the AES.
For a 256 bit key$\Rightarrow$ 2^{255} applications of the AES.

2.3 Strength against known attacks

This section deals with the effectiveness of the AES at the time of its design.

2.3.1 Symmetry and weak keys property of the DES type

The use of round constants in the key schedule thoroughly eliminates any symmetric properties that might exist. Also, the usage of different components for the cipher and its inverse eliminates any existing weak keys similar to that in the DES. The non-linearity in the key schedule also eliminates any *equivalent keys*; a pair of keys giving the same encryption.

2.3.2 Linear and Differential cryptanalysis

Chapter 5 of [Dae95] explains in depth the basic formalisms of differential cryptanalysis (DC) and linear cryptanalysis (LC) and how the "wide trail strategy" of the AES tackles both. The following briefly explains the formalisms.

Linear Cryptanalysis

A *linear trail* is specified by a series of selection patterns, whereas the *correlation coefficient* $C(f,g)$ is associated with a pair of Boolean functions with domain $\{0,1\}^n$, $f(a)$ and $g(a)$ given by $C(f,g) = 2.Prob(f(a) = g(a)) - 1$. A non-zero value means that the functions are correlated. Now, the *correlation coefficient over a linear trail* (positive or negative) is given by the product of correlation coefficients between the linear combinations of input bits and output bits of every subsequent round. An input-output *correlation* between a linear combination of input bits, denoted by selection pattern u and a linear combination of output bits, denoted by selection pattern v is then equal to the sum of the correlation coefficients of all linear trails starting with u and ending in v. LC attacks are possible, if the predictable input-output correlation is significantly larger than $2^{n/2}$ over most of the

rounds, where n is the block length. If there are no linear trails with a correlation coefficient higher than $2^{n/2}$, then one can be resistant against LC.

For the AES, it was proved that there are no 4-round linear trails with a correlation above 2^{-75}, which is sufficent.

Differential Cryptanalysis

A *differential trail* is specified by a series of difference patterns, where the *prop ratio* is the relative amount of all input pairs that for the given input difference, give rise to the output difference. The *difference propagation over a differential trail* is given by the product of difference propagations between the input bits and output bits of every subsequent round. A *difference propagation* between an input difference pattern, denoted by a' and an output difference pattern, denoted by b' is then composed of differential trails, where its *prop ratio* is equal to the sum of the prop ratios of all differential trails starting with a' and ending with b'. DC attacks are possible, if the predictable difference propagations have a prop ratio significantly larger than 2^{1-n} over most of the rounds, where n is the block length. If there are no differential trails with a predicted prop ratio higher than 2^{1-n}, then one can be resistant against DC.

For the AES, it was proved that there are no 4-round differential trails with a predicted prop ratio above 2^{-150}, which is sufficient.

2.3.3 Truncated differentials

First noticed by Knudsen [Knu95], this attack exploits ciphers in which all transformations operate on the State in well aligned blocks or bits. When differential trails tend to *cluster* for certain sets of input difference and output difference patterns, there is a huge number of differential trails. AES uses byte transformations and no shortcut attacks were found.

2.3.4 Interpolation attacks

Jakobsen and Knudsen constructed polynomials using input/output pairs and then formed a compact algebraic expression solvable with manageable complexity [JK97]. The expression for the AES S-box was subsequently computed as

$$63 + 8fx^{127} + b5x^{191} + 01x^{223} + f4x^{239} + 25x^{247} + f9x^{251} + 09x^{253} + 05x^{254}$$

However in the AES, the S-box in GF(2^8) along with the diffusion layer prohibits this type of attack for more than a few rounds.

2.3.5 Related-key attacks

In [Bih93] and [KSW96] demonstrated that several ciphers were prone to have related key weaknesses. Using different or partly unknown keys with a chosen relation, a cryptanalyst can do cipher operations. The usage of non-linear S-boxes and good diffusive properties in the AES key schedule does not make it vulnerable against such attacks.

2.3.6 The Square Attack

The AES inherits many properties from the SQUARE cipher, an early predecessor by the same authors, Vincent Rijmen and John Daemen. The square attack is a dedicated chosen plaintext attack on the SQUARE cipher and hence, it is also applicable to AES because of their byte-oriented structures. Although, it is a basic 4 round attack but it can also be extended to attack reduced AES versions up to 6 rounds. The attack is explained in-depth in [DKR97] and the underlying concept is briefly explained here.

Let $\wedge$ be a set of 256 states that are all different in specific bytes and are the same in the remaining State bytes; call these *active* and *passive* bytes respectively. Applying the round transformation on elements of such a set, one can observe how such mapping affects its active and passive bytes. The ByteSub and the AddRoundKey transformations do not affect the $\wedge$-set because they are byte substitutions and additions. Therefore, they leave the positions of the active and the passive bytes unchanged. The ShiftRow transposition changes the positions of the active and passive bytes, but does not change their number. However, the MixColumn transformation affects the number of active and passive bytes. An input column with a single active byte can trigger an output column with all four bytes active.

Let λ index the active bytes. Then

$$\forall x, y \in \wedge : \begin{cases} x_{i,j} \neq y_{i,j} & \text{for } (i,j) \in \lambda, \\ x_{i,j} = y_{i,j} & \text{for } (i,j) \notin \lambda. \end{cases}$$

Consider a $\wedge$-set having only one active byte. This byte will cycle through all the 256 values. Let us trace the changes in the positions of the active bytes for 3 rounds -

$\rightarrow$ MixColumn of 1^{st} round transforms the active byte to a complete column of active bytes.

$\rightarrow$ Shiftrow of 2^{nd} round transposes the four active bytes to all the columns.

$\rightarrow$ MixColumn of 2^{nd} round converts this to all 4 columns of active bytes.

$\rightarrow$ This stays a $\wedge$-set until the input of MixColumn of the 3^{rd} round.

Bytes of this $\wedge$-set 'a' range over all possible values and are therefore *balanced.*

$$\begin{aligned}
\bigoplus_{b=Mixcolumn(a),a\in\wedge} b_{i,j} &= \bigoplus_{a\in\wedge}(2a_{i,j}\oplus 3a_{i+1,j}\oplus a_{i+2,j}\oplus a_{i+3,j}) \\
&= 2\bigoplus_{a\in\wedge} a_{i,j}\oplus 3\bigoplus_{a\in\wedge} a_{i+1,j}\oplus \bigoplus_{a\in\wedge} a_{i+2,j}\oplus \bigoplus_{a\in\wedge} a_{i+3,j} \\
&= 0\oplus 0\oplus 0\oplus 0 \\
&= 0
\end{aligned}$$

At the input of the fourth round, all bytes are balanced though this is destroyed by the 4^{th} round ByteSub. Assuming, there is no MixColumn in this round, the output of the 4^{th} round will be:

$$a_{i,j} = SBox(b_{i,j})\oplus k_{i,j}$$

By guessing a value for $k_{i,j}$ with $a_{i,j}$ known, the value of $b_{i,j}$ for all elements of the $\wedge$-set can be calculated. Only 1 key byte value will satisfy the balance condition over $\wedge$ and this can be repeated for the other key bytes to get the whole key. This comprises the basic attack on 4 rounds.

In the 5-round attack, Knudsen added a round at the end and a value for four bytes of the fifth round key was guessed. A 5-round variant included the addition of a round at the beginning, after which four bytes of the first Round Key was assumed. A correct assumption followed by an application of the 4-round attack resulted in the recuperation of the last Round Key.

When both the 5-round variants were added together, a 6-round attack seemed feasible. The work factor and the memory requirements for each one of them is listed below.

Attack	# Plaintexts	# Cipher executions	Memory
Basic(4 rounds)	2^9	2^9	small
End extension(5 rounds)	2^{11}	2^{40}	small
Beginning extension(5 rounds)	2^{32}	2^{40}	2^{32}
Both extensions(6 rounds)	2^{32}	2^{72}	2^{32}

Table 1: The Square Attack

2.4 New Literature on the AES

This section deals with all the literature by different cryptanalysts in an effort to determine weaknesses or flaws in the AES.

2.4.1 Extensions to the Square attack

In [FKL+00], the authors improved the best attack on a reduced 6 round AES from complexity 2^{72} to 2^{44}. The basic idea is to associate a "partial sum" to each ciphertext and a key. Under the original 6-round attack, one guesses four bytes of k^0, but here one uses instead 2^{32} plaintexts, whose encryptions consists of 2^{24} groups of 2^8 encryptions that vary in a single byte after the first round. One just guesses five key bytes at the end of the cipher, does a partial decrypt to a single byte up to the end of round 4, sums all the values over 2^{32} encryptions and checks for the balance (zero) condition. Less bits of the key are guessed, but 2^{24} more work is done for each guess.

When substantial plaintexts are available, this technique can also break 7 rounds for 128-bit keys, 8 rounds (out of 12) for 192-bit keys and 8 rounds (out of 14) for 256-bit keys. Most require the whole code book and therefore is impractical.

A co-author of the above paper, S. Lucks proposed the name "saturation attacks" for all such chosen plaintext attacks that exploit the byte-oriented structure of all types of ciphers, provided they have a round structure similar to the AES. In his paper [Luc00], the author improved the 7-round attack by a factor of 2^{16}, deduced similar results as above, but accepted the high security margin for the number of rounds of the cipher.

One such saturation attack was developed by Gilbert and Minier in [GM00]. Different from Square, it exploits the existence of collisions between some partial byte oriented functions induced by the AES cipher such that it provides a distinguisher between four inner rounds of AES and a random permutation. This in turn, enables mounting attacks on 7 rounds for any key length. Such an attack is faster over the exhaustive search over 256 key bits. However, they do not claim their attack to be practical because of the high computational complexity, despite improving previously known cryptanalysis results.

2.4.2 Properties of the linear diffusion layer

By considering the affine transformation in the S-box and the linear transposition (*ShiftRow*) and mixing (*MixColumn*) of a round, all together as part of a complete "linear diffusion

layer", Murphy and Robshaw in [MR00b] observed some unusual properties and questioned the high diffusion of the AES. They represented the action of such a modified linear diffusion layer as 128×128 binary matrix, M over GF(2). Its characteristic polynomial[1], $c(x) = (x+1)^{128} = x^{128} + 1$ and its minimal polynomial polynomial[2], $m(x) = (x+1)^{15}$ were argued to be too simple along with the fact that the degree of $m(x)$ was way too small. Furthermore, since $m(M) = 0$ and $(x^{16}+1) = (x+1) \times m(x)$, we have $M^{16} = I$. This means that any 128-bit input (or difference) to the AES linear diffusion layer is mapped to itself after at most 16 repeated applications of the linear diffusion transformation.

The designers of the AES, Daemen and Rijmen in response [DR00] argued that such unusual properties are no sign of any design weakness and that it does not make the cipher insecure. They conclude that, though finding properties of separate pieces of a round is a useful and important exercise, the round function as a whole is greater than the sum of its parts.

Nevertheless, the original duo again criticised the cipher [MR00a] and stuck by their conclusion. They contend that although the deduced properties might not offer any advantage to differential and linear cryptanalysis, it might offer a novel yet unknown way to combine the rich structure in the diffusion layer with the highly structured inverse map.

2.4.3 Algebraic Representation

Richard Schroeppel in [FSW01] cited concerns about the byte-oriented structure of the AES and that the only non-linear aspect of it was a simple x^{-1}. The AES was then expressed in a neat and compact algebraic formula.

Consider an S-box on the bytes $b_{i,j}$. Then for certain constants $\lambda_0, ..., \lambda_8$, its actions can be written as:

$$S(b_{i,j}) = \lambda_8 + \Sigma_{d=0}^{7} \lambda_d b_{i,j}^{255-2^d}$$

As discussed in the previous section, one can remove the constants in the S-box leaving just the non-linear part. Then the arithmetic in GF(2^8) is:

$$S(b_{i,j}) = \Sigma_{d=0}^{7} \lambda_d b_{i,j}^{-2^d}$$

Now, let $k_{i,j}^r$ be the key(128-bit) in the (i,j) position in round r with $\varepsilon = \{0,1,2,3\}$ and $D = \{0,...,7\}$. Then

[1]The characteristic polynomial of a $n \times n$ matrix M is the polynomial in x given by the formula $Det(M+xI)$

[2]The minimal polynomial of an $n \times n$ matrix M over a fi eld F is the monic polynomial m(x) over F of least degree such that m(A)=0

$$b_{i,j}^{r+1} = k_{i,j}^{(r)} + \Sigma_{e_r \in \varepsilon, d_r \in D} \lambda_{i,e_r,d_r} (b_{e_r,e_r+j}^{(r)})^{-2^{d_r}}$$

This can be written as:

$$b_{i,j}^{r+1} = k_{i,j}^{(r)} + \Sigma_{e_r \in \varepsilon, d_r \in D} \frac{\lambda_{i,e_r,d_r}}{(b_{e_r,e_r+j}^{(r)})^{2^{d_r}}}$$

This can be expressed for the r = 10 round AES. Each round swells up the number of terms. Say for the 3^{rd} round, we get the following expression:

$$b_{i,j}^{(3)} = k_{i,j}^{(2)} + \Sigma_{e_2 \in \varepsilon, d_2 \in D} \frac{\lambda_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)} + \Sigma_{e_1 \in \varepsilon, d_1 \in D} \frac{\lambda_{e_2,e_1,d_1}}{\left(b_{e_1,e_1+e_2+j}\right)^{2^{d_1}}} \right)^{2^{d_2}}}$$

Using Freshman's Dream theorem [Lau03], another way to put this is

$$b_{i,j}^{(3)} = k_{i,j}^{(2)} + \Sigma_{e_2 \in \varepsilon, d_2 \in D} \frac{\lambda_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)}\right)^{2^{d_2}} + \Sigma_{e_1 \in \varepsilon, d_1 \in D} \frac{\left(\lambda_{e_2,e_1,d_1}\right)^{2^{d_2}}}{\left(b_{e_1,e_1+e_2+j}\right)^{2^{d_1+d_2}}}}$$

Replacing expanded key bytes with K, constants with C and subscripts and exponents with asterisks, we get a sense of the formula. Also, one should understand that K^* and C_* represent different values. So, after 6 rounds, one gets:

$$b_{i,j}^{(6)} = \left(K^* + \Sigma_{e_5 \in \varepsilon, d_5 \in D} \frac{\mathbf{C_*}}{\left(K^* + \Sigma_{e_4 \in \varepsilon, d_4 \in D} \frac{C_*}{\left(K^* + \ldots\right)} \right)} \right)$$

After six rounds, there are 2^{35} terms of the form $C_*/(K^* + p_*^*)$. after ten rounds, there are 2^{50} such terms. Solving these efficiently could break the AES. But it is computationally infeasible right now to solve such equations and the security of the AES relies on a new computational hardness assumption.

2.4.4 Linear redundancy in the non-linear S-Box

For $n = 8$ inputs, the inverse mapping ($f(x) = x^{-1}$) of the AES S-Box satisfies several non-linearity criteria and is designed to be provably secure against linear and differential cryptanalysis. The same S-Box is used 160 times in the 10 round cipher and is provides the only non-linearity aspect to the AES.

Fuller and Millan in [FM02] made an observation that all output functions of an AES S-box can be mapped to each other using affine transformations putting them all in the same "equivalence class". They showed that the S-box is made up 8 Boolean functions with 8 common input bits, each exhibiting similar order, non-linearity, the absolute maximum auto-correlation function along with similar frequency distributions. Under a single equivalent class, a relation between two Boolean functions was found. If $f_i(x)$ and $f_j(x)$ are

two distinct outputs of the AES S-box, there exists a non-singular matrix $D_{i,j}$ and a binary constant $c_{i,j}$ such that $f_j(x) = f_i(D_{i,j}) \oplus c_{i,j}$. This implies that the AES can be described by means of one non-linear Boolean function and some linear/affine operations. It follows that this kind of redundancy is due to the inverse mapping in the finite field.

The designers of the AES contend that the observation is not a weakness, but an example of the simplicity in its design.

2.4.5 XSL attacks

Perhaps the most controversial work on the AES, Courtois and Pieprzyck in [CP02] observe that the S-box can be described by an overdefined[1] system of quadratic equations. Representing the 8 bits of the input as $x_1,...x_8$ and the 8 bits of the output as $y_1,...y_8$, degree 2 equations exist of the form:

$$f(x_1,...,x_8,y_1,...,y_8) = 0$$

By overdefining the equations, it is argued that the complexity is reduced by the extra equations. When this strategy of forming equations is used at every step of the cipher, that is at the input and output of every *State*, one ends up with a large system of 8000 quadratic equations with 1600 variables for the 128-bit AES key. A specialized algorithm called the XSL (eXtended Sparse Linearisation) attack is then applied to solve the equations and recover the key. The attack is noteworthy in its use of a handful of known plaintexts. It was also contemplated that the security of the AES does not grow exponentially with each round.

The paper attracted a huge amount of interest; most of them apprehensive of its working because of the highly technical and heuristic method used and its resulting complexity evaluation. Furthermore, the tremendous work-factor (2^{230} for AES 128-bit key) involved does not make it faster than an exhaustive key search. Nonetheless, cryptographers expressed unease at the algebraic simplicity of the AES.

In [MR02], Murphy and Robshaw discovered an alternative description of AES, embedding it in a larger cipher called "BES" (*Big Encryption System*), which can be described using very simple operations over a single field, GF(2^8). There is a map Φ such that

$$AES(x) = \Phi^{-1}(BES(\Phi(x)))$$

A key observation is that BES's round function can be written as

[1] more equations than unknowns

$$\mathbf{b} \mapsto \mathbf{M_B} \mathbf{b}^{-1} + k_{B_i}$$

where k_{B_i} is the i^{th} round key for the BES and $\mathbf{M_B}$ is the 128×128 linear diffusion matrix consisting of the row operation, column operation and the S-box linear operation of a single round. Thus, a round of AES is simply a componentwise inversion and an affine transformation with respect to the same field GF(2^8). Few properties of the BES were discovered, though none of them translate to a weakness in the properties of the AES.

It was further analysed that an XSL attack mounted on such a BES system would yield a simpler set of equations which could break AES with a complexity of around 2^{100}. This, only under the assumption that the Courtois and Pieprzyk analysis is correct and practical to use. No symbolic computation system can currently solve such sets of equations.

Chapter 3

Last Round Differential Attack / DFA Attack

The following chapter is an original contribution and explores a vulnerability in the last round of the AES cipher. The original attack is similar to the Differential Fault Analysis (DFA) attack in [Gir04], which was published in the AES4 Conference, May 2004 - at the same time, the idea was conceptualised by the author of this thesis. The first section discusses the basic attack, whereas the second section deals with the DFA and a variant of the same. The final section extends the basic attack by an extra round, with an obvious corresponding increase in computation complexity.

3.1 Last Round Differential Attack

The attack involves the last round of the AES cipher. It occurs at the possibility of a single bit toggle in any of the bits at the end of the 9^{th} round. The section shows how one can then obtain the last round key; in our case the AES 128 bit key. The following notations will be considered:

-$P_{i,j}$ for the intermediate State value at the end of the 9^{th} round for the i^{th} row and the j^{th} column.
-$K_{i,j}$ for the Round Key values of the 10^{th} round for the i^{th} row and the j^{th} column.
-$C_{i,j}$ for the Ciphertext values at the end of the 10^{th} round for the i^{th} row and the j^{th} column.

Figure 5 explains the structure of the final round with respect to the defined notations.

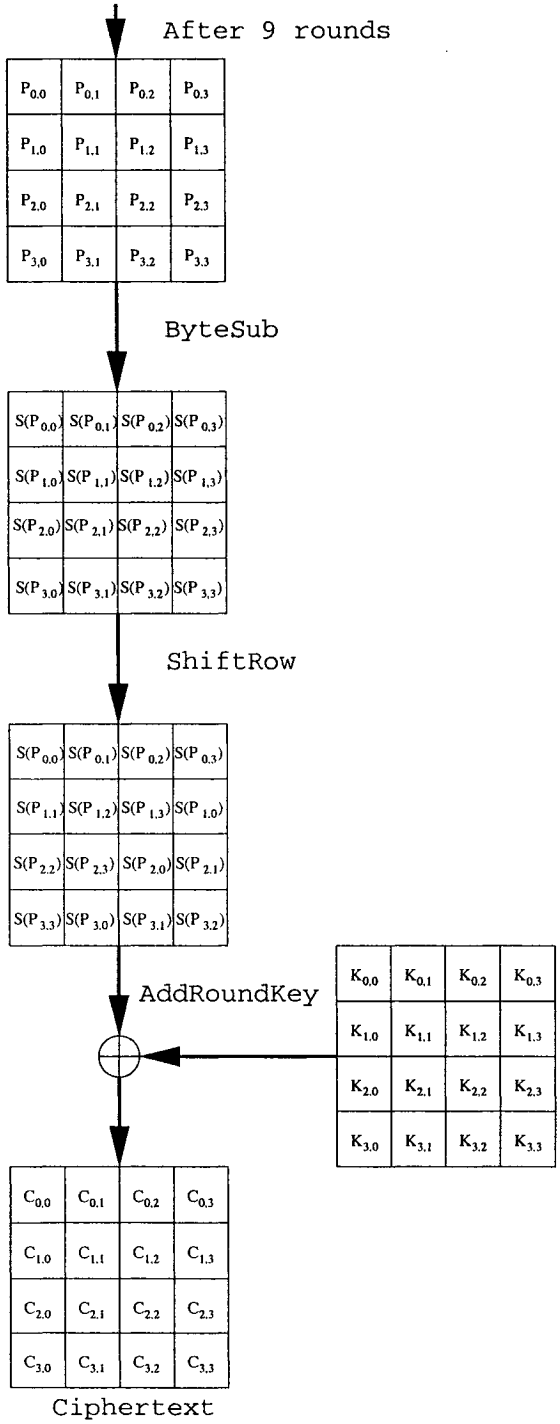

Figure 5: The Last Round Structure

The entire final round transformation can be written as

$$ShiftRow(ByteSub(P_{i,j})) \oplus K_{i,j} = C_{i,j}, \quad \forall x,y \in \{0...3\} \tag{1}$$

Consider the case, when a single bit out of the 128 bits in $P_{i,j}$ toggles. Noticeably, according to our representation, such a difference affects only a single byte in $P_{i,j}$. Let the new *changed* intermediate State byte value at the end of the 9^{th} round be defined as $P'_{i,j}$ for a distinct value of i and j. Furthermore, let $C'_{i,j}$ be the corresponding Ciphertext values generated at the end of the 10^{th} round.

It is now worthwhile to note the propagation of such a single bit difference at the start of the last round on the final ciphertext. The byte $P'_{i,j}$, yields to a different byte value after the ByteSub transformation. Next, the ShiftRow transformation changes the position but does not alter the value of this byte. An EXOR addition with the final Round Key gives a new value to this byte in $C'_{i,j}$, markedly different from the byte value at the same position in $C_{i,j}$. One can thus deduce that, there is only a single byte difference in the two ciphertexts. Mathematically, for the altered byte:

$$ShiftRow(ByteSub(P'_{i,j})) \oplus K_{i,j} = C'_{i,j} \tag{2}$$

And for the rest of the bytes:

$$ShiftRow(ByteSub(P_{i,j})) \oplus K_{i,j} = C'_{i,j} \tag{3}$$

From equations (1) and (3),

$$C_{i,j} \oplus C'_{i,j} = 0 \tag{4}$$

This shows that 15 bytes out of the possible 16 are unaffected by the bit change. At the same time, the difference (EXOR) between equations (1) and (2) gives

$$C_{i,j} \oplus C'_{i,j} = ShiftRow(ByteSub(P_{i,j})) \oplus ShiftRow(ByteSub(P'_{i,j})) \tag{5}$$

The unknowns in the above equation is $P_{i,j}$ and $P'_{i,j}$, for singular values of $i \in \{0...3\}$ and $j \in \{0...3\}$. These values of i and j can be realised by observing the position of the differential bytes in the two ciphertexts followed by an InvShiftRow.

The ShiftRow is applicable to both $P_{i,j}$ and $P'_{i,j}$ and in a way that does not affect the

value of the equation. The above equation can then be thought of as

$$C_{i,j} \oplus C'_{i,j} = ByteSub(P_{i,j}) \oplus ByteSub(P'_{i,j}) \tag{6}$$

For each value of $P_{i,j}$, there exist 8 possibilities in the value of $P'_{i,j}$, depending on the single bit that was toggled. Coupled with the possible 2^8 guesses of $P_{i,j}$, that satisfy the above equation, the total effort required is 8.2^8.

A pseudo C notation for the doing the same is listed:

```
FindP(C,C')
{
    byte Toggle[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
    byte P[16], P'[16];
    int counter = 0;
    For( i = 0; i < 8; i++ )
        For( j = 0; j < 256; j++ )
            If( ByteSub[j] ∧ ByteSub[j ∧ Toggle[i]] == C ∧ C')
            {
                P[counter] = j;
                P'[counter] = j ∧ Toggle[i];
                counter = counter +1
            }
}
```

In the above code, the array Toggle is used to deduce P' from a byte P. The If condition checks for the satisfaction of equation (6) for 256 values corresponding to a single Toggle array value. For example, when the Toggle array's value is $0x01$, it means $P \oplus P' = 0x01$ and for each value of i from 0 to 255, its corresponding value of $i \oplus 0x01$ is calculated and checked to see if the EXORs of their ByteSubs is equal to the differential Ciphertext byte value ($C \oplus C'$). The byte array P contains values that satisfy equation (6), while the byte array P' stores the corresponding toggled values. These values form a set of at most 16 because the S-box is made up of 16x16 unique values and for each toggled bit, a byte EXOR of such unique values yield two solutions satisfying a particular difference. The combined P and P' are in fact 8 swapped pairs of similar values. The knowledge of the toggled bit position makes matters simpler, because P will then evade going through all 8

bit toggle possibilities and will be left with just two possibilities.

By performing a similar toggle on another bit of the same byte $P_{i,j}$, a different value of $C \oplus C'$ is received and the same analysis can be performed to get 16 other possible values of P. This set will definitely have atleast one value similar to one in the set computed earlier. This effectively enables the calculation of $P_{i,j}$.

The same set of operations are performed for calculation of every intermediate State byte $P_{i,j}$, $\forall x, y \in \{0...3\}$, wherein every calculation requires toggling different bits in the same byte *atleast twice*. This means that for the calculation of those 16 bytes, a minimum of 32 bit-toggles are needed. Assuming that not more than 3 bit-toggles are needed for every byte, a maximum of 48 bit-toggles might be needed for calculation of $P_{i,j}$.

The work done to calculate $P_{i,j}$ leads us to evaluate the Last Round Key easily. From equation (1)

$$K_{i,j} = ShiftRow(ByteSub(P_{i,j})) \oplus C_{i,j}, \quad \forall x, y \in \{0...3\} \tag{7}$$

The knowledge of the final Round Key, in effect exposes the entire Expanded Key, helps one to break the cipher and leads to cryptanalysis.

3.2 DFA Attack

Implementing the powerful attack discussed in the previous section seems impractical. Ability to toogle an intermediate State value bit at the end of the 9^{th} round might look good on paper but may not be possible in practice.

However in the late 90s, a new cryptanalytic mechanism was developed by Boneh *et al.* in [BDL97], wherein a fault may be induced on smartcards during the computation of its crytographic algorithm through power glitches, clock pulses, laser radiations etc. Using the faulty ciphertexts, the cryptanalyst is able to extract the key. Such an attack was subsequently named by Biham and Shamir in [BS97] as *Differential Fault Analysis* or simply DFA.

Cryptographic smartcards use DFA to test their security and since most of them began to start using AES, it became necessary to check AES's vulnerability to DFA. Christophe Giraud in [Gir04] presented two fault models for a DFA attack in the AES4 conference in May,04. One of this is similar to the Last Round Differential Attack discussed in the previous section, which he contends, is more theoretical by nature. For inducing different

bit faults on the same byte twice and thrice, it was proved that the chance to obtain $P_{i,j}$ successfully is 50% and 97% respectively.

The other fault model is based on inducing a fault, which may change an entire byte of an intermediate State value or an intermediate Round Key. Contending to be more practical than the earlier bit-fault model, it requires upto 250 faulty ciphertexts (C') to be able to ascertain the 128 bit key of the cipher. A detailed account can be found in the same paper. Briefly, the attack is divided into 3 steps:

1. A fault is induced on one of the last column bytes of the 9^{th} Round Key - $K^9_{i,j}$, just before the computation of the final Round Key. Such a fault is carried over to this next Round Key $K^{10}_{i,j}$, affecting 5 bytes; one in each of the first 3 columns and two in the final column. Similar differences are ultimately reflected in the final faulty ciphertext. An analysis similar to the bit-fault DFA follows. By performing an average of 32 faults, the attack helps obtain the last 4 bytes of $K^9_{i,j}$.

2. A fault is induced on one of the last column bytes of the 8^{th} Round Key - $K^8_{i,j}$, just before the computation of the 9^{th} Round Key. Such a fault is carried over to this next Round Key $K^9_{i,j}$, affecting 5 bytes; one in each of the first 3 columns and two in the final column. Five more byte differences are ultimately reflected in the final faulty ciphertext. A more complex analysis follows to help obtain 4 bytes in the 3^{rd} column of $K^9_{i,j}$. The attack makes use of the previous values obtained in $K^9_{i,j}$ (see Step 1). The last 8 bytes of $K^9_{i,j}$ is thus obtained.

3. A fault is induced on one of the intermediate State bytes at the end of the 8^{th} round - $P^8_{i,j}$, just before entering the 9^{th} round. The fault should be positioned such that a ShiftRow and a Mixcolumn for the 8^{th} round places the entire altered column in either of the last two columns. The benefit of doing this is to be able to use the known values of $K^9_{i,j}$ from the previous two steps to be EXORed with these altered values. This helps in generating the entire 10^{th} round key, leading to effective cryptanalysis.

The second model suggests that one also needs to protect tampering of the *Key Schedule*, other than the intermediate State values to resist DFA attacks.

3.3 An extension to the bit-fault DFA attack

The bit fault DFA attack or the last round differential attack (from the first section) relied on tweaking bits just at the end of the 9^{th} round. The following section extends the same attack by applying bit faults a round earlier.

3.3.1 The basic behaviour

Consider the case, when a bit fault is applied to the first byte at the end of the 8^{th} round. The new notations to be noted here are:

-$P_{i,j}$ for the intermediate State value at the end of the 8^{th} round for the i^{th} row and the j^{th} column.
-$K^9_{i,j}$ for the Round Key values of the 9^{th} round for the i^{th} row and the j^{th} column.
-$K^{10}_{i,j}$ for the Round Key values of the 10^{th} round for the i^{th} row and the j^{th} column.
-$C_{i,j}$ for the Ciphertext values at the end of the 10^{th} round for the i^{th} row and the j^{th} column.

It needs to be understood how one such bit fault affects the final ciphertext two rounds later. Figure 6 explains the same:

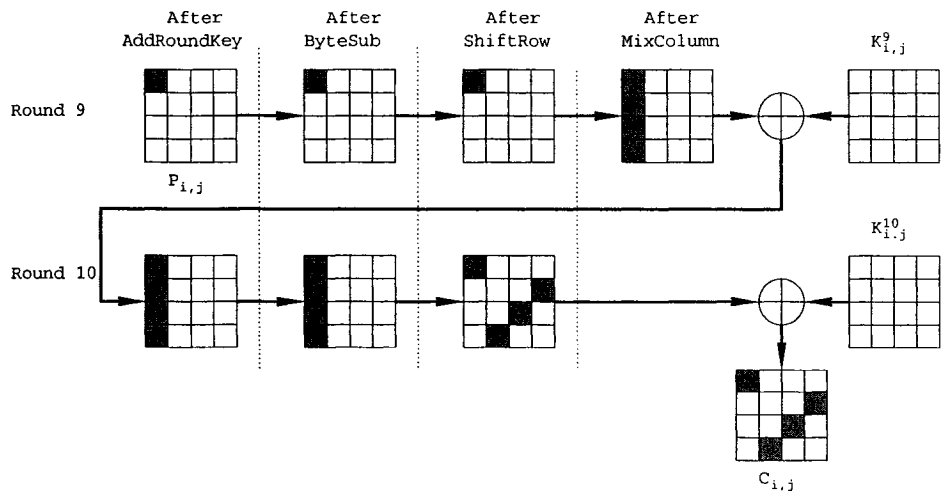

Figure 6: The 8 Round bit-fault DFA Attack

As mentioned, the gray block of $P_{i,j}$ - $P_{0,0}$, is where a bit fault is induced at the end of the 8^{th} round. The *ByteSub* does not spread the bit change to any other bytes. The fault being in the first row, *ShiftRow* does not affect its position, though a fault occuring in any other row will affect a change. This step gets the $P_{0,0}$, $P_{1,1}$, $P_{2,2}$ and $P_{3,3}$ under the same column. A *MixColumn* now affects all the four bytes of the first column, as reflected by the gray area in the figure. *AddRoundKey* adds $K^9_{i,j}$ to the same, with no changes spread to the other bytes. The new *ByteSub* adds further complexity to the altered bytes. The *ShiftRow* now changes the positions of these bytes. The last row does not include the *MixColumn*; an *AddRoundKey* adds $K^{10}_{i,j}$ to all the bytes and $C_{i,j}$ is received. From the figure, it can be easily seen that the altered bytes are at $C_{0,0}$, $C_{1,3}$, $C_{2,2}$ and $C_{3,1}$, with the other values being the same.

It can be deduced from the same figure, that when similar bit faults are applied to either $P_{1,1}$, $P_{2,2}$ or $P_{3,3}$, the altered bytes in the ciphertext will be in similar positions as above. A general rule can be derived by this observation.

$$\begin{aligned}
&\text{Change in } P_{0,0}, P_{1,1}, P_{2,2} \text{ or } P_{3,3} \Rightarrow \text{Change in } C_{0,0}, C_{1,3}, C_{2,2} \text{ and } C_{3,1}\\
&\text{Change in } P_{0,1}, P_{1,2}, P_{2,3} \text{ or } P_{3,0} \Rightarrow \text{Change in } C_{0,1}, C_{1,0}, C_{2,3} \text{ and } C_{3,2}\\
&\text{Change in } P_{0,2}, P_{1,3}, P_{2,0} \text{ or } P_{3,1} \Rightarrow \text{Change in } C_{0,2}, C_{1,1}, C_{2,0} \text{ and } C_{3,3}\\
&\text{Change in } P_{0,3}, P_{1,0}, P_{2,1} \text{ or } P_{3,2} \Rightarrow \text{Change in } C_{0,3}, C_{1,2}, C_{2,1} \text{ and } C_{3,0}
\end{aligned}$$

3.3.2 The algebraic model

Considering the case when a bit-fault is applied to $P_{0,0}$, we get the following four equations:

$$\begin{aligned}
C_{0,0} = ByteSub((&ByteSub(P_{0,0}).02+\\
&ByteSub(P_{1,1}).03+\\
&ByteSub(P_{2,2}).01+\\
&ByteSub(P_{3,3}).01) \oplus K^9_{0,0}) \oplus K^{10}_{0,0}
\end{aligned} \tag{8}$$

$$\begin{aligned}
C_{1,3} = ByteSub((&ByteSub(P_{0,0}).01+\\
&ByteSub(P_{1,1}).02+\\
&ByteSub(P_{2,2}).03+\\
&ByteSub(P_{3,3}).01) \oplus K^9_{1,0}) \oplus K^{10}_{1,3}
\end{aligned} \tag{9}$$

$$
\begin{aligned}
C_{2,2} = ByteSub((&ByteSub(P_{0,0}).01+ \\
&ByteSub(P_{1,1}).01+ \\
&ByteSub(P_{2,2}).02+ \\
&ByteSub(P_{3,3}).03) \oplus K^{9}_{2,0}) \oplus K^{10}_{2,2}
\end{aligned}
\tag{10}
$$

$$
\begin{aligned}
C_{3,1} = ByteSub((&ByteSub(P_{0,0}).03+ \\
&ByteSub(P_{1,1}).01+ \\
&ByteSub(P_{2,2}).01+ \\
&ByteSub(P_{3,3}).02) \oplus K^{9}_{3,0}) \oplus K^{10}_{3,1}
\end{aligned}
\tag{11}
$$

$P_{0,0}$ reflects changes in four ciphertexts deduced above. The absence of *ShiftRow* in the equations is made up by using appropriate bytes at the right places. For example, in equation (9), $C_{1,3}$ is obtained by applying each *ByteSub* values of $P_{0,0}$, $P_{1,1}$, $P_{2,2}$ and $P_{3,3}$ to *MixColumn* values of 01,02,03 and 01 respectively all of which gather under the first column. An EXOR with $K^{9}_{1,0}$ is applied, followed by applying the *ByteSub* of the calculated value to an EXOR with $K^{10}_{1,3}$. Figure 6 validates the model of equations generated above.

The remaining ciphertext values for $C_{i,j}$ are represented similarly in a much more simpler manner. For $\forall x,y \in \{0...3\}, x,y \notin \{\{0,0\},\{1,3\},\{2,2\}.\{3,1\}\}$, if $\{i',j'\}$ and $\{i'',j''\}$ hold corresponding distinct values $\in \{0...3\}$ for P and K respectively, then:

$$
C_{i,j} = ByteSub((MixColumn(ByteSub(P_{i',j'})) \oplus K^{9}_{i'',j''}) \oplus K^{10}_{i,j} \tag{12}
$$

Let $P'_{i,j}$ be the altered byte, in our case it is $P'_{0,0}$. Then, $C'_{i,j}$ are the corresponding ciphertext values, generated at the end of the 10^{th} round, when a bit fault is applied. The following equations are then deduced:

$$
\begin{aligned}
C'_{0,0} = ByteSub((&ByteSub(P'_{0,0}).02+ \\
&ByteSub(P_{1,1}).03+ \\
&ByteSub(P_{2,2}).01+ \\
&ByteSub(P_{3,3}).01) \oplus K^{9}_{0,0}) \oplus K^{10}_{0,0}
\end{aligned}
\tag{13}
$$

$$C'_{1,3} = ByteSub((ByteSub(P'_{0,0}).01+ \\ ByteSub(P_{1,1}).02+ \\ ByteSub(P_{2,2}).03+ \\ ByteSub(P_{3,3}).01) \oplus K^9_{1,0}) \oplus K^{10}_{1,3} \tag{14}$$

$$C'_{2,2} = ByteSub((ByteSub(P'_{0,0}).01+ \\ ByteSub(P_{1,1}).01+ \\ ByteSub(P_{2,2}).02+ \\ ByteSub(P_{3,3}).03) \oplus K^9_{2,0}) \oplus K^{10}_{2,2} \tag{15}$$

$$C'_{3,1} = ByteSub((ByteSub(P'_{0,0}).03+ \\ ByteSub(P_{1,1}).01+ \\ ByteSub(P_{2,2}).01+ \\ ByteSub(P_{3,3}).02) \oplus K^9_{3,0}) \oplus K^{10}_{3,1} \tag{16}$$

Note how a change in $P_{0,0}$ permeates four different bytes. The other ciphertexts can be simply represented as:

$$C'_{i,j} = ByteSub((MixColumn(ByteSub(P_{i',j'})) \oplus K^9_{i'',j''}) \oplus K^{10}_{i,j} \tag{17}$$

Thus, from equations (12) and (17), an EXOR yields:

$$C_{i,j} \oplus C'_{i,j} = 0 \qquad \forall x,y \in \{0...3\}, x,y \notin \{\{0,0\},\{1,3\},\{2,2\}.\{3,1\}\} \tag{18}$$

Hence, it can be mathematically seen that no change in the other ciphertexts occur. However, for the affected bytes, an EXOR of corresponding ciphertexts from equations (8) to (11) with equations (13) to (16) gives us the following:

$$
\begin{aligned}
C_{0,0} \oplus C'_{0,0} = ByteSub((&ByteSub(P_{0,0}).02+ \\
&ByteSub(P_{1,1}).03+ \\
&ByteSub(P_{2,2}).01+ \\
&ByteSub(P_{3,3}).01) \oplus K^9_{0,0}) \\
\oplus\ ByteSub((&ByteSub(P'_{0,0}).02+ \\
&ByteSub(P_{1,1}).03+ \\
&ByteSub(P_{2,2}).01+ \\
&ByteSub(P_{3,3}).01) \oplus K^9_{0,0})
\end{aligned}
\tag{19}
$$

$$
\begin{aligned}
C_{1,3} \oplus C'_{1,3} = ByteSub((&ByteSub(P_{0,0}).01+ \\
&ByteSub(P_{1,1}).02+ \\
&ByteSub(P_{2,2}).03+ \\
&ByteSub(P_{3,3}).01) \oplus K^9_{1,0}) \\
\oplus\ ByteSub((&ByteSub(P'_{0,0}).01+ \\
&ByteSub(P_{1,1}).02+ \\
&ByteSub(P_{2,2}).03+ \\
&ByteSub(P_{3,3}).01) \oplus K^9_{1,0})
\end{aligned}
\tag{20}
$$

$$
\begin{aligned}
C_{2,2} \oplus C'_{2,2} = ByteSub((&ByteSub(P_{0,0}).01+ \\
&ByteSub(P_{1,1}).01+ \\
&ByteSub(P_{2,2}).02+ \\
&ByteSub(P_{3,3}).03) \oplus K^9_{2,0}) \\
\oplus\ ByteSub((&ByteSub(P'_{0,0}).01+ \\
&ByteSub(P_{1,1}).01+ \\
&ByteSub(P_{2,2}).02+ \\
&ByteSub(P_{3,3}).03) \oplus K^9_{2,0})
\end{aligned}
\tag{21}
$$

$$
\begin{aligned}
C_{3,1} \oplus C'_{3,1} = ByteSub((ByteSub(P_{0,0}).03+ \\
ByteSub(P_{1,1}).01+ \\
ByteSub(P_{2,2}).01+ \\
ByteSub(P_{3,3}).02) \ \oplus \ K^9_{3,0}) \\
\oplus \ ByteSub((ByteSub(P'_{0,0}).03+ \\
ByteSub(P_{1,1}).01+ \\
ByteSub(P_{2,2}).01+ \\
ByteSub(P_{3,3}).02) \ \oplus \ K^9_{3,0})
\end{aligned}
\tag{22}
$$

A noteworthy observation from the above equations is the absence of the final round key $K^{10}_{i,j}$ in them. Its presence, one would later understand , would have meant requiring to format a key value in terms of the other using the Key Expansion algorithm. However, that would add extra complexity to the attack.

3.3.3 The attack

Assume that the cryptanalyst knows the location of the faulty bit in the first byte $P'_{0,0}$. Say, it is the last (rightmost) bit of that byte. That means the difference between $P_{0,0}$ and $P'_{0,0}$ is "0x01". That is, $P'_{0,0} = P_{0,0} \oplus$ 0x01. This leads to the following:

$$
\begin{aligned}
Equation(19) \quad \Rightarrow \quad & C_{0,0} \oplus C'_{0,0} = ByteSub(ByteSub(P_{0,0}).02 \oplus D_{0,0}) \ \oplus \\
& \qquad ByteSub(ByteSub(P'_{0,0}).02 \oplus D_{0,0}) \\
\Rightarrow \quad & C_{0,0} \oplus C'_{0,0} = ByteSub(ByteSub(P_{0,0}).02 \oplus D_{0,0}) \ \oplus \\
& \qquad ByteSub(ByteSub(P_{0,0} \oplus 0x01).02 \oplus D_{0,0})
\end{aligned}
\tag{23}
$$

$$
\begin{aligned}
Equation(20) \quad \Rightarrow \quad & C_{1,3} \oplus C'_{1,3} = ByteSub(ByteSub(P_{0,0}).01 \oplus D_{1,3}) \ \oplus \\
& \qquad ByteSub(ByteSub(P'_{0,0}).01 \oplus D_{1,3}) \\
\Rightarrow \quad & C_{1,3} \oplus C'_{1,3} = ByteSub(ByteSub(P_{0,0}).01 \oplus D_{1,3}) \ \oplus \\
& \qquad ByteSub(ByteSub(P_{0,0} \oplus 0x01).01 \oplus D_{1,3})
\end{aligned}
\tag{24}
$$

$$
\begin{aligned}
Equation(21) \quad &\Rightarrow \quad C_{2,2} \oplus C'_{2,2} = ByteSub(ByteSub(P_{0,0}).01 \oplus D_{2,2}) \oplus \\
&\qquad\qquad ByteSub(ByteSub(P'_{0,0}).01 \oplus D_{2,2}) \\
&\Rightarrow \quad C_{2,2} \oplus C'_{2,2} = ByteSub(ByteSub(P_{0,0}).01 \oplus D_{2,2}) \oplus \\
&\qquad\qquad ByteSub(ByteSub(P_{0,0} \oplus 0x01).01 \oplus D_{2,2})
\end{aligned} \tag{25}
$$

$$
\begin{aligned}
Equation(22) \quad &\Rightarrow \quad C_{3,1} \oplus C'_{3,1} = ByteSub(ByteSub(P_{0,0}).03 \oplus D_{3,1}) \oplus \\
&\qquad\qquad ByteSub(ByteSub(P'_{0,0}).03 \oplus D_{3,1}) \\
&\Rightarrow \quad C_{3,1} \oplus C'_{3,1} = ByteSub(ByteSub(P_{0,0}).03 \oplus D_{3,1}) \oplus \\
&\qquad\qquad ByteSub(ByteSub(P_{0,0} \oplus 0x01).03 \oplus D_{3,1})
\end{aligned} \tag{26}
$$

$D_{0,0}$, $D_{1,3}$, $D_{2,2}$ and $D_{3,1}$ are constant bytes encompassing the *MixColumn* and the 9^{th} Round key values. The purpose of the above equation is to find values for $P_{0,0}$, for which there exists a constant byte D, solving which gives the differential faulty ciphertext $(C \oplus C')$.

A pseudo C notation for doing the same is given below:

```
FindP(C,C',hexmult)
{
    byte PossibleBytes[150];
    byte T, T'
    int counter = 0;
    For( i = 0; i < 256; i=i+2 )
    {
        If( hexmult == "03")
        { T = Mix(ByteSub[i],0x03); T' = Mix(ByteSub[i∧0x01], 0x03)}
        If( hexmult == "02")
        { T = Mix(ByteSub[i],0x02); T' = Mix(ByteSub[i∧0x01], 0x02)}
        If( hexmult == "01")
        { T = Mix(ByteSub[i],0x01); T' = Mix(ByteSub[i∧0x01], 0x01)}
        For( j = 0; j < 256; j++ )
        {
            If( ByteSub[T∧j] ∧ ByteSub[T'∧j] == C ∧ C')
            {
                PossibleBytes[counter] = i;
                PossibleBytes[counter + 1] = i ∧ 0x01;
                counter = counter + 2
            }
        }
    }
}
```

The input to the algorithm includes the ciphertext C, the faulty ciphertext C' and the Mixcolumn multiplier for the byte *hexmult*, which is '02', '01', '01' and '03' for equations (23), (24), (25) and (26) respectively. The algorithm attempts to find all the possible bytes that satisfy the above equations. General tests reveal that for every difference of $(C \oplus C')$ value, out of 256 bytes, there are not more than 150 bytes that hold true with the equation. Hence the results are store in *PossibleBytes[]* with a limit of 150. In fact a maximum of 75 byte pairs of P and P' holds true and hence i runs only 128 times, requiring no checks again for $(i \wedge 0x01)$. The temporary bytes T and T' store the Mix of the *ByteSub* of i with

the *hexmult*. The j acts as a constant D from the equation, and it is looped over to see if satisfies the condition. The positive paired results are stored in two successive array locations of *PossibleBytes*[].

Applications of the above pseudocode are done for equations (23) to (26), involving different ciphertexts and hexmults. The *PossibleBytes*[] array received will contains all possible values held by $P_{0,0}$ and it is observed that, byte values common to all the analysis equations, number to almost one-eighth of the numbers for a single equation. In other words, for the common values in the *PossibleBytes*[] array for all the four equations, *FinalPossibleBytes*[], is just around 20, brought down from maximum limits of 150 for each array.

A similar analysis is done by applying a bit-fault on the penultimate bit of the $P_{0,0}$ and reviewing the new ciphertexts. Here the difference of $P \oplus P'$ will be 0x02, which should be reflected in a new pseudocode, similar to the one above. Four new equations are also realised. The *FinalPossibleBytes*[] array received will similarly contain all possible values held by $P_{0,0}$ for those ciphertexts and it will also number around 20. The common value in both the *FinalPossibleBytes*[] arrays will reflect the value of $P_{0,0}$. Eight equations comprising equations (23) to (26) and the four new ones thus make up this retrieval. A highly improbable case occurs, if there is more than one common value in both. Then another bit-fault is applied, a new *FinalPossibleBytes*[] is made, and a common value with the earlier two arrays is sought.

The primary assumption in this byte retrieval process involved only the toggling of the last bit and the penultimate bit. However, it is easy to understand that the attacker can attack any bit of his choice, provided he knows its position. The subsequent P' can be $P \oplus x, \quad x \forall \{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80\}$. Uncertainty in the faulty bit position will involve more extensive analysis - going through all bit toggles for each ciphertext pair and more bit-faults.

Equations (8) to (11) suggest that $C_{0,0}$, $C_{1,3}$, $C_{2,2}$, and $C_{3,1}$ each also rely on $P_{1,1}$, $P_{2,2}$, $P_{3,3}$ other than $P_{0,0}$. This means that if we perform the same set of steps on $P_{1,1}$, $P_{2,2}$ and $P_{3,3}$, we can recover their values. Finding them will require application of a likely average of 6 bit-faults and solving 24 equations. In effect, finding the whole word, $P_{0,0}$, $P_{1,1}$, $P_{2,2}$ and $P_{3,3}$, requires 8 bit-faults and solving 32 corresponding equations.

Applying the same concept, one can extract the other $P_{i,j}$ bytes using 24 bit-faults and 96 corresponding equations. The entire plaintext at the end of the 8^{th} round can be obtained

solving 128 equations using 32 bit-faults.

The final step in this attack is the extraction of the key. Recalling equation (19):

$$\begin{aligned} C_{0,0} \oplus C'_{0,0} = ByteSub((ByteSub(P_{0,0}).02+ \\ ByteSub(P_{1,1}).03+ \\ ByteSub(P_{2,2}).01+ \\ ByteSub(P_{3,3}).01) \oplus K^9_{0,0}) \\ \oplus ByteSub((ByteSub(P'_{0,0}).02+ \\ ByteSub(P_{1,1}).03+ \\ ByteSub(P_{2,2}).01+ \\ ByteSub(P_{3,3}).01) \oplus K^9_{0,0}) \end{aligned}$$

From the 128 equations used, there are 8 equations for each byte of the key. The above equation toggles one bit of $P_{0,0}$ and there is a presence of $K^9_{0,0}$. Another equation can be found, which toggles another bit of the same $P_{0,0}$. Now, for each of the bytes $P_{1,1}$, $P_{2,2}$ and $P_{3,3}$, two equations can be found for two different bit-faults. That is, $P'_{1,1}$, $P'_{2,2}$ and $P'_{3,3}$ each contributes two equations. Only one value for $K^9_{0,0}$ satisfies these eight equations.

Similar evaluation follows for the remaining bytes of the key, leading to cryptanalysis.

3.3.4 Byte Extension

If instead of the bit-fault, a byte fault is induced, the same attack can be followed provided the attacker knows which bits will be faulted in a single byte. In other words, instead of an EXOR with $\{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80\}$, if one EXORs $P_{i,j}$ with any other known byte to get $P'_{i,j}$, a similar attack can ensue. It is assumed that such a fault can only occur on concrete bytes of $P_{i,j}$ and not overlapping different bytes in them.

Chapter 4

Key Representation and its properties

The following chapter is an original contribution and explores the whole Expanded key, dependent on a single set of 16 bytes for a 128-bit key. The first section represents the Key as an expression valid for the different rounds, whereas the second section notes a few interesting observations and properties based on such a representation. The final section discusses the importance of these properties.

4.1 The Different Round Keys

Assume a 128-bit key for the AES cipher. The Round Keys are derived from the Cipher Key by the Key Expansion process. *Section 1.2.4* details how the process is carried out. In most smart-cards, where the AES might be used, it would be infeasible/inefficient to calculate these Round Keys at every step. In fact, it is considered appropriate to represent all the Round Keys as a function of the original Cipher Key, by which all their bytes can be calculated without any reliance on any other Round Key.

Round Keys computed for 10 rounds by the Key Expansion process or the Inverse Key Expansion processes, as functions of the Cipher Key or the Inverse Cipher Key respectively, tend to be very complex involving a large number of terms. Moreover, the Key Expansion process and the Inverse Key Expansion processes are slightly different and representing the 10 Round Keys by both the processes would require 20 different representations. Hence, a middle ground is reached upon by considering the 5^{th} Round Key as a known set of bytes.

Table 2 lists the six representations from Round Keys 5 to 10. The Key Expansion process is responsible at every step for the deduction of the next Round Key. The $S()$

signifies the application of an S-Box. Note the appropriate addition of Round constants to the first column for every Round Key.

Table 3 lists the six representations from Round Keys 5 to 0, the Cipher Key. The Inverse Key Expansion process, similarly is responsible at every step for the deduction of the next Round Key. Also, the $S()$ signifies the application of an S-Box. Note the subtle difference with the normal Key Expansion and the appropriate addition of Round constants to the first column for every Round Key. These Round Keys are not directly EXORed with the plaintexts. They undergo an *InvMixColumn* followed by the EXOR, if used in an order similar to the encryption process. See *Section*1.2.6.

Table 2: The Key Representation from Round 5 to Round 10

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

(a) Round 5 Key

$P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20$	$P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20$	$P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20$	$P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20$
$P_{1,0} \oplus$ $S(P_{2,3})$	$P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3})$	$P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3})$	$P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3})$
$P_{2,0} \oplus$ $S(P_{3,3})$	$P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3})$	$P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3})$	$P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3})$
$P_{3,0} \oplus$ $S(P_{0,3})$	$P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3})$	$P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3})$	$P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3})$

(b) Round 6 Key

$P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $0x20 \oplus 0x40$	$P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $0x40$	$P_{0,2} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $0x20 \oplus 0x40$	$P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $0x40$
$P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))$	$P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))$	$P_{1,2} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))$	$P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))$
$P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3}))$	$P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3}))$	$P_{2,2} \oplus P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3}))$	$P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3}))$
$P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)$	$P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)$	$P_{3,2} \oplus P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)$	$P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)$

(c) Round 7 Key

$P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))) \oplus$ $0x20 \oplus 0x40 \oplus 0x80$	$P_{0,1} \oplus$ $P_{0,0} \oplus$ $S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))) \oplus$ $0x20 \oplus 0x80$	$P_{0,2} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))) \oplus$ $0x40 \oplus 0x80$	$P_{0,3} \oplus$ $P_{0,2}$ $\oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3}))) \oplus$ $0x80$
$P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3})) \oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))$	$P_{1,1} \oplus$ $P_{1,0} \oplus$ $S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))$	$P_{1,2} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3})) \oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))$	$P_{1,3} \oplus$ $P_{1,2}$ $\oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))$
$P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})) \oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20))$	$P_{2,1} \oplus$ $P_{2,0} \oplus$ $S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20))$	$P_{2,2} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})) \oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20))$	$P_{2,3} \oplus$ $P_{2,2}$ $\oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20))$
$P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3}) \oplus 0x40))$	$P_{3,1} \oplus$ $P_{3,0} \oplus$ $S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3}) \oplus 0x40))$	$P_{3,2} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3}) \oplus 0x40))$	$P_{3,3} \oplus$ $P_{3,2}$ $\oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3}) \oplus 0x40))$

(d) Round 8 key

$P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus$ $P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))) \oplus$ $0x20 \oplus 0x40 \oplus 0x80 \oplus 0x1B$	$P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus$ $P_{1,0} \oplus S(P_{2,3})) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))) \oplus$ $0x40 \oplus 0x1B$	$P_{0,2} \oplus$ $S(P_{1,3} \oplus P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus$ $P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))) \oplus$ $0x80 \oplus 0x1B$	$P_{0,3} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})))) \oplus$ $0x1B$
$P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3})) \oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus$ $P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))$	$P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus$ $P_{2,0} \oplus S(P_{3,3})) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))$	$P_{1,2} \oplus$ $S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus$ $P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))$	$P_{1,3} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))$
$P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})) \oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus S(P_{0,3} \oplus P_{0,2} \oplus$ $P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40)))$	$P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus$ $P_{3,0} \oplus S(P_{0,3})) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40)))$	$P_{2,2} \oplus$ $S(P_{3,3} \oplus P_{3,1} \oplus S(P_{0,3} \oplus P_{0,2} \oplus$ $P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40)))$	$P_{2,3} \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40)))$
$P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2} \oplus$ $P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus 0x40)) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80)$	$P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80)$	$P_{3,2} \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2} \oplus$ $P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus 0x40)) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80)$	$P_{3,3} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80)$

(e) Round 9 key

$P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}))$ $\oplus S(P_{1,3} \oplus P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus$ $P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3}$ $\oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})))) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1}$ $\oplus S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))) \oplus$ $0x20 \oplus 0x40 \oplus 0x80 \oplus 0x1B \oplus 0x36$	$P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3})$ $\oplus S(P_{1,3} \oplus P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus$ $P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1}$ $\oplus S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))) \oplus$ $0x20 \oplus 0x80 \oplus 0x36$	$P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3}$ $\oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})))) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1}$ $\oplus S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))) \oplus$ $0x20 \oplus 0x1B \oplus 0x36$	$P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3})$ $\oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1}$ $\oplus S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20)))) \oplus$ $0x20 \oplus 0x36$
$P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))$ $\oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus$ $P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20))) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1}$ $\oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))))$	$P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3})$ $\oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus$ $P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20))) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1}$ $\oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))))$	$P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3}) \oplus 0x20))) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1}$ $\oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))))$	$P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3})$ $\oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1}$ $\oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))))$
$P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))$ $\oplus S(P_{3,3} \oplus P_{3,1} \oplus S(P_{0,3} \oplus P_{0,2} \oplus$ $P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1}$ $\oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80))$	$P_{2,1} \oplus P_{2,0} \oplus P_{2,0} \oplus S(P_{3,3})$ $\oplus S(P_{3,3} \oplus P_{3,1} \oplus S(P_{0,3} \oplus P_{0,2} \oplus$ $P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1}$ $\oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80))$	$P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3}) \oplus 0x40))) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1}$ $\oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80))$	$P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus S(P_{3,3})$ $\oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1}$ $\oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80))$
$P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus$ $P_{0,0} \oplus S(P_{1,3}) \oplus 0x20) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2} \oplus$ $P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus 0x40)) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1}$ $\oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}$ $\oplus S(P_{0,3})))) \oplus 0x1B)$	$P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2} \oplus$ $P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3}) \oplus 0x40)) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1}$ $\oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}$ $\oplus S(P_{0,3})))) \oplus 0x1B)$	$P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3}))) \oplus 0x80) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1}$ $\oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}$ $\oplus S(P_{0,3})))) \oplus 0x1B)$	$P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})$ $\oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1}$ $\oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}$ $\oplus S(P_{0,3})))) \oplus 0x1B)$

(f) Round 10 key

Table 3: The Key Representation from Round 5 to Round 0

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

(a) Round 5 Key

$P_{0,0} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$	$P_{0,1} \oplus P_{0,0}$	$P_{0,2} \oplus P_{0,1}$	$P_{0,3} \oplus P_{0,2}$
$P_{1,0} \oplus S(P_{2,3} \oplus P_{2,2})$	$P_{1,1} \oplus P_{1,0}$	$P_{1,2} \oplus P_{1,1}$	$P_{1,3} \oplus P_{1,2}$
$P_{2,0} \oplus S(P_{3,3} \oplus P_{3,2})$	$P_{2,1} \oplus P_{2,0}$	$P_{2,2} \oplus P_{2,1}$	$P_{2,3} \oplus P_{2,2}$
$P_{3,0} \oplus S(P_{0,3} \oplus P_{0,2})$	$P_{3,1} \oplus P_{3,0}$	$P_{3,2} \oplus P_{3,1}$	$P_{3,3} \oplus P_{3,2}$

(b) Round 4 Key

$P_{0,0} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus S(P_{1,3} \oplus P_{1,1})$ $0x10 \oplus 0x08$	$P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$	$P_{0,2} \oplus P_{0,0}$	$P_{0,3} \oplus P_{0,1}$
$P_{1,0} \oplus S(P_{2,3} \oplus P_{2,2}) \oplus S(P_{2,3} \oplus P_{2,1})$	$P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2})$	$P_{1,2} \oplus P_{1,0}$	$P_{1,3} \oplus P_{1,1}$
$P_{2,0} \oplus S(P_{3,3} \oplus P_{3,2}) \oplus S(P_{3,3} \oplus P_{3,1})$	$P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2})$	$P_{2,2} \oplus P_{2,0}$	$P_{2,3} \oplus P_{2,1}$
$P_{3,0} \oplus S(P_{0,3} \oplus P_{0,2}) \oplus S(P_{0,3} \oplus P_{0,1})$	$P_{3,1} \oplus S(P_{0,3} \oplus P_{0,2})$	$P_{3,2} \oplus P_{3,0}$	$P_{3,3} \oplus P_{3,1}$

(c) Round 3 Key

$P_{0,0} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus S(P_{1,3} \oplus P_{1,1})$ $\oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0}) \oplus$ $0x10 \oplus 0x08 \oplus 0x04$	$P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $0x08$	$P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $0x10$	$P_{0,3} \oplus P_{0,2} \oplus$ $P_{0,1} \oplus P_{0,0}$
$P_{1,0} \oplus S(P_{2,3} \oplus P_{2,2}) \oplus S(P_{2,3} \oplus P_{2,1})$ $\oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0})$	$P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3} \oplus P_{2,1})$	$P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3} \oplus P_{2,2})$	$P_{1,3} \oplus P_{1,2} \oplus$ $P_{1,1} \oplus P_{1,0}$
$P_{2,0} \oplus S(P_{3,3} \oplus P_{3,2}) \oplus S(P_{3,3} \oplus P_{3,1})$ $\oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0})$	$P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3} \oplus P_{3,1})$	$P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3} \oplus P_{3,2})$	$P_{2,3} \oplus P_{2,2} \oplus$ $P_{2,1} \oplus P_{2,0}$
$P_{3,0} \oplus S(P_{0,3} \oplus P_{0,2}) \oplus S(P_{0,3} \oplus P_{0,1})$ $\oplus S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0})$	$P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3} \oplus P_{0,1})$	$P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3} \oplus P_{0,2})$	$P_{3,3} \oplus P_{3,2} \oplus$ $P_{3,1} \oplus P_{3,0}$

(d) Round 2 Key

$P_{0,0} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0}) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2})) \oplus$ $0x10 \oplus 0x08 \oplus 0x04 \oplus 0x02$	$P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0}) \oplus$ $0x10 \oplus 0x04$	$P_{0,2} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $0x10 \oplus 0x08$	$P_{0,3} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $0x10$
$P_{1,0} \oplus$ $S(P_{2,3} \oplus P_{2,2}) \oplus$ $S(P_{2,3} \oplus P_{2,1}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0}) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2}))$	$P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,2}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0})$	$P_{1,2} \oplus$ $S(P_{2,3} \oplus P_{2,2}) \oplus$ $S(P_{2,3} \oplus P_{2,1})$	$P_{1,3} \oplus$ $S(P_{2,3} \oplus P_{2,2})$
$P_{2,0} \oplus$ $S(P_{3,3} \oplus P_{3,2}) \oplus$ $S(P_{3,3} \oplus P_{3,1}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2}))$	$P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,2}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0})$	$P_{2,2} \oplus$ $S(P_{3,3} \oplus P_{3,2}) \oplus$ $S(P_{3,3} \oplus P_{3,1})$	$P_{2,3} \oplus$ $S(P_{3,3} \oplus P_{3,2})$
$P_{3,0} \oplus$ $S(P_{0,3} \oplus P_{0,2}) \oplus$ $S(P_{0,3} \oplus P_{0,1}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0}) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2}))$	$P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,2}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0})$	$P_{3,2} \oplus$ $S(P_{0,3} \oplus P_{0,2}) \oplus$ $S(P_{0,3} \oplus P_{0,1})$	$P_{3,3} \oplus$ $S(P_{0,3} \oplus P_{0,2})$

(e) Round 1 Key

$P_{0,0} \oplus$ $S(P_{1,3} \oplus P_{1,2}) \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0}) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2})) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1})) \oplus$ $0x10 \oplus 0x08 \oplus 0x04 \oplus 0x02 \oplus 0x01$	$P_{0,1} \oplus P_{0,0} \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $S(P_{1,3} \oplus S(P_{2,3} \oplus P_{2,2})) \oplus$ $0x08 \oplus 0x02$	$P_{0,2} \oplus P_{0,1} \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0}) \oplus$ $0x08 \oplus 0x04$	$P_{0,3} \oplus P_{0,2} \oplus$ $S(P_{1,3} \oplus P_{1,1}) \oplus$ $0x08$
$P_{1,0} \oplus$ $S(P_{2,3} \oplus P_{2,2}) \oplus$ $S(P_{2,3} \oplus P_{2,1}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0}) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2})) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus S(P_{3,3} \oplus P_{3,1}))$	$P_{1,1} \oplus P_{1,0} \oplus$ $S(P_{2,3} \oplus P_{2,1}) \oplus$ $S(P_{2,3} \oplus S(P_{3,3} \oplus P_{3,2}))$	$P_{1,2} \oplus P_{1,1} \oplus$ $S(P_{2,3} \oplus P_{2,1}) \oplus$ $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0})$	$P_{1,3} \oplus P_{1,2} \oplus$ $S(P_{2,3} \oplus P_{2,1})$
$P_{2,0} \oplus$ $S(P_{3,3} \oplus P_{3,2}) \oplus$ $S(P_{3,3} \oplus P_{3,1}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0}) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2})) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus S(P_{0,3} \oplus P_{0,1}))$	$P_{2,1} \oplus P_{2,0} \oplus$ $S(P_{3,3} \oplus P_{3,1}) \oplus$ $S(P_{3,3} \oplus S(P_{0,3} \oplus P_{0,2}))$	$P_{2,2} \oplus P_{2,1} \oplus$ $S(P_{3,3} \oplus P_{3,1}) \oplus$ $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0})$	$P_{2,3} \oplus P_{2,2} \oplus$ $S(P_{3,3} \oplus P_{3,1})$
$P_{3,0} \oplus$ $S(P_{0,3} \oplus P_{0,2}) \oplus$ $S(P_{0,3} \oplus P_{0,1}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0}) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2})) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus S(P_{1,3} \oplus P_{1,1}))$	$P_{3,1} \oplus P_{3,0} \oplus$ $S(P_{0,3} \oplus P_{0,1}) \oplus$ $S(P_{0,3} \oplus S(P_{1,3} \oplus P_{1,2}))$	$P_{3,2} \oplus P_{3,1} \oplus$ $S(P_{0,3} \oplus P_{0,1}) \oplus$ $S(P_{0,3} \oplus P_{0,2} \oplus P_{0,1} \oplus P_{0,0})$	$P_{3,3} \oplus P_{3,2} \oplus$ $S(P_{0,3} \oplus P_{0,1})$

(f) The Cipher Key / Round 0 Key

4.2 Relevant Properties

Key Expansion process: With respect to the Round Key representations in Table 2, the following properties can be deduced:

1. An S-Box is applied to a byte of the last column of every previous Round Key and this new *expression* is added to the first column of every Round Key. Such an *expression* for every Round Key spreads over to all the bytes of the same row. This means that the second, third and the final columns are dependent on the first column for their evaluation.

   For example, in Table 2(b), $S(P_{1,3}) \oplus 0x20$ spreads over to all the bytes in the same row.

2. Under such an above mentioned *expression*, there are repeated applications of S-Box mappings over already prevalent S-Box mappings. The number of applications of such S-Box mappings in an added *expression* is equal to the number of applications of the Key Expansion process.

   For example, in Table 2(d), three applications of the process results in the addition of the *expression* $S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3})))$ in the first row, which contains three S-Box mappings.

3. The added *expression* to every $(k+4)^{th}$ Round Key byte $P_{i,j}^{k+4} \quad \forall i,j \in \{0...3\}$ is such that it includes bytes from all the rows, guaranteeing high amount of diffusion. Such an *expression* will contain four S-Box mappings of the following type:

   (a) Initially, an S-box of $P_{i,3}^{k}$, $S(P_{i,3}^{k})$.

   (b) Then, an S-box of $P_{(i+3)mod4,3}^{k+1}$, $S(P_{(i+3)mod4,3}^{k+1})$. This already contains the previous step's S-Box mapping.

   (c) Then, an S-box of $P_{(i+2)mod4,3}^{k+2}$, $S(P_{(i+2)mod4,3}^{k+2})$. This already contains the previous step's S-Box mapping.

   (d) Then, an S-box of $P_{(i+1)mod4,3}^{k+3}$, $S(P_{(i+1)mod4,3}^{k+3})$. This already contains the previous step's S-Box mapping.

   The change in i for all the four mappings confirms the presence of bytes from all the rows. In such a way, a $(k)^{th}$ Round Key byte $P_{i,j}^{k}$ propagates to the $(k+4)^{th}$ Round Key byte $P_{i,j}^{k+4}$ incorporating other column bytes.

For example, from Tables 2(e), the *expression* $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})))) \oplus 0x1B$ for the 9^{th} Round Key is obtained by the four S-Box mappings:

(a) $S(P_{0,3})$ from the 5^{th} Round Key(Table 2(a)).

(b) $S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))$ from the 6^{th} Round Key(Table 2(b)). This includes the previous step.

(c) $S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})))$ from the 7^{th} Round Key(Table 2(c)). This includes the previous step.

(d) $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3}))))$ from the 8^{th} Round Key(Table 2(d)). This includes the previous step.

4. There is a huge influence of the $P_{i,3}$ - the final column byte on all the bytes for all the Round Keys. As stipulated by the first property, the spreading of S-Box mappings of the final column bytes to the rest of the bytes can be attributed the reason for this phenomena. This essentially means that there is a huge diffusion/presence of the final column bytes. However, the first column byte $P_{i,0}$ diffuses the least and the other two column bytes $P_{i,1}$ and $P_{i,2}$ diffuse a bit more than $P_{i,0}$, but very less compared to $P_{i,3}$.

   For example, in Table 2(e), for the first row and the first column byte, $P_{i,3}$, $P_{i,2}$, $P_{i,1}$ and $P_{i,0}$ bytes are present 10 times, 4 times, 5 times and 4 times respectively.

5. The first column of a Round Key is always the first column of the Previous Round Key plus an S-Box mapping of the previous $P_{(i+1)mod4,3}$ byte plus a Round constant.

   For example, in Table 2(a), $P_{0,0}$ EXORs with $S(P_{1,3})$ and the Round constant $0x20$ to deduce values for Table 2(b).

6. The second column of a Round Key is always the second column of the earlier $P_{(i,1)^{k-2}}$ Round Key plus the new *expression* added for that particular Round Key. Thus alternate Round Keys have similar deducible second columns.

   For example, in Tables 2(a) and 2(c), $P_{0,1}$ EXORs with the new added *expression* $S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3} \oplus)) \oplus 0x40$.

7. The third column of a Round Key is always the EXOR of the third columns of the earlier $P_{(i,2)^{k-3}}$ Round Key, the $P_{(i,2)^{k-2}}$ Round Key and the $P_{(i,2)^{k-1}}$ Round Key. Thus, three third columns of Round Keys deduce the third column of a new Round Key.

For example, from Tables 2(a) to 2(d), the third column of 2(d)- $(P_{0,2} \oplus P_{0,1} \oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3})) \oplus S(P_{1,3} \oplus P_{1,1} \oplus S(P_{2,3} \oplus P_{2,2} \oplus P_{2,1} \oplus P_{2,0} \oplus S(P_{3,3}))) \oplus 0x40 \oplus 0x80)$ is an EXOR of the third column of 2(a)- $(P_{0,2})$, the third column of 2(b)- $(P_{0,2} \oplus P_{0,1} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus 0x20)$ and the third column of 2(c)- $(P_{0,2} \oplus P_{0,0} \oplus S(P_{1,3}) \oplus S(P_{1,3} \oplus P_{1,2} \oplus P_{1,1} \oplus P_{1,0} \oplus S(P_{2,3})) \oplus 0x20 \oplus 0x40)$.

8. The last column of a Round Key is always the last column of the earlier $P_{(i,3)^{k-4}}$ Round Key plus the new *expression* added for that particular Round Key. Thus Round Keys spaced every four rounds have similar deducible last columns.

   For example, in Tables 2(a) and 2(e), $P_{0,3}$ EXORs with the new added *expression* $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1} \oplus S(P_{3,3} \oplus P_{3,2} \oplus P_{3,1} \oplus P_{3,0} \oplus S(P_{0,3})))) \oplus 0x1B$.

Inverse Key Expansion process: With respect to the Round Key representations in Table 3, the following properties can be deduced.

1. An S-Box is applied to a byte of the last column of the same Round Key and this new *expression* is added to the first column of every Round Key. This is in contrast to the normal Key Expansion. This means that the first column can only be evaluated after the final column is calculated.

   For example, in Table 3(b), the evaluated final column byte $P_{1,3} \oplus P_{1,2}$ is added to $P_{0,0}$ resulting in $S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$.

2. The added *expression* for every Round Key spreads gradually over to all the bytes of the same row. In fact, a new *expression* added to the first column reaches the final column after three applications of the Inverse Key Expansion process. The first process application passes this *expression* to the second column, the next passes it to the third column culminating in it passed to the final column after three process applications. This is clearly a sign of low row-wise diffusion.

   For example, in Table 3(b), the added *expression*, $S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$ for Round Key 4 finally reaches the final column only in Round Key 1(Table 3(e)). In fact the same expression is present in all the row bytes.

3. Taking a cue from the above property, the number of S-Box mappings on prevalent S-Box mappings is much less. This is inherently due to the slow diffusion as explained above. A new *expression* for the k^{th} Round Key will reach the final column for the

$(k-3)^{th}$ Round Key. The $(k-4)^{th}$ Round Key will include an S-Box mapping of such an *expression* and this will be ultimately reach the final column only for the $(k-7)^{th}$ Round Key. There is much less complexity in the *expression*, compared to the Key Expansion process.

For example, in Table 3(f), the new *expression* added for the first column, is $S(P_{1,3} \oplus P_{1,2} \oplus S(P_{2,3} \oplus P_{2,1})) \oplus 0x01$, which includes only two S-Box mappings.

4. On the basis of the above two properties, it is also seen that there is very less column-wise diffusion. For a particular byte position, four process applications of $P_{i,j}$ finally reflects bytes from $P^k_{(i+2)mod4,j}$. Another four will reflect the bytes from $P^k_{(i+3)mod4,j}$ with another four required for a complete diffusion. Hence, there is a considerable influence of the $P_{i,3}$ - the final column byte on all the bytes for all the Round Keys, though its number is not much as compared to the Key Expansion process.

   For example, in Table 3(e), for the first row and the first column byte, $P_{i,3}$, $P_{i,2}$, $P_{i,1}$ and $P_{i,0}$ bytes are present 5 times, 3 times, 2 times and 2 times respectively.

5. The first column of a Round Key is always the first column of the Previous Round Key plus an S-Box mapping of the previous $P_{(i+1)mod4,3} \oplus P_{(i+1)mod4,2}$ byte plus a Round constant.

   For example, in Table 3(a), $P_{0,0}$ EXORs with $S(P_{1,3} \oplus P_{1,2})$ and the Round constant $0x10$ to deduce the value in Table 3(b).

6. The second column of a Round Key is always the second column of the earlier $P^{k+2}_{(i,1)}$ Round Key plus the new *expression* added for the $P^{k+1}_{(i,0)}$ Round Key. Thus alternate Round Keys have similar deducible second columns.

   For example, in Tables 3(a), 3(b) and 3(c), $P_{0,1}$ EXORs with the new added *expression* $S(P_{1,3} \oplus P_{1,2} \oplus 0x10$ in Table 2(b) to give the new byte in Table 2(c).

7. The third column of a Round Key is always the EXOR of the third column and the first column of the earlier $P^{k+2}_{(i,2)}$ Round Key. Thus, the first and the third columns of Round Keys deduce the third column of a new Round Key, two rounds earlier.

   For example, in Table 3(b), the EXOR of the first column - $P_{0,0} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$ with the third column - $P_{0,2} \oplus P_{0,1}$ yields the third column bytes in Table 3(d).

8. The last column of a Round Key is always the EXOR of all the columns of the earlier $P^{k+3}_{(i,2)}$ Round Key. Thus, all the columns of a Round Key deduce the last column of a new Round Key, three rounds earlier.

   For example, in Table 3(b), the EXOR of the first column - $P_{0,0} \oplus S(P_{1,3} \oplus P_{1,2}) \oplus 0x10$ and the second column - $P_{0,1} \oplus P_{0,0}$ and the third column - $P_{0,2} \oplus P_{0,1}$ with the last column - $P_{0,3} \oplus P_{0,2}$ yields the last column bytes in Table 3(e).

4.3 Importance in Cryptanalysis

The discussed properties suggest that the Key schedule is strong but there are glaring differences between the Key Expansion and the Inverse Key Expansion processes. A cryptanalyst considering the less complex properties and easier properties of the Inverse Key Expansion might be tempted to use a Chosen or a Known Ciphertext attack and use the relevant Inverse Key properties to break the cipher.

However, the deduced Round Keys by the Inverse process are subjected to an *InvMixColumn* when EXORed with State values. This is only under the assumption that one uses the normal *InvByteSub*, *InvShiftRow*, *InvMixColumn* and *AddRoundKey* order of application for the decryption process in a round. If one wishes to do decrypting in exactly the reverse way, without considering the order of those steps, then there is no need to apply the *InvMixColumn* to the Round Key and it is easier to use the mentioned properties in a bid to cryptanalyse the AES.

In the context of a case, wherein a cryptanalyst knows the plaintext-ciphertext pairs (chosen plaintext or chosen ciphertext attack), a mathematical equation can be formulated on similar lines as discussed in *Section 2.4.3*. State values after 5 rounds using the known plaintext P can be written as

$$
\begin{aligned}
X = & K^5 \oplus MixColumn(ShiftRow(ByteSub(\\
& K^4 \oplus MixColumn(ShiftRow(ByteSub(\\
& K^3 \oplus MixColumn(ShiftRow(ByteSub(\\
& K^2 \oplus MixColumn(ShiftRow(ByteSub(\\
& K^1 \oplus MixColumn(ShiftRow(ByteSub(P \oplus K^0)))))))))))))))
\end{aligned}
$$

where K are the Round Key bytes in Table 2. Similar State values are also obtained using the known ciphertext C and decrypting it in an inverse format. Note that in this case, the *InvMixColumn* needs not be applied. Using Round Key values from Table 3, we can represent State values for Round 5 as

$$\begin{aligned} X = & K^5 \oplus InvByteSub(InvShiftRow(InvMixColumn(\\ & K^6 \oplus InvByteSub(InvShiftRow(InvMixColumn(\\ & K^7 \oplus InvByteSub(InvShiftRow(InvMixColumn(\\ & K^8 \oplus InvByteSub(InvShiftRow(InvMixColumn(\\ & K^9 \oplus InvByteSub(InvShiftRow(C \oplus K^{10}))))))))))))))) \end{aligned}$$

Both the above equations deduce the same value X, result in 2^{35} terms on each side and is very complex. Since the Round Keys are all interdependent on each other, as seen by our representations, an effective strategy to solve such an equation can be formulated. Presently, there exists no mechanism to do it. However, more research could definitely reap dividends.

It thus remains to be seen, if any of these properties can help lead to an attack better than an exhaustive 16 byte key search.

Chapter 5

Representation of the AES using Differential Trails

The following chapter is an original contribution and discusses an approach in representing the AES using differential analysis. The first section introduces the differential S-Boxes and their respective couplets. The second discusses how such boxes represent mappings of the entire AES cipher, whereas the third explores the implications of this process.

5.1 The Differential S-Boxes

The AES S-Box is a 16x16 hexadecimal box, with each row and column representing the upper nibble and the lower nibble respectively of the byte required to be substituted. The byte value at a particular row and column replaces this known byte.

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

Figure 7: The AES S-Box

Figure 7 shows the AES S-Box. For example, according to the figure, '01' is always replaced with '7C' after application of the S-Box. Thus, an S-Box can be simply represented as $O = S(I)$, where O is the output byte obtained by replacing the input byte I on the application of the AES S-Box $S()$.

Now, we represent a *Differential S-Box* as $O = S_d(I_1, I_2)$, where O is the output difference[1] obtained by the S-Box application of two input bytes I_1 and I_2 whose difference is d. In simple terms, $S_d(I_1, I_2) = S(I_1) \oplus S(I_2)$, such that $d = I_1 \oplus I_2$, where $S_d()$ is the Differential S-Box.

Since d is a byte, it ranges from 0 to 255. This gives us 256 Differential S-Boxes. Figure 8 demonstrates the making of one such Differential S-Box for $d =' 01'$.

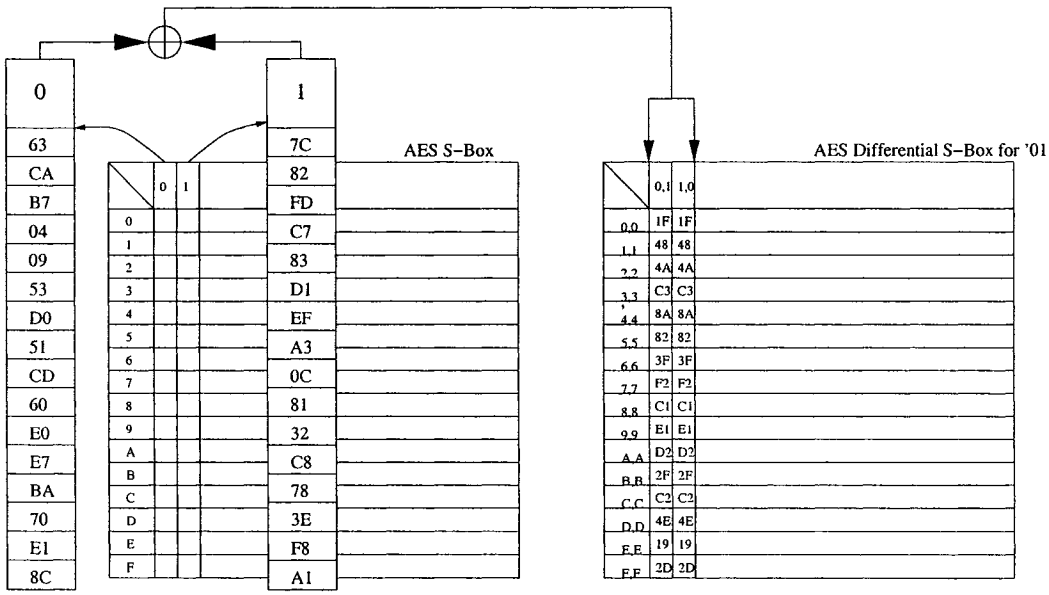

Figure 8: The AES Differential S-Box for '01'

$S_{'01'}()$ is derived from the $S()$ such that the input difference is $'01'$. The first two columns of $S()$ hold for input bytes from $'00'$ and $'01'$ to $'F0'$ and $'F1'$, whose difference is $'01'$ and whose EXOR yields the first two columns of $S_{'01'}()$. The rows in $S_{'01'}()$ represent the upper nibble of both the input bytes, while the lower nibble is for the columns. Thus, $63 \oplus 7C$ returns $1F$ for the row $0,0$ and the columns $0,1$ and $1,0$. Other column values for $S_{'01'}()$ can be similarly generated.

A pseudocode C notation to generate $S_d()$ can be written as:

[1]Difference is in the form of a simple EXOR

```
DiffSB(byte ByteSub[256],byte d)
{
    byte DiffByteSub[256];
    For( i = 0; i < 256; i++ )
    { For( j = 0; j < 256; j++ )
        { If( i ∧ j == d)
            {
                DiffByteSub[i] = ByteSub[i] ∧ ByteSub[j];
            }
        }
    }
}
```

Each Differential S-Box is atmost bound to contain a pair of 128 distinct byte values. From the above example, $'1F'$ is seen in two locations. This means that a byte value will not feature in all the available Differential S-Boxes.

In fact, tests were conducted to search a byte value in all the possible Differential S-Boxes. It was observed that each byte value is present in 256 locations. It is atleast seen twice in most of the Differential S-Boxes. For example, a search of $'1F'$ returns its presence

$\Rightarrow$ in $S_{'01'}()$ for $'00'$ and $'01'$,
$\Rightarrow$ in $S_{'01'}()$ for $'01'$ and $'00'$,
$\Rightarrow$ in $S_{'01'}()$ for $'BC'$ and $'BD'$,
$\Rightarrow$ in $S_{'01'}()$ for $'BD'$ and $'BC'$,
$\Rightarrow$ in $S_{'03'}()$ for $'81'$ and $'82'$,
$\Rightarrow$ in $S_{'03'}()$ for $'82'$ and $'81'$,
$\Rightarrow$ in $S_{'05'}()$ for $'9B'$ and $'9E'$,
$\Rightarrow$ in $S_{'05'}()$ for $'9E'$ and $'9B'$, and 248 other locations.

Each Differential S-Box $S_d()$ consists of a couplet of similar Differential S-Boxes $S'_d()$ and $S''_d()$. The variant $S'_d()$ is obtained from $S_d()$ by multiplying each of its byte value by $'02'$, whereas such a multiplication by $'03'$ on $S_d()$ results in $S''_d()$. Multiplication of a byte with $'02'$ is simply a left shift by 1 bit, while multiplication of a byte with $'03'$ is the addition[1] of the byte with the 1-bit left shift of the byte.

[1]Idea similar to *MixColumn*. It acts as a normal EXOR. A conditional byte EXOR with '1B' may occur depending on the most signifi cant bit of the byte

The significance of $S'_d()$ and $S''_d()$ is its similar functionality as that of the *MixColumn*. Its relevance will be seen in the next section.

The set of 256 Differential S-Boxes along with their individual couplets of $S'_d()$ and $S''_d()$ renders in all 768 Differential S-Box tables.

5.2 The AES Differential Mapping Representation

The following section attempts to serve a dual purpose. It creates a methodology to represent the AES in the form of mapping tables. At the same time, the process is a form of a differential analysis and explores the differential trail followed by the cipher due to small byte changes in the original plaintext/ciphertext. Assume the AES with a 128-bit key.

The reader is encouraged to consult **Appendix A** at the end, which details the process through an instance. The concepts explained below are complicated and the use of the example in the appendix simplifies the understanding.

The following terminology is considered:

$P_{i,j}$ $\rightarrow$ The original plaintext $\forall i, j \in \{0...3\}$

$C_{i,j}$ $\rightarrow$ The resulting plaintext $\forall i, j \in \{0...3\}$

$P^r_{i,j}$ $\rightarrow$ The intermediate State values at the end of round r where $r \in \{0...10\}$.

$P^{r-BS}_{i,j}$ $\rightarrow$ The intermediate State values at the end of the *ByteSub* in round r.

$P^{r-SR}_{i,j}$ $\rightarrow$ The intermediate State values at the end of the *ShiftRow* in round r.

$P^{r-MC}_{i,j}$ $\rightarrow$ The intermediate State values at the end of the *MixColumn* in round r.

Consider the case when the last bit of the byte $P_{0,0}$ is toggled. This means, that the new plaintext byte $P'_{0,0}$ differs from the old by $'01'$. Then, the corresponding notation is

$P'_{i,j}$ $\rightarrow$ The tweaked new plaintext $\forall i, j \in \{0...3\}$

$C'_{i,j}$ $\rightarrow$ The resulting plaintext $\forall i, j \in \{0...3\}$

$P'_{i,j}{}^{r}$ $\rightarrow$ The intermediate State values at the end of round r where $r \in \{0...10\}$.

$P'_{i,j}{}^{r-BS}$ $\rightarrow$ The intermediate State values at the end of the *ByteSub* in round r.

$P'_{i,j}{}^{r-SR}$ $\rightarrow$ The intermediate State values at the end of the *ShiftRow* in round r.

$P'_{i,j}{}^{r-MC}$ $\rightarrow$ The intermediate State values at the end of the *MixColumn* in round r.

As a result, we can say the difference $P_{0,0} \oplus P'_{0,0} =' 01'$. The next step is the original Cipher Key or the Round 0 Key addition. However, this does not disturb the difference between the two bytes. Hence, $P^0_{0,0} \oplus P'_{0,0}{}^{0} =' 01'$.

Round 1:

Now, the start of the first round introduces *ByteSub* on both the bytes. The difference $P_{0,0}^{1-BS} \oplus {P'_{0,0}}^{1-BS}$ is not $'01'$ anymore and here comes the significance of the Differential S-Boxes derived in the earlier section. The initial difference before *ByteSub* being $'01'$, the Differential S-Box table $S_{'01'}()$ is utilized where the row and the column stand for the upper and the lower nibbles respectively of the two bytes $P_{0,0}^{0}$ and ${P'_{0,0}}^{0}$. For example, from Figure 8, if $P_{0,0}^{0} =' 20'$ and ${P'_{0,0}}^{0} =' 21'$, then from $S_{'01'}()$, the difference after the S-Box application is $P_{0,0}^{1-BS} \oplus {P'_{0,0}}^{1-BS} =' 4A'$. Thus, depending on the difference at the end of the *ByteSub*, it is possible to retain or remember the values before the *ByteSub*. We name the $S_{'01'}()$ mapping table as *1-P(0,0)-AfterBS*.

The *ShiftRow*, $P_{0,0}^{1-SR} \oplus {P'_{0,0}}^{1-SR}$ does not affect the difference value nor does it affect the position of the byte because of the change occuring in the top row. The mapping table at this stage is named as *1-P(0,0)-AfterSR*.

The single byte difference spreads over to the entire column after the *MixColumn*. If we recall, the step involves multiplication of every column with the matrix

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Hence, a single byte difference in the first row before the *MixColumn* will mean

For the 1^{st} row byte:

$$\begin{aligned} P_{0,0}^{1-MC} \oplus {P'_{0,0}}^{1-MC} &= (P_{0,0}^{1-SR}).02 \oplus ({P'_{0,0}}^{1-SR}).02 \\ &= (P_{0,0}^{1-SR} \oplus {P'_{0,0}}^{1-SR}).02 \end{aligned}$$

For the 2^{nd} row byte:

$$\begin{aligned} P_{1,0}^{1-MC} \oplus {P'_{1,0}}^{1-MC} &= (P_{0,0}^{1-SR}).01 \oplus ({P'_{0,0}}^{1-SR}).01 \\ &= (P_{0,0}^{1-SR} \oplus {P'_{0,0}}^{1-SR}).01 \end{aligned}$$

For the 3^{rd} row byte:

$$\begin{aligned} P_{2,0}^{1-MC} \oplus {P'_{2,0}}^{1-MC} &= (P_{0,0}^{1-SR}).01 \oplus ({P'_{0,0}}^{1-SR}).01 \\ &= (P_{0,0}^{1-SR} \oplus {P'_{0,0}}^{1-SR}).01 \end{aligned}$$

For the 4^{th} row byte:

$$\begin{aligned} P_{3,0}^{1-MC} \oplus {P'_{3,0}}^{1-MC} &= (P_{0,0}^{1-SR}).03 \oplus ({P'_{0,0}}^{1-SR}).03 \\ &= (P_{0,0}^{1-SR} \oplus {P'_{0,0}}^{1-SR}).03 \end{aligned}$$

Here is where the the variants of the Differential S-Boxes, $S'_d()$ and $S''_d()$ come into picture. $S'_d()$ and $S''_d()$ represent the Differential S-Box tables when $S_d()$ is multiplied by $'02'$ and $'03'$ respectively. Therefore,

$P_{0,0}^{1-MC} \oplus {P'_{0,0}}^{1-MC}$ uses $S'_{'01'}()$ for mapping.
$P_{1,0}^{1-MC} \oplus {P'_{1,0}}^{1-MC}$ uses $S_{'01'}()$ for mapping.
$P_{2,0}^{1-MC} \oplus {P'_{2,0}}^{1-MC}$ uses $S_{'01'}()$ for mapping.
$P_{3,0}^{1-MC} \oplus {P'_{3,0}}^{1-MC}$ uses $S''_{'01'}()$ for mapping.

Note that by using these variant Differential S-Boxes, the differences in the newly modified bytes at positions {1,0}, {2,0} and {3,0} all trail back to $P_{0,0}^{0}$ and ${P'_{0,0}}^{0}$ at the start of the round. The corresponding mapping tables are named as *1-P(0,0)-AfterMC*, *1-P(1,0)-AfterMC*, *1-P(2,0)-AfterMC* and *1-P(3,0)-AfterMC*.

The *AddRoundKey* at this stage does not affect the difference in the four affected bytes. No mapping tables are created, because *AddRoundKey* never influences nor changes the position of differential bytes.

At the end of the first round, the four mapping tables helps remember the two difference bytes at the start of the round.

Round 2:

The *ByteSub* affects the differences of bytes at positions {0,0}, {1,0}, {2,0} and {3,0}. The differences seen in the *MixColumn*, which were retained for the *AddRoundKey* for the four bytes will result in new differences. A mapping is required to retain or remember the byte values and their differences at the end of the 1^{st} round.

So, a Differential S-Box $S_d()$ is developed for each of the four bytes, where d is the difference at the end of the 1^{st} round. Such an $S_d()$ will contain the new difference at the end of the *ByteSub*. For example, if $P_{0,0}^{1-MC} \oplus {P'_{0,0}}^{1-MC} =' E3'$ at the end of the first round, and if $P_{0,0}^{2-BS} \oplus {P'_{0,0}}^{2-BS} =' 1F'$ after application of *ByteSub*, then $S_{'E3'}()$ is created and it will definitely contain $'1F'$. The position of $'1F'$ will point out $P_{0,0}^{1}$ and ${P'_{0,0}}^{1}$.

For the four bytes, we can say

$P_{0,0}^{1-MC} \oplus {P'_{0,0}}^{1-MC}$ maps $P_{0,0}^{2-BS} \oplus {P'_{0,0}}^{2-BS}$ at $P_{0,0}^{1}$ and ${P'_{0,0}}^{1}$
$P_{1,0}^{1-MC} \oplus {P'_{1,0}}^{1-MC}$ maps $P_{1,0}^{2-BS} \oplus {P'_{1,0}}^{2-BS}$ at $P_{1,0}^{1}$ and ${P'_{1,0}}^{1}$
$P_{2,0}^{1-MC} \oplus {P'_{2,0}}^{1-MC}$ maps $P_{2,0}^{2-BS} \oplus {P'_{2,0}}^{2-BS}$ at $P_{2,0}^{1}$ and ${P'_{2,0}}^{1}$
$P_{3,0}^{1-MC} \oplus {P'_{3,0}}^{1-MC}$ maps $P_{3,0}^{2-BS} \oplus {P'_{3,0}}^{2-BS}$ at $P_{3,0}^{1}$ and ${P'_{3,0}}^{1}$

The four mapping tables created will be *2-P(0,0)-AfterBS*, *2-P(1,0)-AfterBS*, *2-P(2,0)-AfterBS* and *2-P(3,0)-AfterBS*.

The *ShiftRow* moves the affected bytes, but does not modify the difference. At the end of this step,

$$
\begin{aligned}
P_{0,0}^{2-SR} \oplus {P'_{0,0}}^{2-SR} &= P_{0,0}^{2-BS} \oplus {P'_{0,0}}^{2-BS} \\
P_{1,3}^{2-SR} \oplus {P'_{1,3}}^{2-SR} &= P_{1,0}^{2-BS} \oplus {P'_{1,0}}^{2-BS} \\
P_{2,2}^{2-SR} \oplus {P'_{2,2}}^{2-SR} &= P_{2,0}^{2-BS} \oplus {P'_{2,0}}^{2-BS} \\
P_{3,1}^{2-SR} \oplus {P'_{3,1}}^{2-SR} &= P_{3,0}^{2-BS} \oplus {P'_{3,0}}^{2-BS}
\end{aligned}
$$

The corresponding mapping tables are *2-P(0,0)-AfterSR*, *2-P(1,3)-AfterSR*, *2-P(2,2)-AfterSR* and *2-P(3,1)-AfterSR*.

The four differential bytes are all spread out in different columns and *MixColumn* spreads the differences to all the bytes of the column. Ineffect, the whole set of 16 bytes are affected by some difference.

Observing the matrix again, we get a set of 16 equations for all the bytes.

For the 1^{st} row and 1^{st} column byte:

$$
\begin{aligned}
P_{0,0}^{2-MC} \oplus {P'_{0,0}}^{2-MC} &= (P_{0,0}^{2-SR}).02 \oplus ({P'_{0,0}}^{2-SR}).02 \\
&= (P_{0,0}^{2-SR} \oplus {P'_{0,0}}^{2-SR}).02
\end{aligned}
$$

For the 2^{nd} row and 1^{st} column byte:

$$
\begin{aligned}
P_{1,0}^{2-MC} \oplus {P'_{1,0}}^{2-MC} &= (P_{0,0}^{2-SR}).01 \oplus ({P'_{0,0}}^{2-SR}).01 \\
&= (P_{0,0}^{2-SR} \oplus {P'_{0,0}}^{2-SR}).01
\end{aligned}
$$

For the 3^{rd} row and 1^{st} column byte:

$$
\begin{aligned}
P_{2,0}^{2-MC} \oplus {P'_{2,0}}^{2-MC} &= (P_{0,0}^{2-SR}).01 \oplus ({P'_{0,0}}^{2-SR}).01 \\
&= (P_{0,0}^{2-SR} \oplus {P'_{0,0}}^{2-SR}).01
\end{aligned}
$$

For the 4^{th} row and 1^{st} column byte:

$$
\begin{aligned}
P_{3,0}^{2-MC} \oplus {P'_{3,0}}^{2-MC} &= (P_{0,0}^{2-SR}).03 \oplus ({P'_{0,0}}^{2-SR}).03 \\
&= (P_{0,0}^{2-SR} \oplus {P'_{0,0}}^{2-SR}).03
\end{aligned}
$$

For the 1^{st} row and 2^{nd} column byte:

$$
\begin{aligned}
P_{0,1}^{2-MC} \oplus {P'_{0,1}}^{2-MC} &= (P_{3,1}^{2-SR}).01 \oplus ({P'_{3,1}}^{2-SR}).01 \\
&= (P_{3,1}^{2-SR} \oplus {P'_{3,1}}^{2-SR}).01
\end{aligned}
$$

For the 2^{nd} row and 2^{nd} column byte:

$$
\begin{aligned}
P_{1,1}^{2-MC} \oplus {P'_{1,1}}^{2-MC} &= (P_{3,1}^{2-SR}).01 \oplus ({P'_{3,1}}^{2-SR}).01 \\
&= (P_{3,1}^{2-SR} \oplus {P'_{3,1}}^{2-SR}).01
\end{aligned}
$$

For the 3^{rd} row and 2^{nd} column byte:

$$
\begin{aligned}
P_{2,1}^{2-MC} \oplus {P'_{2,1}}^{2-MC} &= (P_{3,1}^{2-SR}).03 \oplus ({P'_{3,1}}^{2-SR}).03 \\
&= (P_{3,1}^{2-SR} \oplus {P'_{3,1}}^{2-SR}).03
\end{aligned}
$$

For the 4^{th} row and 2^{nd} column byte:

$$P_{3,1}^{2-MC} \oplus {P'_{3,1}}^{2-MC} = (P_{3,1}^{2-SR}).02 \oplus ({P'_{3,1}}^{2-SR}).02$$
$$= (P_{3,1}^{2-SR} \oplus {P'_{3,1}}^{2-SR}).02$$

For the 1^{st} row and 3^{rd} column byte:

$$P_{0,2}^{2-MC} \oplus {P'_{0,2}}^{2-MC} = (P_{2,2}^{2-SR}).01 \oplus ({P'_{2,2}}^{2-SR}).01$$
$$= (P_{2,2}^{2-SR} \oplus {P'_{2,2}}^{2-SR}).01$$

For the 2^{nd} row and 3^{rd} column byte:

$$P_{1,2}^{2-MC} \oplus {P'_{1,2}}^{2-MC} = (P_{2,2}^{2-SR}).03 \oplus ({P'_{2,2}}^{2-SR}).03$$
$$= (P_{2,2}^{2-SR} \oplus {P'_{2,2}}^{2-SR}).03$$

For the 3^{rd} row and 3^{rd} column byte:

$$P_{2,2}^{2-MC} \oplus {P'_{2,2}}^{2-MC} = (P_{2,2}^{2-SR}).02 \oplus ({P'_{2,2}}^{2-SR}).02$$
$$= (P_{2,2}^{2-SR} \oplus {P'_{2,2}}^{2-SR}).02$$

For the 4^{th} row and 3^{rd} column byte:

$$P_{3,2}^{2-MC} \oplus {P'_{3,2}}^{2-MC} = (P_{2,2}^{2-SR}).01 \oplus ({P'_{2,2}}^{2-SR}).01$$
$$= (P_{2,2}^{2-SR} \oplus {P'_{2,2}}^{2-SR}).01$$

For the 1^{st} row and 4^{th} column byte:

$$P_{0,3}^{2-MC} \oplus {P'_{0,3}}^{2-MC} = (P_{1,3}^{2-SR}).03 \oplus ({P'_{1,3}}^{2-SR}).03$$
$$= (P_{1,3}^{2-SR} \oplus {P'_{1,3}}^{2-SR}).03$$

For the 2^{nd} row and 4^{th} column byte:

$$P_{1,3}^{2-MC} \oplus {P'_{1,3}}^{2-MC} = (P_{1,3}^{2-SR}).02 \oplus ({P'_{1,3}}^{2-SR}).02$$
$$= (P_{1,3}^{2-SR} \oplus {P'_{1,3}}^{2-SR}).02$$

For the 3^{rd} row and 4^{th} column byte:

$$P_{2,3}^{2-MC} \oplus {P'_{2,3}}^{2-MC} = (P_{1,3}^{2-SR}).01 \oplus ({P'_{1,3}}^{2-SR}).01$$
$$= (P_{1,3}^{2-SR} \oplus {P'_{1,3}}^{2-SR}).01$$

For the 4^{th} row and 4^{th} column byte:

$$P_{3,3}^{2-MC} \oplus {P'_{3,3}}^{2-MC} = (P_{1,3}^{2-SR}).01 \oplus ({P'_{1,3}}^{2-SR}).01$$
$$= (P_{1,3}^{2-SR} \oplus {P'_{1,3}}^{2-SR}).01$$

Using $S'_d()$ and $S''_d()$ for multiplications in '02' and '03', steps similar to Round 1 are performed. Appropriate mappings are taken care using the original $S_d()$ and these variants.

Similar to Round 1,

- Differences in the newly modified bytes at positions {1,0}, {2,0} and {3,0} all trail back to $P_{0,0}^{1}$ and ${P'_{0,0}}^{1}$ at the start of the round.

- Differences in the newly modified bytes at positions {0,1}, {1,1} and {2,1} all trail back to $P^1_{3,0}$ and ${P'_{3,0}}^1$ at the start of the round.
- Differences in the newly modified bytes at positions {0,2}, {1,2} and {3,2} all trail back to $P^1_{2,0}$ and ${P'_{2,0}}^1$ at the start of the round.
- Differences in the newly modified bytes at positions {0,3}, {2,3} and {3,3} all trail back to $P^1_{1,0}$ and ${P'_{1,0}}^1$ at the start of the round.

Sixteen mapping tables from *2-P(0,0)-AfterMC* to *2-P(3,3)-AfterMC* are created. The *AddRoundKey* does not affect the difference in all the affected sixteen bytes.

Round 3:

At the start of this round, all the sixteen bytes in P have differences with their corresponding values in P'. The application of a *ByteSub* affects all these bytes and operations similar to the one in Round 2 are performed.

For all $i, j \in \{0...3\}$,

$$P^{2-MC}_{i,j} \oplus {P'_{i,j}}^{2-MC} \quad \text{maps} \quad P^{3-BS}_{i,j} \oplus {P'_{i,j}}^{3-BS} \quad \text{at } P^2_{i,j} \text{ and } {P'_{i,j}}^2$$

As always, sixteen tables ranging from *3-P(0,0)-AfterBS* to *3-P(3,3)-AfterBS* are developed.

The *ShiftRow* positions all the 16 differential bytes. Thus,

$$\text{For the first row, } P^{3-SR}_{0,j} \oplus {P'_{0,j}}^{3-SR} = P^{3-BS}_{0,j} \oplus {P'_{0,j}}^{3-BS}$$

$$\text{For the second row, } P^{3-SR}_{1,j} \oplus {P'_{1,j}}^{3-SR} = P^{3-BS}_{1,(j+1)mod4} \oplus {P'_{1,(j+1)mod4}}^{3-BS}$$

$$\text{For the third row, } P^{3-SR}_{2,j} \oplus {P'_{2,j}}^{3-SR} = P^{3-BS}_{2,(j+2)mod4} \oplus {P'_{2,(j+2)mod4}}^{3-BS}$$

$$\text{For the fourth row, } P^{3-SR}_{3,j} \oplus {P'_{3,j}}^{3-SR} = P^{3-BS}_{3,(j+3)mod4} \oplus {P'_{3,(j+3)mod4}}^{3-BS}$$

The corresponding mapping tables are from *3-P(0,0)-AfterSR* to *3-P(3,3)-AfterSR*.

The *MixColumn* of Round 3 is complex, because at this stage, differences are observed in every byte of the State. To get the mapping of every byte at this level involves knowing the mappings of all bytes in that particular column. For example, to find the new difference at {0,0}, the following equation gives the desired mapping:

$$
\begin{aligned}
P_{0,0}^{3-MC} \oplus {P'_{0,0}}^{3-MC} &= (P_{0,0}^{3-SR}).02 \oplus ({P'_{0,0}}^{3-SR}).02 \\
&\quad (P_{0,1}^{3-SR}).03 \oplus ({P'_{0,1}}^{3-SR}).03 \\
&\quad (P_{0,2}^{3-SR}).01 \oplus ({P'_{0,2}}^{3-SR}).01 \\
&\quad (P_{0,3}^{3-SR}).01 \oplus ({P'_{0,3}}^{3-SR}).01 \\
&= (P_{0,0}^{3-SR} \oplus {P'_{0,0}}^{3-SR}).02 \\
&\quad (P_{0,1}^{3-SR} \oplus {P'_{0,1}}^{3-SR}).03 \\
&\quad (P_{0,2}^{3-SR} \oplus {P'_{0,2}}^{3-SR}).01 \\
&\quad (P_{0,3}^{3-SR} \oplus {P'_{0,3}}^{3-SR}).01
\end{aligned}
$$

Mappings are taken care of by using the $S_d()$ and its variants $S'_d()$ and $S''_d()$, similar to the earlier round. At this stage, it can be observed that differences in bytes at positions {i,j} trails back to $P_{i,j}^2$ and ${P'_{i,j}}^2$ at the start of that round. Also, sixteen new mapping tables are formed: *3-P(0,0)-AfterMC* to *3-P(3,3)-AfterMC*.

As before, *AddRoundKey* does not affect the differences in the bytes.

Rounds 4 to 10:

Since all the bytes in P and P' differ which is similar in Round 3, the forthcoming rounds work the same way as Round 3. Similar mapping tables are created and each mapping table preserves the earlier round State values. The trail continues one round after the other, ending in Round 10. Noteworthy is the absence of *MixColumn* in the final Round.

Figure 9 details the entire process. Each Round involves mappings from the start of the round till its end. The mappings are preserved until the end of the round, such that an inverse map leads back to the original byte values at the start of the round. It can be deduced that the *ShiftRow* spreads differences along the row, while the *MixColumn* does the same for the columns.

Round 3 contains 16 mapping tables each over three steps. Thus, Round 3 comprises 48 mapping tables. Rounds 4 to 9 are similar and thus contribute another 288 mapping tables. Rounds 1, 2 and 10 add 6, 24 and 32 mapping tables respectively. In all, 398 mapping tables are required.

The representation can be extended for the same plaintext and the key. Instead of applying a difference of $'01'$ at the beginning, any difference d is acceptable, utilized to

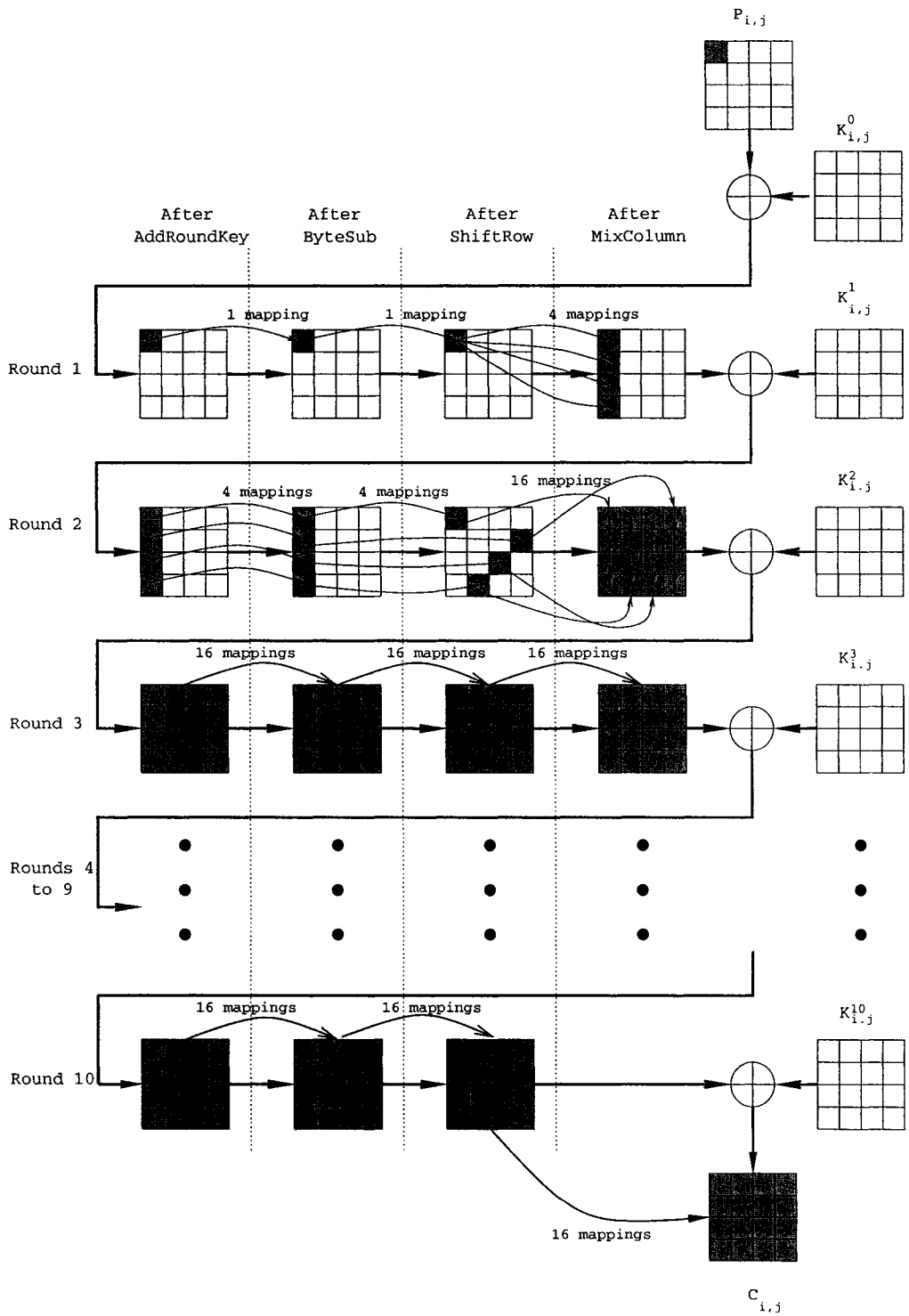

Figure 9: The AES Representation using Mapping tables

build the $S_d()$ or *1-P{0,0}-AfterBS*. Also, instead of modifying only $P_{0,0}$, any of the 16 bytes can be changed by any amount to create the varied mapping tables.

A similar process is achievable by performing the analysis in the reverse order. By varying the ciphertext bytes, similar mapping tables can be created and an inverse differential trail will be established. To avoid the *InvMixColumn* in case of the key, an exactly reverse cipher order is maintained.

5.3 Relevance of the representation

The most important observation is the absence of the effect of Round Keys on the difference cipher. At the end of the *AddRoundKey* step, the same differences are preserved as seen before the step. Though this observation might seem very trivial, such a differential process is ideal against attackers who have access to any two consecutive State values in the AES cipher. If such a codebreaker gets access to consecutive State values after a *MixColumn* and an *AddRoundKey*, the Round key is easily recoverable. However when differences are used for representing the AES[1], knowledge of consecutive differential bytes after a *MixColumn* and an *AddRoundKey* does not help retrieve the Round Key. Despite the functionality of an *AddRoundKey* tending to be discreet, it contributes to the differences in the next step, *ByteSub*.

Based on the above observation, the use of such a representation can be extended by its efficacy against attackers who can have access to any two consecutive Round State values. If an attacker is able to retrieve end of Round n and end of Round $n+1$ State values, he can apply the *ByteSub*, *ShiftRow* and the *MixColumn* to the Round n State values and EXOR them with the Round $n+1$ State values to extract the Round Key.

However, knowledge of difference bytes at the end of Round n and the end of Round $n+1$ does not impart the Round Key extraction. For every byte difference d in Round n, an $S_d()$ is created, which will store 256 probable differences at the end of the *ByteSub*. The *ShiftRow* affects the position of these new differences and ultimately, the *MixColumn* utilizes four differences in the same column to calculate new differences. The meaning here is that for each of the 16 byte differences at the start, there are 256 values possible and with the interaction between the rows and the column in the *ShiftRow* and the *MixColumn*, the correct differences at the end have to be deduced for four columns with each column

[1]Extreme right matrices in the Appendix example

contributing 256×4 guesses. Hence, for the four columns, it takes $4 \times (256 \times 4)$ guesses. With the right difference values deduced for each $S_d()$, one can backtrack to get the intermediate State values of the cipher at the end of Round n. Nevertheless, this does not help retrieve the Round Key because the differential bytes at the end of *MixColumn* and the end of *AddRoundKey* or Round $n+1$ is the same. The intermediate State values realised can be applied the *ByteSub*, the *ShiftRow* and the *MixColumn*, but the *AddRoundKey* cannot be applied.

The exception to the above case is when the ciphertexts along with the Round 9 difference bytes are known. *MixColumn* is absent at this stage and $S_d()$ tables created at this stage directly reflect the differences in the two ciphertexts. Backtracking retrieves the intermediate State bytes and application of *ByteSub* and *ShiftRow* to them with a subsequent EXOR with the ciphertexts extracts the Round 10 Key.

Consideration should be given to a circumstance when large instances of differences are applied on the same byte of the plaintext or on all bytes of the plaintext. In other words, if a lot of test cases are generated using a plaintext with different byte changes under the presence of a single key, then there are chances to predict the difference for a particular byte change. Since the same Original Key and similar Round Keys are applied at every stage in all the test cases, a relation can be formulated in all the *AddRoundKey* steps. This will essentially require the Round Key properties inferred in the previous chapter. A methodology to develop and devise such relations in a difference analysis is an interesting future research problem. As an example, discarding certain probable values in new Round Keys, depending on earlier Round Keys, can help create *ByteSub* mapping tables with more probability of selecting a few *ByteSub* differences over other.

Finally, the AES can be represented using differential mapping tables. The process gives a fresh view of differential trails created by changing plaintext bytes.

Chapter 6

Conclusion and Future Work

There are three important issues about the AES discussed in the thesis.

Primary among them is the Last Round Differential attack, lately better known as the Differential Fault Analysis(DFA) attack. The attack exploits the weakness in the Last Round and is achieved with precise employment of appropriate bit toggling mechanisms. An extension for 8 rounds can still assist the attacker in acquiring the key. It is a very powerful attack and possible in smartcards. Smart-cards which can protect against fault intrusions can deter such attacks.

The next issue was attempting to evaluate the different properties present in the Key schedule. The Key schedule is complex and lack of sufficient research entailed representing it in a proper format. Since both the *KeyExpansion* and the *InvKeyExpansion* are different in the way their Round Keys are affiliated with each other, the 5^{th} Round Key was the ideal starting point for generation of other Round Keys in either directions. Quite a few properties were observed and realised. That along with a few notable differences in the Keys expanded through the *KeyExpansion* and the *InvKeyExpansion* processes.

The AES had always been represented in an algebraic form, which as a matter of fact is tough to solve in the present context. In an effort to represent the same at every step, a differential analysis was sought. With the right amount of memory, the entire cipher steps can be thought of as running through quite of number of mapping tables. In the process, an inverse map can lead back to the original plaintexts. The other purpose of such an analysis is an observation of differential trails. The AES might have been designed to thwart differential cryptanalysis but a practical study on the same was essential. The relevance of doing this also hinders a few attacks possible by the knowledge of a few

consecutive State values in the normal cipher.

The thesis overall examines the up-to-date literature on the AES along with three major concepts and their analyses. It can also be concluded in general that the AES is inherently very strong and no shortcut attacks are possible in the present situation.

The AES is a substantial future research problem. Being used by the entire world for most of the encryption, it has to be subjected to every possible analysis, despite quite a lot being done during its selection.

A vital research problem is the ability to use the key properties as a means to crypt-analyse the cipher. Solving equations involving terms collected in Round 5 after forward and inverse cipher operations on plaintexts and ciphertexts respectively is a hard problem. No current mechanism exists to solve such a problem and the properties derived can prove useful.

These derived properties might also be of help when applied to a differential analysis. The derived AES representation does attempt to hide the Round Key addition. However, the properties of the Key bytes and their affiliations in such a representation can yield interesting results. More research is sought.

Finally, the Last Round Differential/DFA attack was extended upto 8 rounds. More research can aid in attacking earlier rounds. Also, appropriate hardware systems have to be researched and built to prevent fault intrusions.

Bibliography

[AES01] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[BB97] Eli Biham and Alex Biryukov. An Improvement of Davies' Attack on DES. *Journal of Cryptology*, 10(3):195–206, 1997.

[BDL97] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology — EUROCRYPT '97*, pages 37–51. Springer, 1997.

[Bih93] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. In *Advances in Cryptology, EUROCRYPT 93*, pages 398–409. Springer-Verlag, 1993.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology — CRYPTO '97*, pages 513–525. Springer, 1997.

[Cam03] Peter J. Cameron. Galois fields, The Encyclopedia of Design Theory, 2003. http://www.designtheory.org/library/encyc/topics/gf.pdf.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Asiacrypt 2002*, pages 267–287. Springer, 2002.

[Dae95] John Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. PhD thesis, K.U.Leuven, March 1995.

[DH77] Whitfield Diffie and Martin Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 10(6):74–84, June 1977.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher SQUARE. *Lecture Notes in Computer Science*, 1267:149–165, 1997.

[DM95] Donald Davies and Sean Murphy. Pairs and Triplets of DES S-Boxes. *Journal of Cryptology*, 8(1):1–25, 1995.

[DR99] John Daemen and Vincent Rijmen. *AES Proposal: Rijndael*, September 1999. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[DR00] 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.

[EFF98] DES Cracker, 1998. http://www.eff.org/DEScracker/.

[FKL+00] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved Cryptanalysis of Rijndael. In *Seventh Fast Software Encryption Workshop*, page 19. Springer-Verlag, 2000.

[FM02] Joanne Fuller and William Millan. On linear redundancy in the AES S-box, 2002.

[FSW01] Niels Ferguson, Richard Shroeppel, and Doug Whiting. A simple algebraic representation of Rijndael. In *Selected Areas in Cryptography*, pages 103–111. Springer-Verlag, December 2001.

[Gir04] Christophe Giraud. DFA on AES. In *Fourth Conference on the Advanced Encryption Standard (AES)," AES - State of the Crypto Analysis"*, 2004.

[GM00] Henri Gilbert and Marine Minier. A collision attack on 7 rounds of rijndael. In *Third AES Candidate Conference, AES3*, pages 230–241, New York, 2000.

[HMS+76] M. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, and P. Schweitzer. *Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard*. Department of Electrical Engineering, Stanford University, 1976. Technical Report SEL 76-042.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The Interpolation Attack on Block Ciphers. *Lecture Notes in Computer Science*, 1267:28+, 1997.

[Kah96] David Kahn. *The CodeBreakers*. Scribner, New York, 1996.

[Knu95] Lars R. Knudsen. Truncated and Higher Order Differentials. In *Fast Software Encryption*, pages 196–211, 1995.

[KSW96] John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptanalysis of 3-WAY, IDEA, G-DES, RC4, SAFER, and Triple-DES. In *Advances in Cryptology, CRYPTO 96*, pages 237–251. Springer-Verlag, August 1996.

[Lau03] Niels Lauritzen. *Concrete Abstract Algebra:From Numbers to Grobner Bases*. Cambridge University Press, 2003.

[LN86] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986.

[Luc00] Stefan Lucks. Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In *Third AES Candidate Conference, AES3*, pages 215–229, New York, 2000.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES Cipher. In *EUROCRYPT 93*, pages 386–397, 1993.

[MR00a] Sean Murphy and Matt Robshaw. Further Comments on the Structure of Rijndael. Available at http://www.isg.rhul.ac.uk/ sean/Response.pdf, 17 August 2000.

[MR00b] Sean Murphy and Matt Robshaw. New Observations on Rijndael. Available at http://isg.rhbnc.ac.uk/ mrobshaw/rijndael/rijndael.pdf, 7 August 2000.

[MR02] Sean Murphy and Matt Robshaw. Essential Algebraic Structure within the AES. In *CRYPTO 2002*, pages 1–16, 2002.

[MS87] J.H. Moore and G.J. Simmons. Cycle structure of the DES with Weak and Semi-Weak keys. In *Advances in Cryptology — CRYPTO '86*, pages 3–32. Springer-Verlag, 1987.

[Nat77] 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.

[NIS01] National Institute of Standards and Technology, 2001. http://csrc.nist.gov/CryptoToolkit/aes/.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In *Advances in Cryptology, EUROCRYPT 93*, pages 55–64. Springer-Verlag, 1994.

[RD02] Vincent Rijmen and Joan Daemen. *The Design of Rijndael: AES - the Advanced Encryption Standard.* Springer-Verlag, Berlin, 2002.

[SB91] Miles Smid and Dennis Branstad. *Contemporary Cryptology*, chapter The Data Encryption Standard: Past and Future. IEEE Press, 1991.

[Wie93] Michael Wiener. Efficient DES key search. In *Rump session — CRYPTO '93*, 1993. Also available in Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.

Appendix A

Test Case of an AES differential trail using Mapping Tables

Plaintext = 0x3243F6A8885A308D313198A2E0370734

Key = 0x2B7E151628AED2A6ABF7158809CF4F3C

Initial Cipher Key

$K_{i,j}$

$$\begin{bmatrix} 2B & 28 & AB & 09 \\ 7E & AE & F7 & CF \\ 15 & D2 & 15 & 4F \\ 16 & A6 & 88 & 3C \end{bmatrix}$$

	$P_{i,j}$	$P'_{i,j}$	$P_{i,j} \oplus P'_{i,j}$
Initial plaintexts	$\begin{bmatrix} 32 & 88 & 31 & E0 \\ 43 & 5A & 31 & 37 \\ F6 & 30 & 98 & 07 \\ A8 & 8D & A2 & 34 \end{bmatrix}$	$\begin{bmatrix} 33 & 88 & 31 & E0 \\ 43 & 5A & 31 & 37 \\ F6 & 30 & 98 & 07 \\ A8 & 8D & A2 & 34 \end{bmatrix}$	$\begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$
AddRoundKey	$\begin{bmatrix} 19 & A0 & 9A & E9 \\ 3D & F4 & C6 & F8 \\ E3 & E2 & 8D & 48 \\ BE & 2B & 2A & 08 \end{bmatrix}$	$\begin{bmatrix} 18 & A0 & 9A & E9 \\ 3D & F4 & C6 & F8 \\ E3 & E2 & 8D & 48 \\ BE & 2B & 2A & 08 \end{bmatrix}$	$\begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$

	$P^1_{i,j}$	$P'^1_{i,j}$	$P^1_{i,j} \oplus P'^1_{i,j}$
Round 1 *ByteSub*. Creation of mapping table *1-P{0,0}-AfterBS* with a return trail to '19' and '18'.	$\begin{bmatrix} D4 & E0 & B8 & 1E \\ 27 & BF & B4 & 41 \\ 11 & 98 & 5D & 52 \\ AE & F1 & E5 & 30 \end{bmatrix}$	$\begin{bmatrix} AD & E0 & B8 & 1E \\ 27 & BF & B4 & 41 \\ 11 & 98 & 5D & 52 \\ AE & F1 & E5 & 30 \end{bmatrix}$	$\begin{bmatrix} 79 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$
Round 1 *ShiftRow*. Creation of mapping table *1-P{0,0}-AfterSR* with a return trail to '19' and '18'.	$\begin{bmatrix} D4 & E0 & B8 & 1E \\ BF & B4 & 41 & 27 \\ 5D & 52 & 11 & 98 \\ F1 & E5 & 30 & AE \end{bmatrix}$	$\begin{bmatrix} AD & E0 & B8 & 1E \\ BF & B4 & 41 & 27 \\ 5D & 52 & 11 & 98 \\ F1 & E5 & 30 & AE \end{bmatrix}$	$\begin{bmatrix} 79 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$
Round 1 *MixColumn*. Creation of four mapping tables *1-P{i,0}-AfterMC*, $i \in \{0...3\}$ with each returning the trail to '19' and '18'.	$\begin{bmatrix} 04 & E0 & 48 & 28 \\ 66 & CB & F8 & 06 \\ 81 & 19 & D3 & 26 \\ E5 & 9A & 7A & 4C \end{bmatrix}$	$\begin{bmatrix} F6 & E0 & 48 & 28 \\ 1F & CB & F8 & 06 \\ F8 & 19 & D3 & 26 \\ 6E & 9A & 7A & 4C \end{bmatrix}$	$\begin{bmatrix} F2 & 00 & 00 & 00 \\ 79 & 00 & 00 & 00 \\ 79 & 00 & 00 & 00 \\ 8B & 00 & 00 & 00 \end{bmatrix}$

First Round Key

$$K^1_{i,j}$$

$$\begin{bmatrix} A0 & 88 & 23 & 2A \\ FA & 54 & A3 & 6C \\ FE & 2C & 39 & 76 \\ 17 & B1 & 39 & 05 \end{bmatrix}$$

Round 1 *AddRoundKey*.	$\begin{bmatrix} A4 & 68 & 6B & 02 \\ 9C & 9F & 5B & 6A \\ 7F & 35 & EA & 50 \\ F2 & 2B & 43 & 49 \end{bmatrix}$	$\begin{bmatrix} 56 & 68 & 6B & 02 \\ E5 & 9F & 5B & 6A \\ 06 & 35 & EA & 50 \\ 79 & 2B & 43 & 49 \end{bmatrix}$	$\begin{bmatrix} F2 & 00 & 00 & 00 \\ 79 & 00 & 00 & 00 \\ 79 & 00 & 00 & 00 \\ 8B & 00 & 00 & 00 \end{bmatrix}$

	$P^2_{i,j}$	$P'^2_{i,j}$	$P^2_{i,j} \oplus P'^2_{i,j}$
Round 2 *ByteSub*. Creation of four mapping tables *2-P{i,0}-AfterBS*, $i \in \{0...3\}$ with four corresponding return trails to '*A4*' and '*56*' for $i = 0$,...etc.	$\begin{bmatrix} 49 & 45 & 7F & 77 \\ DE & DB & 39 & 02 \\ D2 & 96 & 87 & 53 \\ 89 & F1 & 1A & 3B \end{bmatrix}$	$\begin{bmatrix} B1 & 45 & 7F & 77 \\ D9 & DB & 39 & 02 \\ 6F & 96 & 87 & 53 \\ B6 & F1 & 1A & 3B \end{bmatrix}$	$\begin{bmatrix} F8 & 00 & 00 & 00 \\ 07 & 00 & 00 & 00 \\ BD & 00 & 00 & 00 \\ 3F & 00 & 00 & 00 \end{bmatrix}$
Round 2 *ShiftRow*. Creation of four mapping tables *2-P{i,j}-AfterSR*, for {i,j} as {0,0},{1,3},{2,2} or {3,1} with return trails as above.	$\begin{bmatrix} 49 & 45 & 7F & 77 \\ DB & 39 & 02 & DE \\ 87 & 53 & D2 & 96 \\ 3B & 89 & F1 & 1A \end{bmatrix}$	$\begin{bmatrix} B1 & 45 & 75 & 77 \\ DB & 39 & 02 & D9 \\ 87 & 53 & 6F & 96 \\ 3B & B6 & F1 & 1A \end{bmatrix}$	$\begin{bmatrix} F8 & 00 & 00 & 00 \\ 00 & 00 & 00 & 07 \\ 00 & 00 & BD & 00 \\ 00 & 3F & 00 & 00 \end{bmatrix}$
Round 2 *MixColumn*. Creation of 16 mapping tables *2-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with four corresponding return trails as above.	$\begin{bmatrix} 58 & 1B & DB & 1B \\ 4D & 4B & E7 & 6B \\ CA & 5A & CA & B0 \\ F1 & AC & A8 & E5 \end{bmatrix}$	$\begin{bmatrix} B3 & 24 & 66 & 12 \\ B5 & 74 & 3B & 65 \\ 32 & 1B & AB & B7 \\ E2 & D2 & 15 & E2 \end{bmatrix}$	$\begin{bmatrix} EB & 3F & BD & 09 \\ F8 & 3F & DC & 0E \\ F8 & 41 & 61 & 07 \\ 13 & 7E & BD & 07 \end{bmatrix}$

Second Round Key $K^2_{i,j}$

$$\begin{bmatrix} F2 & 7A & 59 & 73 \\ C2 & 96 & 35 & 59 \\ 95 & B9 & 80 & F6 \\ F2 & 43 & 7A & 7F \end{bmatrix}$$

Round 2 *AddRoundKey*.	$\begin{bmatrix} AA & 61 & 82 & 68 \\ 8F & DD & D2 & 32 \\ 5F & E3 & 4A & 46 \\ 03 & EF & D2 & 9A \end{bmatrix}$	$\begin{bmatrix} 41 & 5E & 3F & 61 \\ 77 & E2 & 0E & 3C \\ A7 & A2 & 2B & 41 \\ 10 & 91 & 6F & 9D \end{bmatrix}$	$\begin{bmatrix} EB & 3F & BD & 09 \\ F8 & 3F & DC & 0E \\ F8 & 41 & 61 & 07 \\ 13 & 7E & BD & 07 \end{bmatrix}$

	$P^3_{i,j}$	$P'^3_{i,j}$	$P^3_{i,j} \oplus P'^3_{i,j}$
Round 3 *ByteSub*. Creation of 16 mapping tables *3-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '*AA* and '41' for $\{i, j\}$ as $\{0,0\}$,...etc.	$\begin{bmatrix} AC & EF & 13 & 45 \\ 73 & C1 & B5 & 23 \\ CF & 11 & D6 & 5A \\ 7B & DF & BF & B8 \end{bmatrix}$	$\begin{bmatrix} 83 & 58 & 75 & EF \\ F5 & 98 & AB & EB \\ 5C & 3A & F1 & 83 \\ CA & 81 & A8 & 5E \end{bmatrix}$	$\begin{bmatrix} 2F & B7 & 66 & AA \\ 86 & 59 & 1E & C8 \\ 93 & 2B & 27 & D9 \\ B1 & 5E & 1D & E6 \end{bmatrix}$
Round 3 *ShiftRow*. Creation of 16 mapping tables *3-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} AC & EF & 13 & 45 \\ C1 & B5 & 23 & 73 \\ D6 & 5A & CF & 11 \\ B8 & 7B & DF & BF \end{bmatrix}$	$\begin{bmatrix} 83 & 58 & 75 & EF \\ 98 & AB & EB & F5 \\ F1 & 83 & 5C & 3A \\ 5E & CA & 81 & A8 \end{bmatrix}$	$\begin{bmatrix} 2F & B7 & 66 & AA \\ 59 & 1E & C8 & 86 \\ 27 & D9 & 93 & 2B \\ E6 & B1 & 5E & 1D \end{bmatrix}$
Round 3 *MixColumn*. Creation of 16 mapping tables *3-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 75 & 20 & 53 & BB \\ EC & 0B & C0 & 25 \\ 09 & 63 & CF & D0 \\ 93 & 33 & 7C & DC \end{bmatrix}$	$\begin{bmatrix} 01 & 1F & 11 & 53 \\ FE & 41 & DD & F8 \\ 00 & AB & BE & 8D \\ 4B & 4F & 31 & AE \end{bmatrix}$	$\begin{bmatrix} 74 & 3F & 42 & E8 \\ 12 & 4A & 1D & DD \\ 09 & C8 & 71 & 5D \\ D8 & 7C & 4D & 72 \end{bmatrix}$
Third Round Key	$K^3_{i,j}$		
	$\begin{bmatrix} 3D & 47 & 1E & 6D \\ 80 & 16 & 23 & 7A \\ 47 & FE & 7E & 88 \\ 7D & 3E & 44 & 3B \end{bmatrix}$		
Round 3 *AddRoundKey*.	$\begin{bmatrix} 48 & 67 & 4D & D6 \\ 6C & 1D & E3 & 5F \\ 4E & 9D & B1 & 58 \\ EE & 0D & 38 & E7 \end{bmatrix}$	$\begin{bmatrix} 3C & 58 & 0F & 3E \\ 7E & 57 & FE & 82 \\ 47 & 55 & C0 & 05 \\ 36 & 71 & 75 & 95 \end{bmatrix}$	$\begin{bmatrix} 74 & 3F & 42 & E8 \\ 12 & 4A & 1D & DD \\ 09 & C8 & 71 & 5D \\ D8 & 7C & 4D & 72 \end{bmatrix}$

	$P^4_{i,j}$	$P'^4_{i,j}$	$P^4_{i,j} \oplus P'^4_{i,j}$
Round 4 *ByteSub*. Creation of 16 mapping tables *4-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '48 and '3C' for $\{i, j\}$ as $\{0,0\}$,...etc.	$\begin{bmatrix} 52 & 85 & E3 & F6 \\ 50 & A4 & 11 & CF \\ 2F & 5E & C8 & 6A \\ 28 & D7 & 07 & 94 \end{bmatrix}$	$\begin{bmatrix} EB & 6A & 76 & B2 \\ F3 & 5B & BB & 13 \\ A0 & FC & BA & 6B \\ 05 & A3 & 9D & 2A \end{bmatrix}$	$\begin{bmatrix} B9 & EF & 95 & 44 \\ A3 & FF & AA & DC \\ 8F & A2 & 72 & 01 \\ 2D & 74 & 9A & BE \end{bmatrix}$
Round 4 *ShiftRow*. Creation of 16 mapping tables *4-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 52 & 85 & E3 & F6 \\ A4 & 11 & CF & 50 \\ C8 & 6A & 2F & 5E \\ 94 & 28 & D7 & 07 \end{bmatrix}$	$\begin{bmatrix} EB & 6A & 76 & B2 \\ 5B & BB & 13 & F3 \\ BA & 6B & A0 & FC \\ 2A & 05 & A3 & 9D \end{bmatrix}$	$\begin{bmatrix} B9 & EF & 95 & 44 \\ FF & AA & DC & A3 \\ 72 & 01 & 8F & A2 \\ BE & 2D & 74 & 9A \end{bmatrix}$
Round 4 *MixColumn*. Creation of 16 mapping tables *4-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 0F & 60 & 6F & 5E \\ D6 & 31 & C0 & B3 \\ DA & 38 & 10 & 13 \\ A9 & BF & 6B & 01 \end{bmatrix}$	$\begin{bmatrix} B0 & 6C & DA & 10 \\ A2 & BF & 08 & CD \\ A1 & 08 & C0 & 1E \\ 93 & 64 & 74 & E3 \end{bmatrix}$	$\begin{bmatrix} BF & 0C & B5 & 4E \\ 74 & 8E & C8 & 7E \\ 7B & 30 & D0 & 0D \\ 3A & DB & 1F & E2 \end{bmatrix}$
Fourth Round Key	$K^4_{i,j}$		
	$\begin{bmatrix} EF & A8 & B6 & DB \\ 44 & 52 & 71 & 0B \\ A5 & 5B & 25 & AD \\ 41 & 7F & 3B & 00 \end{bmatrix}$		
Round 4 *AddRoundKey*.	$\begin{bmatrix} E0 & C8 & D9 & 85 \\ 92 & 63 & B1 & B8 \\ 7F & 63 & 35 & BE \\ E8 & C0 & 50 & 01 \end{bmatrix}$	$\begin{bmatrix} 5F & C4 & 6C & CB \\ E6 & ED & 79 & C6 \\ 04 & 53 & E5 & B3 \\ D2 & 1B & 4F & E3 \end{bmatrix}$	$\begin{bmatrix} BF & 0C & B5 & 4E \\ 74 & 8E & C8 & 7E \\ 7B & 30 & D0 & 0D \\ 3A & DB & 1F & E2 \end{bmatrix}$

	$P^5_{i,j}$	$P'^5_{i,j}$	$P^5_{i,j} \oplus P'^5_{i,j}$
Round 5 *ByteSub*. Creation of 16 mapping tables *5-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '*E*0 and '*5F*' for $\{i, j\}$ as $\{0,0\}$,...etc.	$\begin{bmatrix} E1 & E8 & 35 & 97 \\ 4F & FB & C8 & 6C \\ D2 & FB & 96 & AE \\ 9B & BA & 53 & 7C \end{bmatrix}$	$\begin{bmatrix} CF & 1C & 50 & 1F \\ 8E & 55 & B6 & B4 \\ F2 & ED & D9 & 6D \\ B5 & AF & 84 & 11 \end{bmatrix}$	$\begin{bmatrix} 2E & F4 & 65 & 88 \\ C1 & AE & 7E & D8 \\ 20 & 16 & 4F & C3 \\ 2E & 15 & D7 & 6D \end{bmatrix}$
Round 5 *ShiftRow*. Creation of 16 mapping tables *5-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} E1 & E8 & 35 & 97 \\ FB & C8 & 6C & 4F \\ 96 & AE & D2 & FB \\ 7C & 9B & BA & 53 \end{bmatrix}$	$\begin{bmatrix} CF & 1C & 50 & 1F \\ 55 & B6 & B4 & 8E \\ D9 & 6D & F2 & ED \\ 11 & B5 & AF & 84 \end{bmatrix}$	$\begin{bmatrix} 2E & F4 & 65 & 88 \\ AE & 7E & D8 & C1 \\ 4F & C3 & 20 & 16 \\ 6D & 2E & 15 & D7 \end{bmatrix}$
Round 5 *MixColumn*. Creation of 16 mapping tables *5-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 25 & BD & B6 & 4C \\ D1 & 11 & 3A & 4C \\ A9 & D1 & 33 & C0 \\ AD & 68 & 8E & B0 \end{bmatrix}$	$\begin{bmatrix} B2 & 21 & 3A & DE \\ 04 & 69 & 81 & B0 \\ 00 & B4 & F1 & C7 \\ E4 & 8E & F3 & 51 \end{bmatrix}$	$\begin{bmatrix} 97 & 9C & 8C & 92 \\ D5 & 78 & BB & FC \\ A9 & 65 & C2 & 07 \\ 49 & E6 & 7D & E1 \end{bmatrix}$

Fifth Round Key $K^5_{i,j}$

$$\begin{bmatrix} D4 & 7C & CA & 11 \\ D1 & 83 & F2 & F9 \\ C6 & 9D & B8 & 15 \\ F8 & 87 & BC & BC \end{bmatrix}$$

Round 5 *AddRoundKey*.	$\begin{bmatrix} F1 & C1 & 7C & 5D \\ 00 & 92 & C8 & B5 \\ 6F & 4C & 8B & D5 \\ 55 & EF & 32 & 0C \end{bmatrix}$	$\begin{bmatrix} 66 & 5D & F0 & CF \\ D5 & EA & 73 & 49 \\ C6 & 29 & 49 & D2 \\ 1C & 09 & 4F & ED \end{bmatrix}$	$\begin{bmatrix} 97 & 9C & 8C & 92 \\ D5 & 78 & BB & FC \\ A9 & 65 & C2 & 07 \\ 49 & E6 & 7D & E1 \end{bmatrix}$

	$P^{6}_{i,j}$	$P'^{6}_{i,j}$	$P^{6}_{i,j} \oplus P'^{6}_{i,j}$
Round 6 *ByteSub*. Creation of 16 mapping tables *6-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '$F1$ and '66' for $\{i, j\}$ as $\{0,0\}$,...etc.	$\begin{bmatrix} A1 & 78 & 10 & 4C \\ 63 & 4F & E8 & D5 \\ A8 & 29 & 3D & 03 \\ FC & DF & 23 & FE \end{bmatrix}$	$\begin{bmatrix} 33 & 4C & 8C & 8A \\ 03 & 87 & 8F & 3B \\ B4 & A5 & 3B & B5 \\ 9C & 01 & 84 & 55 \end{bmatrix}$	$\begin{bmatrix} 92 & 34 & 9C & C6 \\ 60 & C8 & 67 & EE \\ 1C & 8C & 06 & B6 \\ 60 & DE & A7 & AB \end{bmatrix}$
Round 6 *ShiftRow*. Creation of 16 mapping tables *6-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} A1 & 78 & 10 & 4C \\ 4F & E8 & D5 & 63 \\ 3D & 03 & A8 & 29 \\ FE & FC & DF & 23 \end{bmatrix}$	$\begin{bmatrix} 33 & 4C & 8C & 8A \\ 87 & 8F & 3B & 03 \\ 3B & B5 & B4 & A5 \\ 55 & 9C & 01 & 84 \end{bmatrix}$	$\begin{bmatrix} 92 & 34 & 9C & C6 \\ C8 & 67 & EE & 60 \\ 06 & B6 & 1C & 8C \\ AB & 60 & DE & A7 \end{bmatrix}$
Round 6 *MixColumn*. Creation of 16 mapping tables *6-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 4B & 2C & 33 & 37 \\ 86 & 4A & 9D & D2 \\ 8D & 89 & F4 & 18 \\ 6D & 80 & E8 & D8 \end{bmatrix}$	$\begin{bmatrix} 9A & 3B & FB & 2B \\ 3E & 11 & 3C & FC \\ 3D & 0D & C7 & 4F \\ 43 & CD & 02 & 30 \end{bmatrix}$	$\begin{bmatrix} D1 & 17 & C8 & 1C \\ B8 & 5B & A1 & 2E \\ B0 & 84 & 33 & 57 \\ 2E & 4D & EA & E8 \end{bmatrix}$

Sixth Round Key $K^{6}_{i,j}$

$$\begin{bmatrix} 6D & 11 & DB & CA \\ 88 & 0B & F9 & 00 \\ A3 & 3E & 86 & 93 \\ 7A & FD & 41 & FD \end{bmatrix}$$

Round 6 *AddRoundKey*.	$\begin{bmatrix} 26 & 3D & E8 & FD \\ 0E & 41 & 64 & D2 \\ 2E & B7 & 72 & 8B \\ 17 & 7D & A9 & 25 \end{bmatrix}$	$\begin{bmatrix} F7 & 2A & 20 & E1 \\ B6 & 1A & C5 & FC \\ 9E & 33 & 41 & DC \\ 39 & 30 & 43 & CD \end{bmatrix}$	$\begin{bmatrix} D1 & 17 & C8 & 1C \\ B8 & 5B & A1 & 2E \\ B0 & 84 & 33 & 57 \\ 2E & 4D & EA & E8 \end{bmatrix}$

Round 7 *ByteSub*.
Creation of 16 mapping tables *7-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '26 and '*F7*' for $\{i, j\}$ as $\{0,0\}$,...etc.

$P^7_{i,j}$

$$\begin{bmatrix} F7 & 27 & 9B & 54 \\ AB & 83 & 43 & B5 \\ 31 & A9 & 40 & 3D \\ F0 & FF & D3 & 3F \end{bmatrix}$$

$P'^7_{i,j}$

$$\begin{bmatrix} 68 & E5 & B7 & F8 \\ 4E & A2 & A6 & B0 \\ 0B & C3 & 83 & 86 \\ 12 & 04 & 1A & BD \end{bmatrix}$$

$P^7_{i,j} \oplus P'^7_{i,j}$

$$\begin{bmatrix} 9F & C2 & 2C & AC \\ E5 & 21 & E5 & 05 \\ 3A & 6A & C3 & BB \\ E2 & FB & C9 & 82 \end{bmatrix}$$

Round 7 *ShiftRow*.
Creation of 16 mapping tables *7-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.

$$\begin{bmatrix} F7 & 27 & 9B & 54 \\ 83 & 43 & B5 & AB \\ 40 & 3D & 31 & A9 \\ 3F & F0 & FF & D3 \end{bmatrix}$$

$$\begin{bmatrix} 68 & E5 & B7 & F8 \\ A2 & A6 & B0 & 4E \\ 83 & 86 & 0B & C3 \\ BD & 12 & 04 & 1A \end{bmatrix}$$

$$\begin{bmatrix} 9F & C2 & 2C & AC \\ 21 & E5 & 05 & E5 \\ C3 & BB & 3A & 6A \\ 82 & E2 & FB & C9 \end{bmatrix}$$

Round 7 *MixColumn*.
Creation of 16 mapping tables *7-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.

$$\begin{bmatrix} 14 & 46 & 27 & 34 \\ 15 & 16 & 46 & 2A \\ B5 & 15 & 56 & D8 \\ BF & EC & D7 & 43 \end{bmatrix}$$

$$\begin{bmatrix} 13 & B4 & B1 & E0 \\ 14 & 31 & D5 & 20 \\ 0B & 62 & 1D & 05 \\ F8 & 30 & 71 & AA \end{bmatrix}$$

$$\begin{bmatrix} 07 & F2 & 96 & D4 \\ 01 & 27 & 93 & 0A \\ BE & 77 & 4B & DD \\ 47 & DC & A6 & E9 \end{bmatrix}$$

Seventh Round Key

$K^7_{i,j}$

$$\begin{bmatrix} 4E & 5F & 84 & 4E \\ 54 & 5F & A6 & A6 \\ F7 & C9 & 4F & DC \\ 0E & F3 & B2 & 4F \end{bmatrix}$$

Round 7 *AddRoundKey*.

$$\begin{bmatrix} 5A & 19 & A3 & 7A \\ 41 & 49 & E0 & 8C \\ 42 & DC & 19 & 04 \\ B1 & 1F & 65 & 0C \end{bmatrix}$$

$$\begin{bmatrix} 5D & EB & 35 & AE \\ 40 & 6E & 73 & 86 \\ FC & AB & 52 & D9 \\ F6 & C3 & C3 & E5 \end{bmatrix}$$

$$\begin{bmatrix} 07 & F2 & 96 & D4 \\ 01 & 27 & 93 & 0A \\ BE & 77 & 4B & DD \\ 47 & DC & A6 & E9 \end{bmatrix}$$

	$P^8_{i,j}$	$P'^8_{i,j}$	$P^8_{i,j} \oplus P'^8_{i,j}$
Round 8 *ByteSub*. Creation of 16 mapping tables *8-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '5A and '5D' for $\{i, j\}$ as $\{0,0\}$,...etc.	$\begin{bmatrix} BE & D4 & 0A & DA \\ 83 & 3B & E1 & 64 \\ 2C & 86 & D4 & F2 \\ C8 & C0 & 4D & FE \end{bmatrix}$	$\begin{bmatrix} 4C & E9 & 96 & E4 \\ 09 & 9F & 8F & 44 \\ B0 & 62 & 00 & 35 \\ 42 & 2E & 2E & D9 \end{bmatrix}$	$\begin{bmatrix} F2 & 3D & 9C & 3E \\ 8A & A4 & 6E & 20 \\ 9C & E4 & D4 & C7 \\ 8A & EE & 63 & 27 \end{bmatrix}$
Round 8 *ShiftRow*. Creation of 16 mapping tables *8-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} BE & D4 & 0A & DA \\ 3B & E1 & 64 & 83 \\ D4 & F2 & 2C & 86 \\ FE & C8 & C0 & 4D \end{bmatrix}$	$\begin{bmatrix} 4C & E9 & 96 & E4 \\ 9F & 8F & 44 & 09 \\ 00 & 35 & B0 & 62 \\ D9 & 42 & 2E & 2E \end{bmatrix}$	$\begin{bmatrix} F2 & 3D & 9C & 3E \\ A4 & 6E & 20 & 8A \\ D4 & C7 & 9C & E4 \\ 27 & 8A & EE & 63 \end{bmatrix}$
Round 8 *MixColumn*. Creation of 16 mapping tables *8-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 00 & B1 & 54 & FA \\ 51 & C8 & 76 & 1B \\ 2F & 89 & 6D & 99 \\ D1 & FF & CD & EA \end{bmatrix}$	$\begin{bmatrix} FB & 34 & 65 & 84 \\ B0 & F1 & FB & 7E \\ A3 & CA & DB & 5B \\ E2 & 1E & 09 & 00 \end{bmatrix}$	$\begin{bmatrix} FB & 85 & 31 & 7E \\ E1 & 39 & 8D & 65 \\ 8C & 43 & B6 & C2 \\ 33 & E1 & C4 & EA \end{bmatrix}$

Eighth Round Key $K^8_{i,j}$

$$\begin{bmatrix} EA & B5 & 31 & 7F \\ D2 & 8D & 2B & 8D \\ 73 & BA & F5 & 29 \\ 21 & D2 & 60 & 2F \end{bmatrix}$$

Round 8 *AddRoundKey*.	$\begin{bmatrix} EA & 04 & 65 & 85 \\ 83 & 45 & 5D & 96 \\ 5C & 33 & 98 & B0 \\ F0 & 2D & AD & C5 \end{bmatrix}$	$\begin{bmatrix} 11 & 81 & 54 & FB \\ 62 & 7C & D0 & F3 \\ D0 & 70 & 2E & 72 \\ C3 & CC & 69 & 2F \end{bmatrix}$	$\begin{bmatrix} FB & 85 & 31 & 7E \\ E1 & 39 & 8D & 65 \\ 8C & 43 & B6 & C2 \\ 33 & E1 & C4 & EA \end{bmatrix}$

	$P^9_{i,j}$	$P'^9_{i,j}$	$P^9_{i,j} \oplus P'^9_{i,j}$
Round 9 *ByteSub*. Creation of 16 mapping tables *9-P{i,j}-AfterBS*, $i, j \in \{0...3\}$ with 16 corresponding return trails to '*EA* and '11' for $\{i, j\}$ as {0,0},...etc.	$\begin{bmatrix} 87 & F2 & 4D & 97 \\ EC & 6E & 4C & 90 \\ 4A & C3 & 46 & E7 \\ 8C & D8 & 95 & A6 \end{bmatrix}$	$\begin{bmatrix} 82 & 0C & 20 & 0F \\ AA & 10 & 70 & 0D \\ 70 & 51 & 31 & 40 \\ 2E & 4B & F9 & 15 \end{bmatrix}$	$\begin{bmatrix} 05 & FE & 6D & 98 \\ 46 & 7E & 3C & 9D \\ 3A & 92 & 77 & A7 \\ A2 & 93 & 6C & B3 \end{bmatrix}$
Round 9 *ShiftRow*. Creation of 16 mapping tables *9-P{i,j}-AfterSR*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 87 & F2 & 4D & 97 \\ 6E & 4C & 90 & EC \\ 46 & E7 & 4A & C3 \\ A6 & 8C & D8 & 95 \end{bmatrix}$	$\begin{bmatrix} 82 & 0C & 20 & 0F \\ 10 & 70 & 0D & AA \\ 31 & 40 & 70 & 51 \\ 15 & 2E & 4B & F9 \end{bmatrix}$	$\begin{bmatrix} 05 & FE & 6D & 98 \\ 7E & 3C & 9D & 46 \\ 77 & A7 & 3A & 92 \\ B3 & A2 & 93 & 6C \end{bmatrix}$
Round 9 *MixColumn*. Creation of 16 mapping tables *9-P{i,j}-AfterMC*, $i, j \in \{0...3\}$ with 16 corresponding return trails as above.	$\begin{bmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{bmatrix}$	$\begin{bmatrix} 0B & E6 & 6C & 53 \\ E4 & 02 & E1 & 4A \\ CF & 8E & 10 & 17 \\ 96 & 78 & 8B & 03 \end{bmatrix}$	$\begin{bmatrix} 4C & A6 & CF & 1F \\ D3 & D6 & 91 & D5 \\ 5B & 6A & 2A & 55 \\ 7B & DD & 2D & BF \end{bmatrix}$
Ninth Round Key	$K^9_{i,j}$		
	$\begin{bmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ F3 & 21 & 41 & 6E \end{bmatrix}$		
Round 9 *AddRoundKey*.	$\begin{bmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D2 \end{bmatrix}$	$\begin{bmatrix} A7 & FF & 44 & 04 \\ 93 & F8 & 30 & 16 \\ A9 & 52 & 39 & 17 \\ 65 & 59 & CA & 6D \end{bmatrix}$	$\begin{bmatrix} 4C & A6 & CF & 1F \\ D3 & D6 & 91 & D5 \\ 5B & 6A & 2A & 55 \\ 7B & DD & 2D & BF \end{bmatrix}$

Round 10 *ByteSub*.

Creation of 16 mapping tables *10-P{i,j}-AfterBS*, $i,j \in \{0...3\}$ with 16 corresponding return trails to *'EB* and *'A7'* for $\{i,j\}$ as $\{0,0\}$,...etc.

$P_{i,j}^{10}$

$$\begin{bmatrix} E9 & CB & 3D & AF \\ 09 & 31 & 32 & 2E \\ 89 & 07 & 7D & 2C \\ 72 & 5F & 94 & B5 \end{bmatrix}$$

$P'^{10}_{i,j}$

$$\begin{bmatrix} 5C & 16 & 1B & F2 \\ DC & 41 & 04 & 47 \\ D3 & 00 & 12 & F0 \\ 4D & CB & 74 & 3C \end{bmatrix}$$

$P_{i,j}^{10} \oplus P'^{10}_{i,j}$

$$\begin{bmatrix} B5 & DD & 26 & 5D \\ D5 & 70 & 36 & 69 \\ 5A & 07 & 6F & DC \\ 3F & 94 & E0 & 89 \end{bmatrix}$$

Round 10 *ShiftRow*.

Creation of 16 mapping tables *10-P{i,j}-AfterSR*, $i,j \in \{0...3\}$ with 16 corresponding return trails as above.

$$\begin{bmatrix} E9 & CB & 3D & AF \\ 31 & 32 & 2E & 09 \\ 7D & 2C & 89 & 07 \\ B5 & 72 & 5F & 94 \end{bmatrix}$$

$$\begin{bmatrix} 5C & 16 & 1B & F2 \\ 41 & 04 & 47 & DC \\ 12 & F0 & D3 & 00 \\ 3C & 4D & CB & 74 \end{bmatrix}$$

$$\begin{bmatrix} B5 & DD & 26 & 5D \\ 70 & 36 & 69 & D5 \\ 6F & DC & 5A & 07 \\ 89 & 3F & 94 & E0 \end{bmatrix}$$

Tenth Round Key

$K_{i,j}^{10}$

$$\begin{bmatrix} D0 & C9 & E1 & B6 \\ 14 & EE & 3F & 63 \\ F9 & 25 & 0C & 0C \\ A8 & 89 & C8 & A6 \end{bmatrix}$$

Round 10 *AddRoundKey*.

$$\begin{bmatrix} 39 & 02 & DC & 19 \\ 25 & DC & 11 & 6A \\ 84 & 09 & 85 & 0B \\ 1D & FB & 97 & 32 \end{bmatrix}$$

$$\begin{bmatrix} 8C & DF & FA & 44 \\ 55 & EA & 78 & BF \\ EB & D5 & DF & 0C \\ 94 & C4 & 03 & D2 \end{bmatrix}$$

$$\begin{bmatrix} B5 & DD & 26 & 5D \\ 70 & 36 & 69 & D5 \\ 6F & DC & 5A & 07 \\ 89 & 3F & 94 & E0 \end{bmatrix}$$

$C_{i,j}^{10}$

$C'^{10}_{i,j}$

$C_{i,j}^{10} \oplus C'^{10}_{i,j}$