適用於無線感知網路的 Modified AES
Modified AES for Wireless Sensor Networks

Shen-Wei Chen
Dept. Electrical Engineering,
National Tsing Hua University
mr923969@ee.nthu.edu.tw

Wei-Chi Ting
Dept. Computer Science,
National Tsing Hua University
mr936303@cs.nthu.edu.tw

Hung-Min Sun
Dept. Computer Science,
National Tsing Hua University
hmsun@cs.nthu.edu.tw

Yarsun Hsu
Dept. Electrical Engineering,
National Tsing Hua University
yshsu@ee.nthu.edu.tw

摘要

無線感知節點的主要限制包括計算量，電力消耗和體積。與有線網路所不同的是，任何攻擊者皆能竊聽與發送偽造訊息，因此重要的訊息應該被加密傳輸。為了提供無線感知網路的資訊安全方案，我們簡化AES的演算法，不同的行混合參數以提高資料混合效率。位元組取代和行混合運算合併成單一查詢表，加解密一共減少 512 位元組的儲存空間。我們也使用GF(2^4)^2 代替計算GF(2^8)反函數的技術以節省面積。FPGA實作結果顯示資料傳輸速率可達 1.6Gps，能契合無線感知網路的需求。

關鍵字：AES, 無線感知網路, 加密

Abstract

The major constraints of wireless sensor nodes are computational costs, communication power and size. Different from wired networks, any adversary can receive and transmit fabricated data. Hence, important messages through the public channel should be encrypted. To provide data security over WSN, standard AES is too bulky for WSN. In this work, we proposed a variant of AES which is suitable for WSN by choosing different parameters for MixColumns to achieve higher data diffusion rate. SubBytes and MixColumns transformations are combined into a look-up table, which reduces 512 bytes space in total. In addition, we implement InvSubBytes and SubWord with composite field calculation by using GF(2^4)^2 inverter. Our throughput can reach 1.6 Gps using FPGA simulation, and is suitable for WSN transmission.

Keyword： AES, WSN, encryption

This work is supported by both NSC and MOEA under grant numbers 94-2213-E-007-040, 94-EC-17-A-04-S1-044

1 Introduction

Wireless Sensor Networks (WSN) come with the rapidly development of Wireless Network Technologies in the recent years. They are composed of many sensor nodes that gather data and transmit to the base station. Since they are usually used in military applications, their most important requirements are security, efficiency and size [8].

Encryption algorithms have been well developed over the last decade, e.g. AES [18], BLOWFISH [5], RC5 [26], RC6 [24], DES [19], TEA [7], TwoFish [4], RSA [22], etc. Considering computational cost, which is one of the bottlenecks in wireless sensor networks, exponential computation like RSA requires too much energy. We suggest symmetric cryptosystems rather than asymmetric ones. In particular, AES is the most famous and popular symmetric encryption which is believed to meet the security needs in WSN. However, AES has two main problems in low-computational power devices like sensor. First, MixColumns require polynomial multiplications which are costly for sensors. Secondly, unlike the symmetric structure of DES, its encryption and decryption components are almost different; they share only about 46% of area mostly at the part of Sbox and key expansion.

We propose a modified AES algorithm (MAES in brief) based on Rijndael with fewer rounds; different MixColumns coefficients to maintain its security to a certain level. We show that differential [11] and linear [17] attacks are infeasible in WSN environment. MAES is very efficient in encryption process where shared components minimize the area in hardware implementation.

The rest of the paper is organized as follows. We first review AES in section 2. The MAES algorithm is described in Section 3. In section 4, we discuss the way choosing coefficient for MixColumns and apply both linear and differential cryptanalysis. Section 5 provides an implementation of our cipher including performance comparison. Tables of SubMix transformation are given in Appendix A and test vectors are available in Appendix B.

2 Backgrounds

AES, stands for Advanced Encryption Standard, is a fast block cipher using symmetric key. The initiation of AES was announced by the National Institute of Standards and Technology (NIST) in January, 1997. After a series of evaluation, Rijndael [12][13] developed by Joan Daemen and Vincent Rijnment was selected by NIST as new encryption standard in October, 2000 [15][18]. AES is iterated with Boolean transformation applied to the plaintext block called *state*. Recently, various hardware implementations of AES showed good performance and suitable for wireless applications.

2.1 AES Polynomial Multiplication

Polynomial multiplication in AES corresponds with multiplications of polynomials modulo an irreducible polynomial of degree 8. By default setting, this irreducible polynomial is $m(x) = x^8 + x^4 + x^3 + x + 1$. For example, {57} • {83} = {C1}, where '•' denotes polynomial multiplication in GF(2^8).

First we have binary representation of {57} as $\{01010111\}_2$, which stands for $x^6 + x^4 + x^2 + x + 1$. Polynomial multiplication is computed by

$$\begin{aligned}&(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)\\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1\\ &= (x^7 + x^6 + 1) \bmod (x^8 + x^4 + x^3 + x + 1) \quad (1)\end{aligned}$$

One can easily verify that {57} • {01} = {57} where {01} is the identity. If given two polynomials and their multiplications over an irreducible polynomial equals to the identity element, for example {DE} • {90} = {01}, they are inverses of each other.

2.2 AES Equivalent Round

Each round of AES has an equivalent format which outputs exactly the same ciphertext; that is, we can change the order of SubBytes and ShiftRows. The standard AES round is depicted as follow:

```
AES-round ( ) {
        SubBytes (state);
        ShiftRows (state);
        MixColumns (state);
        AddRoundKey (state, RoundKey);
}
```

SubBytes transformation operates on each byte of current *state*, and ShiftRows only change their positions among rows. We can rearrange their order and have equivalent round:

```
Equivalent-round ( ) {
        ShiftRows (state);
        SubBytes (state);
        MixColumns (state);
        AddRoundKey (state, RoundKey);
}
```

Then, we merge SubBytes and MixColumns into a new transformation called SubMix. Thus, encryption requires only three stages.

```
MAES-round ( ) {
        ShiftRows (state);
        SubMix (state);
        AddRoundKey (state, RoundKey);
}
```

2.3 MixColumns Transformation

MixColumns substitutes and permutes data at the same time to achieve high diffusion property. During MixColumns transformation, columns are considered as polynomials over GF(2^8) and multiplied with a fixed polynomial $a(x)$ under x^4+1, given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2)$$

Let $S_{0,c}$ denotes the first byte of column c, $S'_{0,c}$ after MixColumns and $S_{2,c} \sim S_{3,c}$ respectively. Each column then multiplies the rotations of a(x). This can be written as a matrix multiplication.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (3)$$

In standard AES, coefficients of MixColumns are very simple, but more complicated of InvMixColumns, because AES was designed to be also efficient in software. The *xtime* operation means to multiply a polynomial *x* modulo *m*(*x*), and can be implemented with a shifter and bit-wised XOR. It is a very efficient way to compute polynomial multiplications when there are more zeros in binary representation of the operand, since we only need to shift left iteratively. Obviously, computational cost of {37} is much greater than that of {03} using *xtime* computations.

$$\begin{aligned}S'_{0,c} &= S_{0,c} \bullet \{02\} \oplus S_{1,c} \bullet \{03\}\\ &\oplus S_{2,c} \bullet \{01\} \oplus S_{3,c} \bullet \{01\} \quad (4)\end{aligned}$$

If we take a closer look, the portion of $S_{3,c} \bullet \{01\} = S_{3,c}$ doesn't contributes effort in mixing data if $S_{0,c}$, $S_{1,c}$, and $S_{2,c}$ are fixed to zeros, then output of this stage is exactly same to the input.

In contrast, MAES is designed for hardware where table look-up is the fastest way, and computing {37} and {03} are the same. Hence, we are allowed to choose these polynomials freely. However, a polynomial in $GF(2^8)[x]$ modulo $x^4 + 1$ doesn't guarantee to have inverse. Even if it has one, its bit-wised mapping may not be sufficiently complex (measured by hamming weight). We suggest using polynomial

$$p(x) = \{37\}x^3 + \{5D\}x^2 + \{17\}x + \{17\} \qquad (5)$$

and its inverse

$$p^{-1}(x) = \{4F\}x^3 + \{E4\}x^2 + \{E4\}x + \{DE\} \qquad (6)$$

They both have high hamming weights in form of bit-wised mapping. Numbers of these polynomials are quite few in the space of 2^{32} and we believe that our choice is good enough (may not be optimal), since there is no efficient way to compute inverses, we are not able to perform exhaustive search for optimal in $2^{32} \rightarrow 2^{32}$. The suggested p(x) and $p(x)^{-1}$ both have good mapping complexity, and their coefficients are three different values, which means they need only three tables, while conventional AES needs four.

3. The MAES Algorithm

In this section, we describe the specific algorithm of MAES, where the length of key, input block and output block are all 128 bits (16 bytes). As for ShiftRows and AddRoundKey transformations, they are defined as the standard AES algorithm. The cipher is reduced to 7 rounds. Key generation process is the same as in AES. Our thought is to do one more MixColumns transformation within the last round. Although this change doesn't gain more security, it does save spaces if we use SubMix instead. The SubMix transformation is implemented by three look-up tables (in hardware) instead of computing polynomial multiplications for efficiency concern.

We move computational costs from encryption onto decryption, because data broadcasted through radio might be easily corrupted and retransmission occurs very often. Besides, every message needs a message authentication code (called MAC) also computed by encryption algorithm to prevent fabricated data. One can see sensors usually do more encryption than decryption.

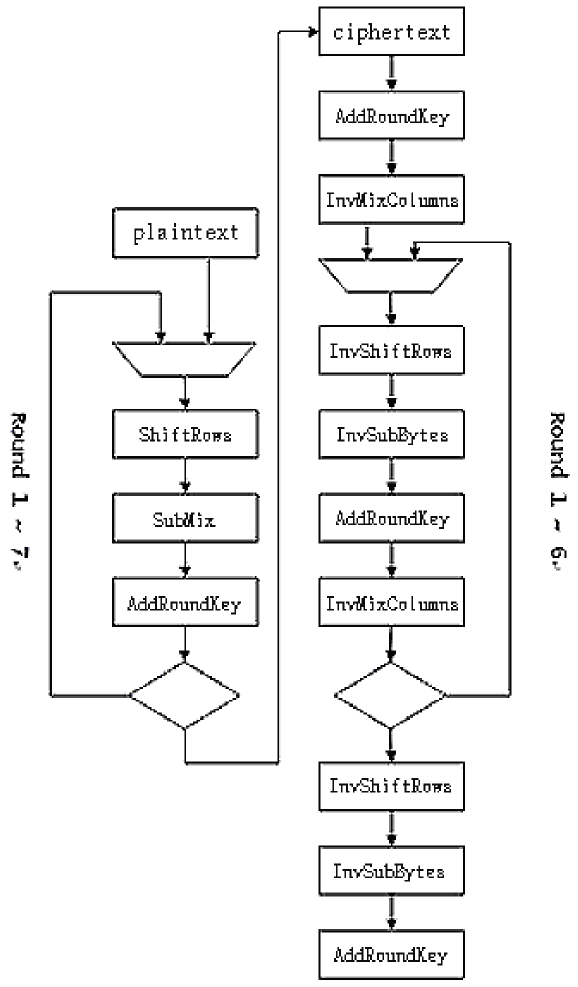


Figure 1 encryption/decryption process of MAES

3.1 Encryption

Encryption is very efficient, with only three stages: ShiftRows, SubMix and AddRoundKey. The cipher is described in pseudo code.

```
Nr = 7
Begin
    state = input
    AddRoundKey (state, RoundKey[first] )
    for round = 1 step 1 to Nr
        ShiftRows (state)
        SubMix (state)
        AddRoundKey(state, RoundKey[round])
    end for
    output = state
end
```

Figure 2 Pseudo Code for the cipher

3.2 SubMix Transformation

The SubMix transformation is a combination of SubBytes and MixColumns that takes four polynomials over GF(2^8) (shifted column) as input and substitutes them with different values by table look-up. The outputs of previous stage are XORed to obtain the final value. This can be written as a matrix multiplication like equation (3).

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 17 & 37 & 5D & 17 \\ 17 & 17 & 37 & 5D \\ 5D & 17 & 17 & 37 \\ 37 & 5D & 17 & 17 \end{bmatrix} \begin{bmatrix} \text{SubBytes}(S_{0,c}) \\ \text{SubBytes}(S_{1,c}) \\ \text{SubBytes}(S_{2,c}) \\ \text{SubBytes}(S_{3,c}) \end{bmatrix} \quad (7)$$

According to the Design of Rijndael [13], SubBytes and MixColumns have the following property:

$$\begin{aligned} &\text{SubBytes}(a(x)) \bullet b(x) \\ &= \text{SubBytes}(a(x) \bullet b(x)) \qquad (8) \end{aligned}$$

For example, if input $s(x) = \{01\} + \{02\}x + \{03\}x^2 + \{04\}x^3$, the value of $S_{0,c}$ after SubMix is computed by

$$\begin{aligned} S_{0,c}' &= \text{SubMix}(\{01\}, \{02\}, \{03\}, \{04\}) \\ &= \text{SubByte}(01) \bullet \{17\} \oplus \text{SubByte}(02) \bullet \{37\} \\ &\quad \oplus \text{SubByte}(03) \bullet \{5D\} \oplus \text{SubByte}(04) \bullet \{17\} \\ &= \text{SubByte}(01 \bullet 17) \oplus \text{SubByte}(02 \bullet 37) \\ &\quad \oplus \text{SubByte}(03 \bullet 5D) \oplus \text{SubByte}(04 \bullet 17) \\ &= \{17\} \oplus \{6E\} \oplus \{E7\} \oplus \{5C\} = \{C2\} \end{aligned}$$

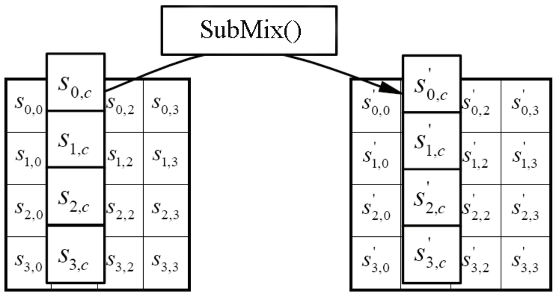

SubMix()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

Figure 3 Illustration of the SubMix transformation.

SubByte(a(x) • b(x)) can be realized in two different ways. One by look-up table, which is faster but requires larger space; the other is by bit-wised mapping that can be implemented by XOR gates [21], smaller but slower. To handle the tradeoff, we attempt to use look-up tables for SubMix.

First, if implemented by look-up tables, each of them requires 256 bytes in size (same as Sbox in AES), and we need both 3*256 bytes for encryption and decryption. Compared to AES, which needs one table for SubBytes, 3*256 bytes for MixColumns and 4*256 bytes for InvMixColumns, our result saves 512 bytes space in total. Table 1 shows a comparison of AES and our method.

Table 1 Comparison of AES and MAES

	AES	MAES
SubBytes	1 table	0 table
MixColumns / SubMix	3 tables	3 tables
InvSubBytes	1 table	1 table
InvMixColumns	4 tables	3 tables

Secondly, if implemented with bit-wised mapping, here is an example for computing multiplication of {DE} with input $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$ and outputs $a(x)$':

$$\begin{aligned} a_0' &= a_1 \oplus a_2 \oplus a_4 \oplus a_6 \\ a_1' &= a_0 \oplus a_1 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \\ a_2' &= a_0 \oplus a_1 \oplus a_2 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \\ a_3' &= a_0 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_7 \\ a_4' &= a_0 \oplus a_2 \oplus a_5 \\ a_5' &= a_1 \oplus a_3 \oplus a_6 \\ a_6' &= a_0 \oplus a_2 \oplus a_4 \oplus a_7 \\ a_7' &= a_0 \oplus a_1 \oplus a_3 \oplus a_5 \qquad (9) \end{aligned}$$

3.3 Decryption

The decryption process is slightly different, because we perform an extra MixColumns at the end of encryption. There needs an InvMixColumns right after the first step of decryption --- AddRoundKey to stay equivalence. The rest parts of decryption are exactly same to the standard AES. It is described in pseudo code in Fig 10.

```
Nr = 7
Begin
    state = input
    AddRoundKey(RoundKey[Last])
    InvMixColumns(state)
    For round=Nr-1 step -1 downto 1
        InvShiftRows (state)
        InvSubBytes(state)
        AddRoundKey (state, RoundKey[round])
        InvMixColumns(state)
    End for

    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, RoundKey[First]);
    output = state
End
```

Figure 4 Pseudo code for the inverse cipher

4. Security Analysis

AES was designed to stand against both linear and differential attacks. Linear attack was introduced by Mitsuru Matsui [17]. The main purpose of linear attack is to find the relationship between input and output. Although the probability is quite low, suppose the attacker is allowed to choose plaintext and ciphertext adaptively (known as adaptive chosen ciphertext attack), with sufficient pairs, he is able to recover some parts of a round key and perhaps the whole secret key by exhaustive search. Differential attacks to DES-like ciphers suggested by E.Biham & A. Shamir [11] extended from fault attacks [3] analyses the effect of differences in plaintext pairs on differences in ciphertext pairs. The probabilities discovered will help an attacker to extract the most probable key as long as there are sufficient chosen plaintext pairs. The standard way is to trace a path of high probable differences through the various stages of encryption.

To cut down computational cost, we attempt to abbreviate the number of rounds. However, this will also reduce the complexity to break the cipher. In this section, we will show that the coefficients of SubMix transformation we chose not only reduce number of tables required, but also have high *diffusion* property. Linear and differential attacks against MEAS are infeasible in WSN environment. By calculating their probabilities finds out the required plaintext and ciphertext pairs are far more than a sensor node can possibly send within rekeying period.

4.1 Linear Cryptanalysis

In cryptography, *confusion* refers to making the relationship between the key and ciphertext as complex as possible; *diffusion* requirement on a cipher is that each plaintext bit should influence every ciphertext bit and each key should influence every ciphertext bit as well. In a cipher with good diffusion like AES, each flipping input bit should change each output bit with a probability of one half.

The aim of linear cryptanalysis is to find the linear equations of the form

$$P[i_1, i_2, ..., i_a] \oplus C[i_1, i_2, ... i_b] = K[i_1, i_2, ..., i_c] \qquad (10)$$

A successful attacker can discover many of these linear equations and estimate the best probabilities of each round. Other than DES, AES has only one Sbox and already performs good diffusion property.
During MixColumns transformation, we find that multiplying {03} can be computed by bit-wised mapping in Figure 5. We can rewrite the equations and obtain a linear mapping from input a(x) to output a(x)'.

$$\begin{array}{lcl}
a_0' = a_0 \oplus a_7 & & a_0 \to a_0', a_1' \\
a_1' = a_0 \oplus a_1 \oplus a_7 & & a_1 \to a_1', a_2' \\
a_2' = a_1 \oplus a_2 & & a_2 \to a_2', a_3' \\
a_3' = a_2 \oplus a_3 \oplus a_7 & \to & a_3 \to a_3', a_4' \\
a_4' = a_3 \oplus a_4 \oplus a_7 & & a_4 \to a_4', a_5' \\
a_5' = a_4 \oplus a_5 & & a_5 \to a_5', a_6' \\
a_6' = a_5 \oplus a_6 & & a_6 \to a_6', a_7' \\
a_7' = a_6 \oplus a_7 & & a_7 \to a_0', a_1', a_3', a_4', a_7'
\end{array}$$

Figure 5 linear mapping of {03} from bit-wised mapping

The following equation estimates the probability of every linear equation introduced in [17].

$$N(\alpha, \beta) \equiv \# \{x \mid 0 \le x \le 256, x \wedge \alpha = F(x) \wedge \beta\} \qquad (11)$$

α denotes input mask, β denotes output mask and '∧'denotes bit-wised AND operation. If α={01} means observing the last bit a_0 only and β is for observing the output bits. According to Figure 5, if the attacker fixes bit a_0, he needs to observe the bits a_0 and a_1 at the output. Suppose he is fixing input bit a_7, there are five output bits influenced.
Compare to equation (3.3), we rewrite the linear mapping of {DE} in Figure 6. One can see {DE} provides a better way of mixing data and more complex linear mapping equations than {03} in AES.

$$\begin{array}{l}
a_0 \to a_1', a_2', a_3', a_4', a_6', a_7' \\
a_1 \to a_0', a_1', a_2', a_5', a_7' \\
a_2 \to a_0', a_4', a_6' \\
a_3 \to a_1', a_3', a_5', a_7' \\
a_4 \to a_0', a_1', a_2', a_3', a_6' \\
a_5 \to a_1', a_2', a_3', a_4', a_7' \\
a_6 \to a_0', a_1', a_2', a_5' \\
a_7 \to a_1', a_2', a_3', a_6'
\end{array}$$

Figure 6 linear mapping of {DE} from bit-wised mapping

When choosing the coefficients for SubMix, first we expand polynomials in GF(2^8) into bit-wised mapping equations and calculate their hamming weights. In fact every coefficient of p(x) has good diffusion property.

4.2 Differential Cryptanalysis

We now show a way to do differential cryptanalysis of last two rounds. A byte of fault is injected (like the method introduced in [20]) before SubMix transformation and denotes as ε. Since AddRoundKey and ShiftRows have no effects on the value of difference. The fault propagates from one byte to a column after MixColumns. After ShiftRows, the fault is shifted to different columns, and will be distributed to the whole state after next MixColumns.

$$\begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \varepsilon_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow$$

after ShiftRows after SubBytes

$$\begin{bmatrix} 17\cdot\varepsilon_0 & 0 & 0 & 0 \\ 17\cdot\varepsilon_0 & 0 & 0 & 0 \\ 5D\cdot\varepsilon_0 & 0 & 0 & 0 \\ 37\cdot\varepsilon_0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 17\cdot\varepsilon_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17\cdot\varepsilon_0 \\ 0 & 0 & 5D\cdot\varepsilon_0 & 0 \\ 0 & 37\cdot\varepsilon_0 & 0 & 0 \end{bmatrix}$$

after MixColumns after ShiftRows

$$\rightarrow \begin{bmatrix} \varepsilon_0' & 0 & 0 & 0 \\ 0 & 0 & 0 & \varepsilon_1' \\ 0 & 0 & \varepsilon_2' & 0 \\ 0 & \varepsilon_3' & 0 & 0 \end{bmatrix}$$

after SubBytes

To estimate our success rate, first we construct a differential distribution table of AES's Sbox like in [11] and find out some probable faults are likely to happen more than others. For Example, let ε = {1F}, then we have ε_0 = SubBytes(ε) = {A3}. After MixColumns transformation, difference propagation of the first column are {5D}, {A3}, {A3}, and {FE} with a probability less than 0.006169%. In the last round, we check the differential table again to obtain the last probability, which is about 0.003060%. Hence, number of pairs to mount a successful differential attack (to obtain the last round key) will be $5.29*10^8$ pairs.

As for MAES, let ε = {67}, then we have ε_0 = SubBytes(ε) = {E6}. After MixColumns transformation, difference propagation of the first column are {66}, {66}, {65}, and {B9} with a probability less than 0.006168% and 0.003064% in the last round. Our proposed SubMix transformation turns out to be as strong as standard AES.

5. Implementation

Due to the limited resources including power supply and area on wireless sensor network nodes, we will introduce some techniques we use in the design of MAES to achieve the goals of reduced power and low gate count in this section.

After the analysis of overall AES and modified AES, inclusive of encryption, decryption, and key expansion, some portions of these three algorithms should be focused to implement an efficient design with lower power consumption and lower gate count.

Firstly, as to the key expansion part, two methods including pre-computation method and on-the-fly key schedule method [6] are usually used to generate round keys. On-the-fly key schedule first generates the expanded keys from the Cipher key and then decides the round key of each round. Each round key is used when it is generated, so it is not necessary to use storage elements to store keys as what pre-computation method does. Under the concern of area and power, on-the-fly key schedule is adopted.

Secondly, the most critical part and power-hungry part of AES and MAES are (1) SubBytes in key expansion and encryption round functions, and (2) InvSubBytes in decryption round functions [26] [2]. We use Galois field arithmetic [16] to complete the operations under the consideration of timing and area.

5.1 Area Reduction Techniques

There are many methods that can be used to implement Sbox. The better way is using the Galois field arithmetic [16] [9] under the consideration of performance and area. One operation of SubBytes is calculating the multiplicative inverse over GF(2^8). However, it is quite complex to compute directly, therefore we first transform it from GF(2^8) to GF($(2^4)^2$) and then we can use inverter in GF($(2^4)^2$). Since the polynomial $x^8 + x^4 + x^3 + x + 1$ ({11B} in hexadecimal form) used in AES and MAES is not a primitive irreducible polynomial, we had better use isomorphism before and after the implementation of GF($(2^4)^2$) inverter. In GF($(2^4)^2$), the primitive irreducible polynomial $x^8 + x^4 + x^3 + x^2 + 1$ ({11D} in hexadecimal form) is a better basis. The isomorphism function from GF(2^8) to GF($(2^4)^2$) is B as shown in Equation 12 [16] [25] and the isomorphism function from GF($(2^4)^2$) to GF(2^8) is B^{-1} as shown in Equation 13 [16] [25].

$$\begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&1&0&0 \\ 1&1&0&1&0&0&1&0 \\ 0&1&1&1&0&0&0&0 \\ 0&0&0&1&0&1&0&0 \\ 1&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&1&0 \\ 0&1&1&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \quad (12)$$

$$\begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \quad (13)$$

Since both SubBytes and InvSubBytes have the operation of calculating multiplicative inverse, it is a good idea to share the GF inverters to reduce gate count [27] [1] , as illustrated in Figure 7, 8, and 9.

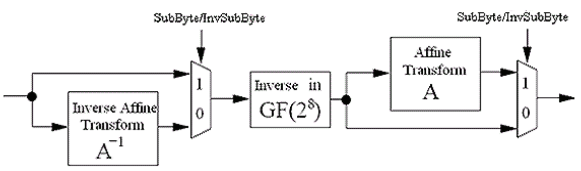


Figure 7 GF(2^8) inverter shared by SubBytes and InvSubBytes

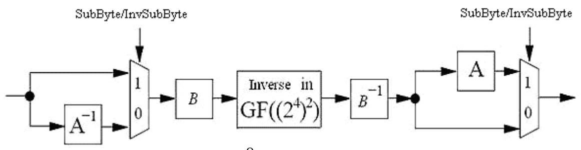


Figure 8 GF(2^8) inverter is transformed to GF(($2^4)^2$) inverter

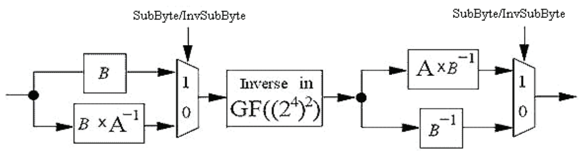


Figure 9 Isomorphism functions and affine transformations are further combined together

In Figure 7, when SubBytes is performed, SubBytes/InvSubBytes = 1, the input is directly going to the GF(2^8) inverter and then taking affine transformation to generate the output. When InvSubBytes is performed, SubByte/InvSubBytes = 0, the input is first taking inverse affine transformation and then going through the GF(2^8) inverter to produce the output. In Figure 8, GF(2^8) inverter is changed to GF(($2^4)^2$) inverter by placing isomorphism transformations before and after the inverter. In Figure 9, isomorphism functions *B* and B^{-1} are further merged with affine transformations A and A^{-1}, the path delay is therefore shortened. Besides, the area is also reduced.

The GF(($2^4)^2$) inverter is implemented based on Euclid's algorithm [10]. Input a is 8-bit long. After a series of operations, the 8-bit substituted result $\mathbf{a}^{-1}$ is obtained. Among those transformations, multiplication in GF(($2^4)^2$) is similar to byte-multiplication.

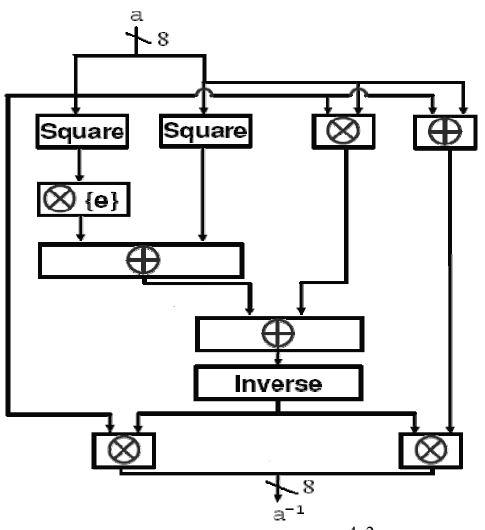


Figure 10 The structure of the GF(($2^4)^2$) inverter, the input is **a** and output is $\mathbf{a}^{-1}$

We find that SubBytes and the multiplications of MixColumns can be combined together with tables to reduce the critical path and gate count. In Figure 11, b0 = {MixColumns (SubBytes (a0), {17})}, which means that a0 is first substituted with Sbox and then multiplies with {17}over GF(2^8). For example, if input a0 = {00}, then output b0 = {58}.

As a result, the way to complete both the transformations of SubBytes and MixColumns in the encryption process is to XOR the results obtained from the tables described above.

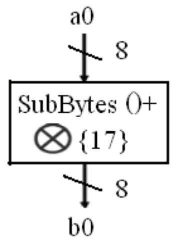
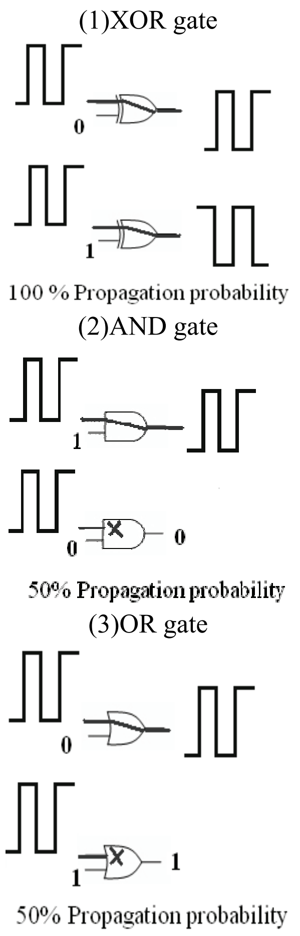


Figure 11 SubMix table of {17}

5.2 Power Reduction Techniques

In MAES, SubBytes and InvSubBytes are not only the most critical parts but also the most power-consuming parts. So, it is critical to reduce the overall power by reducing Sbox and Inverse-Sbox power [26].

Power consumption of Sbox and Inverse-Sbox is greatly influenced by the number of dynamic hazards, which is caused by differences of signal arrival times at each gates and the propagation probability of signal transitions. As illustrated in Figure 12, an XOR gate transfers signal transitions from input to output with probability 100%. For AND, OR gates, the probability is 50%. Therefore, it is a good way to place AND or OR gates before XOR gates to reduce the probabilities of signal transition, as illustrated in Figure 13.


(1)XOR gate
0
1
100 % Propagation probability
(2)AND gate
1
0
0
50% Propagation probability
(3)OR gate
0
1
1
50% Propagation probability
Figure 12 Propagation probabilities of signal transitions

As shown in Figure 14, composite field Sbox and Inverse Sbox are divided into three blocks to reduce the probabilities of signal transitions [2]. In addition, the two-level logic, i.e. AND-XOR arrays are used to reduce the number of dynamic hazards.

In conclusion, we use the following approaches to achieve power reduced Sbox and Inverse Sbox: (1) Use composite field Sbox to reduce gate count, (2) Divide combinational logic into three stages to reduce the probabilities of signal transitions, and (3) Adjust the signal timing by using two-level (AND-XOR) logic to reduce the number of dynamic hazards.

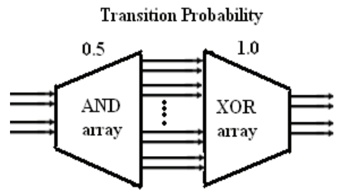


Figure 13.1 Two-level AND-XOR logic

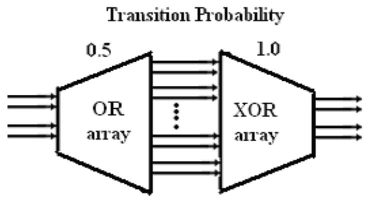


Figure 13.2 Two-level OR-XOR logic

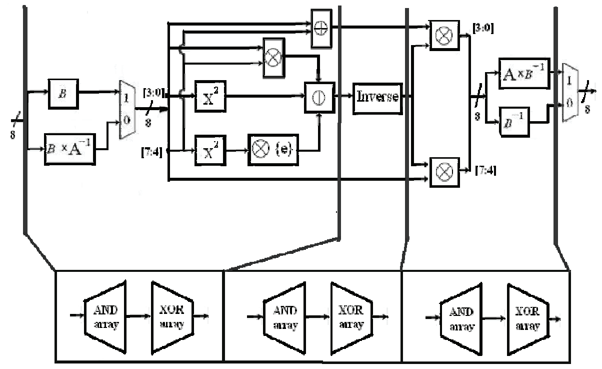


Figure 14 3-stage AND-XOR architecture used for composite field S-box

Figure 15 shows the key scheduler we use in our MAES design [26] [14]. When the encryption routine is processed, EN/ DE = 1 and Init_K0 through Init_K3 are the Cipher key. In the beginning, Init_K0 through Init_K3 are put into K0 through K3, respectively. After one clock cycle,

$$K0 = K0 \oplus \mathrm{Rcon}[0] \oplus [\mathrm{SubWord}(\mathrm{RotWord}(K3))]$$

K’1=K’0 ⊕ K1, K’2=K’1 ⊕ K2, K’3=K’2 ⊕ K3 and the generated K’0 through K’3 become the new K0 through K3. After another 6 clock cycles, all round keys are generated and the final round key are stored in registers. Similarly, when the decryption routine is processed, EN/DE = 0 and Init_K0 through Init_K3 are the final key stored in registers. At first, Init_K0 through Init_K3 are put into K0 through K3, respectively. After one clock cycle, K’3=K2 ⊕ K3, K’1=K0 ⊕ K1, K’2=K1 ⊕ K2,

$$K'0 = K0 \oplus \mathrm{Rcon}[0] \oplus [\mathrm{SubWord}(\mathrm{RotWord}(K'3))]$$

, and the generated K’0 through K’3 become the new K0 through K3. After another 6 clock cycles, all round keys are generated.

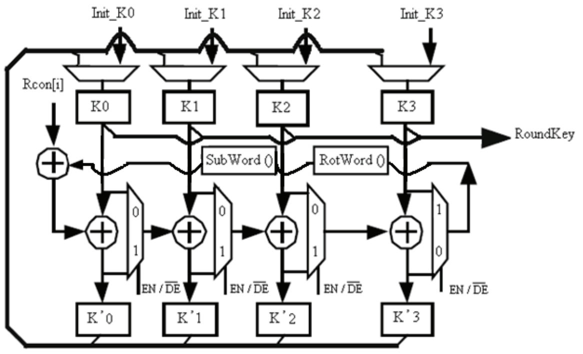


Figure 15 Circuit of key scheduler

5.3 Design Flow and Design Environment

We use cell-based design flow to implement our design, since our design belongs to digital logic. To perform simulation and synthesis, we utilize some EDA tools supplied by CIC (National Chip Implementation Center). In the beginning, we write Verilog-HDL (Hardware Description Language) codes according to the spec. Then we use ModelSim SE to simulate the design. The simulation is just to verify the correctness of the functions we want to design and the simulation is in the phase of RTL (Register Transfer Level) simulation. After the function is checked and is correct, we use another EDA tool, named Design Compiler, issued by Synopsis Corporation to transfer our design from RTL to a gate-level design. This process is called logic synthesis, which obtains a logic gate list according to some synthesizing rules and cell libraries. The cell library we choose for synthesis is UMC 0.18um process. Besides, it also inserts the concept of timing into the design. Afterward, we use Verilog-XL to simulate the gate-level design with timing annotated. After both timing and functions are checked to be correct, we use Xilinx FPGA board to verify again the functions of the design. The board we use is Xilinx Virtex-II Pro XC2VP30. The FPGA view of our circuit after place and route is illustrated in Figure 16.

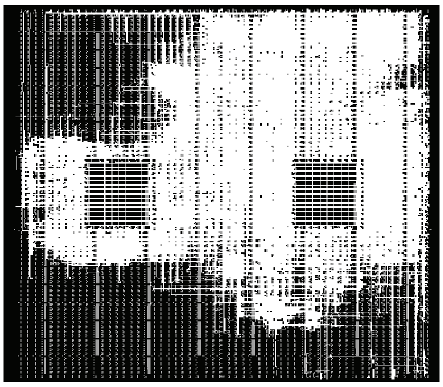


Figure 16 The FPGA view of our circuit after place and route

Table 2 shows a comparison of Horng's [6] implementation of AES and our implementation of modified AES. According to the table, we can find that the gate count of our circuit is about 69% of Horng's circuit. In addition, the power consumption of our circuit is about 79% of Horng's circuit. Throughputs are equal.

Table 2 Comparisons

	Horng's	Ours
Technology	UMC 0.18um	UMC 0.18um
Clock rate	125MHz	125MHz
Throughput	1.6Gbps	1.6Gbps
Gate count	67.9K	47.2K
Power consumption	56mW	44.21mW

6. Future work

Coefficients of MixColumns transformation may not be the optimal values when considering branch numbers. For example, $\{FE\}x^3 + \{6F\}x^2 + \{C4\}x + \{C4\}$ is also a good choice. On the other hand, we are looking forward to a reduction in the decryption process for those low-end devices like RFID. We can further combine InvSubBytes and InvMixColumns into InvSubMix and implement all the functions by bit-wised mapping instead of look-up tables.

7. Conclusions

In this paper, we proposed a modified AES algorithm. SubMix transformation is new stage combining SubBytes and MixColumns, implemented by look-up tables that perform faster encryption. Polynomial p(x) for MixColumns is chosen dedicatedly to achieve higher diffusion rate, and have a 512 bytes reduction of table spaces comparing to standard AES. We provided linear and differential cryptanalysis to show our result has certain security because the number of pairs required to mount a successful attack is far more than a sensor node can broadcast.

Overall our result can reach 1.6 Gps of throughput at 125MHz consuming 44.21mW and is suitable for encrypting data in wireless sensor networks.

References

[1] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “Unified hardware architecture for128-bit block ciphers AES and Camellia”, in Cryptographic Hardware and Embedded Systems (CHES) 2003. Aug. 2003, Springer-Verlag.

[2] A. Hodjat, I. Verbauwhede, “Minimum Area Cost for a 30 to 70 Gbits/s AES Processor”, IEEE computer Society Annual Symposium on VLSI,. pp. 83-88, Feb. 2004.

[3] Boneh, DeMillo, and Lipton, On the Importance of Checking Cryptographic Protocols for Faults, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT’97, pp. 37-51, 1997.

[4] B. Schneier, et. al. The Twofish Encryption Algorithm: A 128-Bit Block Cipher. John Wiley & Sons, April 1999.

[5] B. Schneier, "Blowfish" ,Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

[6] C.-L. Horng, “An AES Cipher Chip Design Using On-the-Fly Key Scheduler”, Master Thesis, Dept. Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan, June 2004.

[7] D. Wheeler and R. Needham. Tea, a tiny encryption algorithm. http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html , November 1994.

[8] D.W. Carman. Constraints and approaches for distributed sensor network security. Technical Report, #00-010, NAI Labs, 2000.

[9] E. D. Mastrovito, “VLSI Architecture for Computations in Galois Fields”, Ph.D. Thesis, Dept of EE, Linköping Univ., Lingköping, Sweden 1991.

[10] E. Trichina, “Combinational logic design for AES SubByte transformation on masked data”, IACR report, 2003. Available at http://eprint.iacr.org/2003/236.pdf.

[11] E. Biham, A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, Vol. 4 No. 1 1991.

[12] J. Daemen, L. Knudsen, and V. Rijmen. The Block Cipher Square. In E. Biham, editor, Fast Software Encryption ’97, volume 1267 of Lecture Notes in Computer Science, pages 149–165, Haifa, Israel, January 1997. Springer-Verlag.

[13] J. Daemen and V. Rijmen. *AES proposal: Rijndael.* http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf, 1999

[14] J. H. Shim, D. W. Kim, Y. K. Kang, T. W. Kwon, and J. R. Choi, “A rijndael cryptoprocessor using shared on-the-fly key scheduler”, in Proc. 3rd IEEE Asia-Pacific Conf. ASIC, Taipei, Aug. 2002, pp. 89–92.

[15] J. Nechvatal, et. al., Report on the Development of the Advanced Encryption Standard(AES), National Institute of Standards and Technology, October 2,2000.

[16] J. Wolkerstorfer, E. Oswald, and M. Lamberger, “An ASIC Implementation of the AES SBoxes “, CT-RSA 2002, LNCS 2271, pp. 67-78,2002.

[17] M. Matusi, "*Linear Cryptanalysis Method for DES Cipher*." In T. Helleseth, editor, Advances in Cryptology - EUROCRYPT'93, Volume765 of Lecture Notes in Computer Science, pp.386-397. Springer-Verlag, Berlin, Heidelberg, NewYork, 1994.

[18] National Institute of Standards and Technology (NIST), Advanced Encryption Standard (AES), National Technical Information Service, Springfield, VA 22161, Nov. 2001.

[19] National Bureau of Standards, NBS FIPS PUB 46, “*Data Encryption Standard*," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[20] P. Dusart, G. Letourneux, O. Vivolo, Differential Fault Analysis on AES, available at: http://www.unilim.fr/laco/rapports/2003/R2003_01.pdf, 2003

[21] P. Noo-intara, S. Chantarawong, and S. Choomchuay, "Architectures for MixColumn Transform for the AES," Proc. of Information and Computer Engineering Workshop 2004 (ICEP2004), Prince of Songkla University (Phuket Campus), January 2004, pp.152-156.

[22] RSA Laboratories. PKCS #1: *RSA encryption standard*, Version 1.5, November 1993.

[23] R. L. Rivest. *The RC5 Encryption Algortihm*, Proceedings of Fast Software Encryption Workshop 1994, pp. 86-96.

[24] R. L. Rivest, M. J. B. Robshaw, R. Sydney, and Y. L. Yin, "*The RC6 block cipher*," v1.1, Aug. 1998, available at http://www.rsasecurity.com/rsalabs/rc6.

[25] S. Chantarawong, P. Noo-intara, and S. Choomchuay, "An Architecture for Sbox Computation in the AES", Proc. of Information and Computer Engineering Workshop 2004 (ICEP2004), Prince of Songkla University (Phuket Campus), January 2004, pp.157-162.

[26] S. Chantarawong and S. Choomchuay, "An Architecture for a compact AES System", Proc. of Electrical Eng./Electronics, Communications, Computer and Information Technology Conference 2004 (ECTI-CON2004), ECTI Association, Thailand, May 2004, pp. 121-124.

[27] T.-Fu Lin, C.-Pin Su, C.-Tsun Huang, and C.-Wen Wu, "A High-Throughput Low-Cost AES Cipher Chip", in Proc. 3rd IEEE Asia-Pacific Conf. ASIC, Taipei, Aug. 2002, pp. 85-88.

Appendix A

Table 3 SubMix table of {17}

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

Table 4 SubMix table of {37}

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

Table 5 SubMix table of {5D}

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

Appendix B

```
Plaintext  bits 74 68 69 73 20 69 73 20 61 20 74 65 73 74 21 21
Key bits        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext bits 82 8f 0d 72 38 71 26 fc 0e 46 6a 19 9f 07 3e 4b
==========================================================================
Round Key  00 00 00 00
           00 00 00 00
           00 00 00 00
           00 00 00 00
Round 1>
After ShiftRows  74 20 61 73
                 69 20 74 68
                 74 21 69 73
                 21 73 20 65
After SubMix      2c 42 1d 71
                  f6 07 0e ea
                  ac ec ba 6a
                  c5 ad 02 8c
After AddRoundKey  4e 20 7f 13
                   95 64 6d 89
                   cf 8f d9 09
                   a6 ce 61 ef
Round 2>
After ShiftRows  4e 20 7f 13
                 64 6d 89 95
                 d9 09 cf 8f
                 ef a6 ce 61
After SubMix      89 e2 b9 f8
                  9d f9 24 8e
                  b1 1d f9 ea
                  53 62 07 0b
After AddRoundKey  12 1b 22 01
                   05 02 bc 75
                   29 e6 61 11
                   9a c8 ce a1
Round 3>
After ShiftRows  12 1b 22 01
                 02 bc 75 05
                 61 11 29 e6
                 a1 9a c8 ce
After SubMix      15 c5 68 71
                  67 87 2b b7
                  49 a7 a2 92
                  af a4 9a 7a
After AddRoundKey  85 ac 9a 7a
                   f0 eb df b8
                   7d 68 f5 3e
                   ff 5e a9 e3
Round 4>
After ShiftRows  85 ac 9a 7a
                 eb df b8 f0
                 f5 3e 7d 68
                 e3 ff 5e a9
After SubMix      fd 85 ff 96
                  83 f8 56 8f
                  d7 68 d5 fe
                  2f a8 12 b3
After AddRoundKey  13 02 8a e8
                   85 92 c8 1e
                   0d 7d 97 10
                   54 29 a0 98
Round 5>
After ShiftRows  13 02 8a e8
                 92 c8 1e 85
                 97 10 0d 7d
                 98 54 29 a0
After SubMix      e4 90 04 cc
                  91 ea a5 24
                  ce 92 34 03
                  34 e1 5f af
After AddRoundKey  9b 68 89 3f
                   bf ae 7f 6f
                   e5 ac 48 91
                   bc e8 e4 3f
Round 6>
After ShiftRows  9b 68 89 3f
                 ae 7f 6f bf
                 48 91 e5 ac
                 3f bc e8 e4
After SubMix      16 46 5f 99
                  49 5f 74 31
                  11 43 f6 9a
                  ed 34 bc 7a
After AddRoundKey  fa 52 c6 f3
                   28 7a 8b 85
                   5a 36 ff 01
                   68 b8 8b dd
Round 7>
After ShiftRows  fa 52 c6 f3
                 7a 8b 85 28
                 ff 01 5a 36
                 dd 68 b8 8b
After SubMix      a3 ba a1 b4
                  4d 21 89 e7
                  19 24 01 e9
                  18 0c 02 d0
After AddRoundKey  82 8f 0d 72
                   38 71 26 fc
                   0e 46 6a 19
                   9f 07 3e 4b
Ciphertext
```