Tạp chí Tin học và Điều khiển học, T. 17, S. 4 (2001), 45-56

THE 256/384/512-BIT VERSION OF THE RIJNDAEL BLOCK CIPHER

DUONG ANH DUC – TRAN MINH TRIET – LUONG HAN CO

Abstract. The Rijndael Block Cipher has been chosen to be Advanced Encryption Standard (AES) since October 2nd 2000. The Cipher processes blocks and keys having 128, 192, or 256 bits. The Extended Rijndael Block Cipher is proposed to process larger blocks and keys of the length 256, 384, or 512.

Tóm tắt. Phương pháp mã hóa Rijndael vừa được Viện Tiêu Chuẩn và Công Nghệ Hoa Kỳ (NIST) chính thức chọn làm chuẩn mã hóa AES (Advanced Encryption Standard) vào ngày 2 tháng 10 năm 2000. Trên thực tế, phương pháp mã hóa Rijndael xử lý các khối dữ liệu và mã khóa có độ dài 128, 192 hoặc 256 bit. Trong bài viết này, chúng tôi giới thiệu phiên bản mở rộng 256/384/512-bit của thuật toán này có khả năng xử lý các khối dữ liệu và mã khóa có độ dài 256, 384 hoặc 512 bit.

1. INTRODUCTION

In this document we describe the 256/384/512-bit extended Rijndael-like Block Cipher. This is the extended version of the Rijndael Block Cipher, proposed by Vincent Rijmen and Joan Daeman, which has been chosen to be the AES by the National Institute of Standards and Technology (NIST). The input, the output and the cipher key for the Extended-Rijndael are 256, 384 or 512 bits in length.

2. NOTATION AND CONVENTIONS

2.1. Extended-Rijndael Inputs and Outputs

The input, the output and the cipher key for Extended-Rijndael are each bit sequences containing 256, 384 or 512 bits with the constraint that the input and output sequences have the same length.

2.2. Bytes

The basic unit for processing in this algorithm is a **byte,** a sequence of eight bits treated as a single entity. Each bytes b is interpreted as a finite field element which can be represented in binary notation ($\{b_7b_6b_5b_4b_3b_2b_1b_0\}$) or hexadecimal notation ($\{h_1h_0\}$) or polynomial nocation $\sum_{i=0}^{7} b_i x^i$.

2.3. The State

All operations are performed on a two-dimensional array of bytes called the **State** consisting of ***eight*** rows of bytes, each containing Nb bytes, where Nb is the block length divided by 64. An individual byte of the State (denoted by the symbol s) is referred to as either $s_{r,c}$ or $s[r,c]$ where r is its row number in the range $0 \le r < 8$ and c is its column number in the range $0 \le c < Nb$.

At the beginning of the Cipher or Inverse Cipher, the input array, *in*, is copied to the State array according to the scheme: $s[r, c] = in[r + 8c]$ for $0 \le r < 8$ and $0 \le c < Nb$ and at the end of the Cipher and Inverse Cipher, the State is copied to the output array *out* as follows $out[r + 8c] = s[r, c]$ for $0 \le r < 8$ and $0 \le c < Nb$.

The eight bytes in each column of the state array can be considered either as an array of eight bytes indexed by the row number r or as a single 64-bit **word**. The state can hence be considered as a one-dimensional array of words for which the column number c provides the array index.

3. POLYNOMIALS WITH COEFFICIENTS IN GF(2^8)

All bytes in the Extended-Rijndael algorithm are interpreted as finite field elements which can be added and multiplied. For these operations please refer to [2, 10, 15].

Eight-term polynomials can be defined - with coefficients that are finite field elements - as $a(x) = \sum_{i=0}^{7} a_i x^i$ which will be denoted as a word in the form $[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$.

Let $b(x) = \sum_{i=0}^{7} b_i x^i$ define a second eight-term polynomial. Addition is performed by adding (XORing) the finite field coefficients of like powers of x:

$$a(x) + b(x) = \sum_{i=0}^{7} (a_i \oplus b_i) x^i). \quad (1)$$

Multiplication is achieved in two steps. In the first step, the polynomial product $c(x) = a(x) \bullet b(x)$ is algebraically expanded, and like powers are collected to give:

$$c(x) = \sum_{i=0}^{14} c_i x^i \text{, where } c_i = \bigoplus_{j=0}^{i} (a_j \bullet b_{i-j}). \quad (2)$$

The second step of the multiplication is to reduce $c(x)$ modulo a polynomial of degree 8; the result can be reduced to a polynomial of degree less than 8. **This is accomplished with the polynomial $x^8 + 1$**, so that:

$$x^i \bmod (x^8 + 1) = x^{i \bmod 8}. \quad (3)$$

The modular product of $a(x)$ and $b(x)$, denoted by $a(x) \otimes b(x)$, is given by the eight-term polynomial $d(x)$, defined as follows:

$$d(x) = \sum_{i=0}^{7} d_i x^i \text{, where } d_i = \left(\bigoplus_{j=0}^{i} (a_j \bullet b_{i-j}) \right) \oplus \left(\bigoplus_{j=0}^{8+i} (a_j \bullet b_{8+i-j}) \right). \quad (4)$$

Because x^8+1 is not an irreducible polynomial over GF(2^8), multiplication by a fixed eight-term polynomial is not necessarily invertible. However, the Extended-Rijndael algorithm specifies a fixed eight-term polynomial that *does* have an inverse:

$$a(x) = \{03\}x^7 + \{05\}x^6 + \{03\}x^5 + \{02\}x^4 + \{02\}x^3 + \{04\}x^2 + \{02\}x + \{02\}, \quad (5)$$

$$a^{-1}(x) = \{03\}x^7 + \{04\}x^6 + \{03\}x^5 + \{03\}x^4 + \{02\}x^3 + \{05\}x^2 + \{02\}x + \{03\}. \quad (6)$$

4. THE CIPHER

The length *(Nb)* of the cipher input, the cipher output, the cipher state the cipher key *(Nk)*, measured in multiples of 64 bits *(Nb)*, is 4, 6 or 8. The number of rounds *(Nr)* depends on the block length and the key length: $Nr = \max\{Nb, Nk\}+6$.

The cipher is described in the following pseudo code, for which the individual transformations and the key schedule are described in the following sections (the array *w* contains the key schedule, an array of round keys).

```
Cipher(byte in[8 * Nb], byte out[8 * Nb], word w[Nb * (Nr + 1)])
begin
   byte state[8,Nb]
   state = in
   AddRoundKey(state, w)
   for round = 1 step 1 to Nr - 1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w + round * Nb)
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w + Nr * Nb)
   out = state
end
```

4.1. The SubBytes Transformation

The **SubBytes** transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box) as in the original Rijndael algorithm ([2, 10, 15]).

This S-box, which is invertible, is constructed by composing two transformations:

1. Take the multiplicative inverse in the finite field GF(2^8); the element {00} is mapped to itself.
2. Apply an affine (over GF(2)) transformation defined by:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \,. \tag{7}$$

for $0 \le i < 8$, where b_i is the i^{th} bit of the byte, and $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$ = {63} = {01100011}. A prime on a variable (e.g., b') indicates that the variable is to be updated with the value on the right.

```
SubBytes(byte state[8,Nb])
begin
    for r = 0 step 1 to 8
        for c = 0 step 1 to Nb - 1
            state[r,c] = Sbox[state[r,c]]
        end for
    end for
end
```

4.2. The ShiftRows Transformation

The **ShiftRows** transformation operates individually on each row of the state by cyclically shifting the bytes in the row such that:

$$s_{r,c}' = s_{r,(c+shift(r,Nb)) \bmod Nb} \tag{8}$$

for $0 < r < 8$ and $0 \le c < Nb$, where the shift value $shift(r,Nb) = r \bmod Nb$

This has the effect of moving bytes to "lower" positions in the row (i.e., lower values of c in a given row), while the "lowest" bytes wrap around into the "top" of the row (i.e., higher values of c in a given row).

```
ShiftRows(byte state[8,Nb])
begin
    byte t[Nb]
    for r = 1 step 1 to 8
        for c = 0 step 1 to Nb - 1
            t[c] = state[r, (c + shift[r,Nb]) mod Nb]
        end for
        for c = 0 step 1 to Nb - 1
            state[r,c] = t[c]
        end for
    end for
end
```

4.3. The MixColumns Transformation

The **MixColumns** transformation operates on the State column-by-column, treating each column as a eight-term polynomial as described in Sec. 3. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^8 + 1$ with a fixed polynomial $a(x)$, given by:

$$a(x) = \{03\}x^7 + \{05\}x^6 + \{03\}x^5 + \{02\}x^4 + \{02\}x^3 + \{04\}x^2 + \{02\}x + \{02\} \tag{9}$$

The pseudo code for this transformation is as follows, where the function **FFmul(x, y)** returns the product of two finite field elements x and y.

```
MixColumns(byte state[8,Nb])
begin
    byte t[8]
    for c = 0 step 1 to Nb - 1
        for r = 0 step 1 to 8
            t[r] = state[r,c]
        end for
        for r = 0 step 1 to 8
            state[r,c] =
                FFMul(0x02, t[r])              xor FFmul(0x03, t[(r + 1) mod 8]) xor
                FFmul(0x05, t[(r + 2) mod 8]) xor FFMul(0x03, t[(r + 3) mod 8]) xor
```

```
                FFmul(0x02, t[(r + 4) mod 8]) xor FFmul(0x02, t[(r + 5) mod 8]) xor
                FFmul(0x04, t[(r + 6) mod 8]) xor FFmul(0x02, t[(r + 7) mod 8])
        end for
    end for
end
```

4.4. The AddRoundKey Transformation

In the **AddRoundKey** transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of *Nb* words from the key schedule (described in Sec. 4.5). Those *Nb* words are each added into the columns of the State, such that:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}, s'_{4,c}, s'_{5,c}, s'_{6,c}, s'_{7,c}] =$$
$$[s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}, s_{4,c}, s_{5,c}, s_{6,c}, s_{7,c}] \oplus [w_{round*Nb+c}] \quad (10)$$

where $[w_i]$ are the key schedule words described in Sec. 4.5, and *round* is a value in the range $0 \leq round \leq Nr$. In the Cipher, the initial Round Key addition occurs when $round = 0$, prior to the first application of the round function. The application of the **AddRoundKey** transformation to the *Nr* rounds of the Cipher occurs when $1 \leq round \leq Nr$. This transformation is its own inverse, since it only involves an application of the XOR operation.

4.5. Key Expansion

The round keys are derived from the cipher key by means of a key schedule with each round requiring *Nb* words of key data with an extra initial set making $Nb(Nr + 1)$ words in total. The key schedule consists of a linear array of 8-byte words denoted by either w_i or $w[i]$ with i in the range $0 \leq i < Nb(Nr + 1)$.

The expansion of the input key into the key schedule proceeds according to the following pseudo code where the function **SubWord(x)** gives an output word in which the S-box substitution has been individually applied to each of the eight bytes of its input x.

The function **RotWord(x)** takes a word $[b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7]$ as input and returns the word $[b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_0]$.

The word array **Rcon[i]** contains the values given by $[x^{i-1}, 0, 0, 0, 0, 0, 0, 0]$ with x^{i-1} being powers of x in the field GF(2^8) (note that i starts at 1, not 0).

```
KeyExpansion(byte key[8 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i = 0
    while (i < Nk)
        w[i] = word[key[8*i]   , key[8*i+1], key[8*i+2], key[8*i+3],
                    key[8*i+4], key[8*i+5], key[8*i+6], key[8*i+7]]
        i = i + 1
    end while
    i = Nk
    while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
        else
            if ((Nk = 8) and (i mod Nk = 4))
                temp = SubWord(temp)
            end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end
```

Note that this key schedule, which is illustrated in Figure 1 for $Nk = 4$ and $Nb = 6$, can be generated 'on-the fly' if necessary using a buffer of max(*Nb*, *Nk*) words.

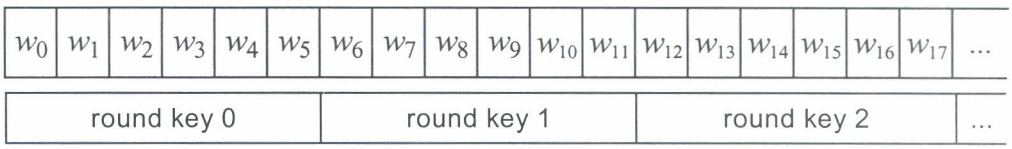

Figure 1. The key schedule and round key selection for $Nk = 4$ and $Nb = 6$

5. INVERSE CIPHER

The inversion of the cipher code presented in Sec. 4 is straightforward and provides the following pseudo code for the inverse cipher.

```
InvCipher(byte in[8 * Nb], byte out[8 * Nb], word w[Nb * (Nr + 1)])
begin
   byte state[8,Nb]
   state = in
   AddRoundKey(state, w + Nr * Nb)
   for round = Nr - 1 step -1 to 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w + round * Nb)
      InvMixColumns(state)
   end for
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w)
   out = state
end
```

5.1. The InvShiftRows Transformation

The **InvShiftRows** transformation operates individually on each row of the state cyclically shifting the bytes in the row such that:

$$s'_{r,(c+shift(r,Nb)) \bmod Nb} = s_{r,c} \tag{11}$$

for $0 < r < 8$ and $0 \le c < Nb$ where the cyclic shift values $shift(r, Nb)$ are mentioned in Sec. 4.2.

```
InvShiftRows(byte state[8,Nb])
begin
   byte t[Nb]
   for r = 1 step 1 to 8
      for c = 0 step 1 to Nb - 1
         t[(c + shift[r,Nb]) mod Nb] =
         state[r,c]
      end for
      for c = 0 step 1 to Nb - 1
         state[r,c] = t[c]
      end for
   end for
end
```

5.2. The InvSubBytes Transformation

InvSubBytes is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation (4.1) followed by taking the multiplicative inverse in GF(2^8).

The inverse of the affine tranformation (4.1) being:

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i \tag{12}$$

where byte $d = \{05\}$

```
InvSubBytes(byte state[8,Nb])
begin
    for r = 0 step 1 to 8
        for c = 0 step 1 to Nb - 1
            state[r,c] = InvSbox[state[r,c]]
        end for
    end for
end
```

5.3. The InvMixColumns Transformation

The **InvMixColumns** transformation acts independently on every column of the state and treats each column as a eight-term polynomial as described in Sec. 3. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^8 + 1$ with a fixed polynomial $a^{-1}(x)$, given by:

$$a^{-1}(x) = \{03\}x^7 + \{04\}x^6 + \{03\}x^5 + \{03\}x^4 + \{02\}x^3 + \{05\}x^2 + \{02\}x + \{03\}. \quad (13)$$

The pseudo code for this transformation is as follows, where the function **FFmul(x, y)** returns the product of two finite field elements x and y.

```
InvMixColumns(byte block[8,Nb])
begin
    byte t[8]
    for c = 0 step 1 to Nb - 1
        for r = 0 step 1 to 7
            t[r] = block[r,c]
        end for
        for r = 0 step 1 to 7
            block[r,c] =
            FFmul(0x03, t[r])              xor FFmul(0x03, t[(r + 1) mod 8]) xor
            FFmul(0x04, t[(r + 2) mod 8]) xor FFmul(0x03, t[(r + 3) mod 8]) xor
            FFmul(0x03, t[(r + 4) mod 8]) xor FFmul(0x02, t[(r + 5) mod 8]) xor
            FFmul(0x05, t[(r + 6) mod 8]) xor FFmul(0x02, t[(r + 7) mod 8])
        end for
    end for
end
```

5.4. Equivalent Inverse Cipher

In the straightforward Inverse Cipher presented above, the sequence of the transformations differs from that of the Cipher, while the form of the key schedules for encryption and decryption remains the same. However the order of **InvSubBytes** and **InvShiftRows** can be reversed. The order of **AddRoundKey** and **InvMixColumns** can also be reversed, provided that the columns (words) of the decryption key schedule are transformed using **InvMixColumns**. This latter operation shall *not* be performed on the first or the last ***Nb*** words in the key schedule, since those do not operate with **InvMixColumns**. Given these changes, the resulting Equivalent Inverse Cipher offers a more efficient structure than the straightforward Inverse Cipher described above.

In the pseudo code for the Equivalent Inverse Cipher, the word array **dw[]** contains the modified decryption key schedule.

```
EqInvCipher(byte in[8 * Nb], byte out[8 * Nb], word dw[Nb * (Nr + 1)])
begin
    byte   state[8,Nb]
    state = in
    AddRoundKey(state, dw + Nr * Nb)
    for round = Nr - 1 step -1 to 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state, dw + round * Nb)
    end for
```

```
        InvSubBytes(state)
        InvShiftRows(state)
        AddRoundKey(state, dw)
        out = state
    end
```

For the Equivalent Inverse Cipher, the following pseudo code is added at the end of the Key Expansion routine:

```
for i = 0 step 1 to (Nr + 1) * Nb – 1
    dw[i] = w[i]
end for
for rnd = 1 step 1 to Nr - 1
    InvMixColumns(dw + rnd * Nb)
end for
```

6. MOTIVATION FOR DESIGN CHOICES

The motivations for designing the S-box, the ShiftRows offsets, the Key Schedule, and the number of rounds of the Extended Rijndael are based on the original Rijndael Cipher ([1, 2, 9]). The most important changes have been made in designing the MixColumns transformation.

The MixColumns transformation:

`MixColumns` has been chosen from the space of 8-byte to 8-byte linear transformations using the following criteria:

1. Invertibility;
2. Linearity in GF(2);
3. Relevant diffusion power;
4. Speed on 8-bit processors;
5. Symmetry;
6. Simplicity of description.
7. Speed on 8-bit processors for the `InvMixColumns`

The criteria from 1 to 6 have been used to choose `MixColumns` in the standard Rijndael algorithm. The criterion 7 imposes that the coefficients of the `InvMixColumns` have small values, in order of preference {00}, {01}, {02}, {03}... For the original Rijndael algorithm, no solution can be found to satisfy all these seven criteria. The `MixColumns` used in the Rijndael does not satisfy the criterion 7; so the inverse cipher takes significantly more time than the cipher itself on 8-bit processors [1]

Branch number:

Let F be a linear transformation acting on byte vectors and let the byte weight of a vector be the number of nonzero bytes. The byte weight of a vector is denoted by $W(a)$. The *Branch Number* of a linear transformation is a measure of its diffusion power

Definition. The branch number of a linear transformation F is:

$$\min_{a \neq 0} \left(W(a) + W(F(a)) \right) \tag{14}$$

A non-zero byte is called an active byte. The coefficients of the MixColumns have been chosen in such a way that its branch number, denoted by B, is 8. A linear relation between input and output bits involves bits from at least 8 different bytes from input and output.

7. STRENGTH AGAINST KNOWN ATTACKS

7.1. Symmetry Properties and Weak Keys of the DES Type

Symmetry in the behavior of the cipher has been eliminated by using the round constants that are different for each round. The fact that the cipher and its inverse use different components practically eliminates the possibility for weak and semi-weak keys, as existing for DES. The non-linearity of the key expansion practically eliminates the possibility of equivalent keys.

7.2. Differential and Linear Cryptanalysis

7.2.1. Differential cryptanalysis (DC)

DC attacks [8] are possible if there are predictable difference propagations over all but a few (typically 2

or 3) rounds that have a prop ratio (the relative amount of all input pairs that for the given input difference give rise to the output difference) significantly larger than 2^{1-n} if n is the block length.

For this 256/384/512-bit extended Rijndael-like Block Cipher, we prove that there are no 4-round differential trails with a predicted prop ratio above $2^{-48(Nb+1)}$ (and no 8-round trails with a predicted prop ratio above $2^{-96(Nb+1)}$). For all block lengths of this extended version, this is sufficient. The proof is given in Sec. 7.2.3.

7.2.2. Linear cryptanalysis (LC)

LC attacks [13] are possible if there are predictable input-output correlations over all but a few (typically 2 or 3) rounds significantly larger than $2^{-n/2}$. For this extended version, we prove that there are no 4-round linear trails with a correlation above $2^{-24(Nb+1)}$ (and no 8-round trails with a correlation above $2^{-48(Nb+1)}$). For all block lengths of Extended-Rijndael, this is sufficient. The proof is given in Sec. 7.2.4.

7.2.3. Weight of differential and linear trails.

In [9], it is shown that:

- The prop ratio of a differential trail can be approximated by the product of the prop ratios of its active S-boxes.
- The correlation of a linear trail can be approximated by the product of input-output correlations of its active S-boxes.

The wide trail strategy can be summarised as follows:

- Choose an S-box where the maximum prop ratio and the maximum input-output correlation are as small as possible. For the 256/384/512-bit extended Rijndael-like Block Cipher, this is respectively 2^{-6} and 2^{-3}.
- Construct the diffusion layer in such a way that there are no multiple-round trails with few active S-boxes.

We prove that the minimum number of active S-boxes in any 4-round differential or linear trail is $8(Nb+1)$. This gives a maximum prop ratio of $2^{-48(Nb+1)}$ for any 4-round differential trail and a maximum of $2^{-24(Nb+1)}$ for the correlation for any 4-round linear trail. This is independent of the value of the Round Keys.

7.2.4. Propagation of patterns

For DC, the active S-boxes in a round are determined by the nonzero bytes in the difference of the states at the input of a round. Let the pattern that specifies the positions of the active S-boxes be denoted by the term (*difference*) activity pattern and let the (*difference*) *byte weight* be the number of active bytes in a pattern.

For LC, the active S-boxes in a round are determined by the nonzero bytes in the *selection vectors [9]* at the input of a round. Let the pattern that specifies the positions of the active S-boxes be denoted by the term (*correlation*) *activity pattern* and let the (*correlation*) *byte weight* $W(a)$ be the number of active bytes in a pattern a.

Moreover, let a column of an activity pattern with at least one active byte be denoted by *active column*. Let the *column weight*, denoted by $W_c(a)$, be the number of active columns in a pattern. The byte weight of a column j of a, denoted by $W(a)|_j$, is the number of active bytes in it.

The weight of a trail is the sum of the weights of its activity patterns at the input of each round.

Difference and correlation activity patterns can be seen as propagating through the transformations of the different rounds of the block cipher to form linear and differential trails.

This is illustrated with an example in Figure 2.

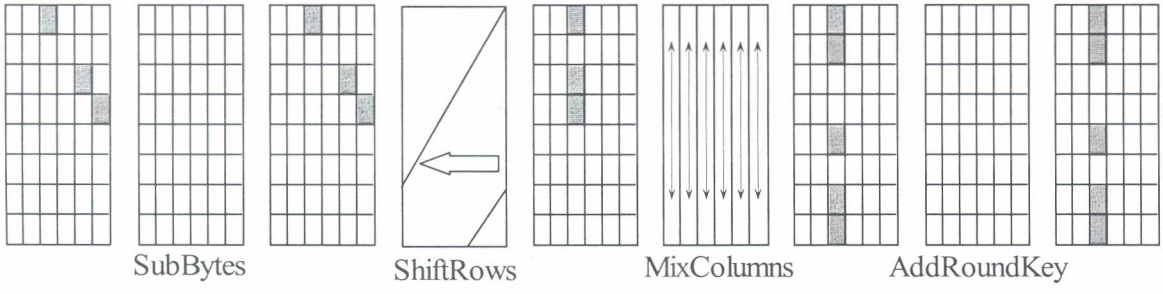

Figure 2. Propagation of activity pattern (in gray) through a single round

In our description, the activity pattern at the input of a round i is denoted by a_{i-1} and the activity pattern after applying `ShiftRows` of round i is denoted by b_{i-1}. The initial round is numbered 1 and the initial difference

pattern is denoted by a_0. Clearly, a_i and b_i are separated by `ShiftRows` and have the same byte weight, b_{j-1} and a_j are separated by `MixColumns` and have the same column weight. The weight of an m-round trail is given by the sum of the weights of a_0 to a_{m-1} . In the following figures, active bytes are indicated in dark grey, active columns in light grey.

Theorem 1. *The weight of a two-round trail with Q active columns at the input of the second round is lower bounded by 8Q.*

Proof. The fact that `MixColumns` has a Branch Number equal to 8 implies that sum of the byte weights of each column in b_0 and a_1 is lower bounded by 8. If the column weight of a_1 is Q, this gives a lower bounded of $8Q$ for the sum of the byte weights of b_0 and a_1 . As a_0 and b_0 have the same byte weight, the lower bounded is also valid for the sum of the weights a_0 and a_1, proving the theorem. **QED**

Theorem 1 is illustrated in Figure 3.

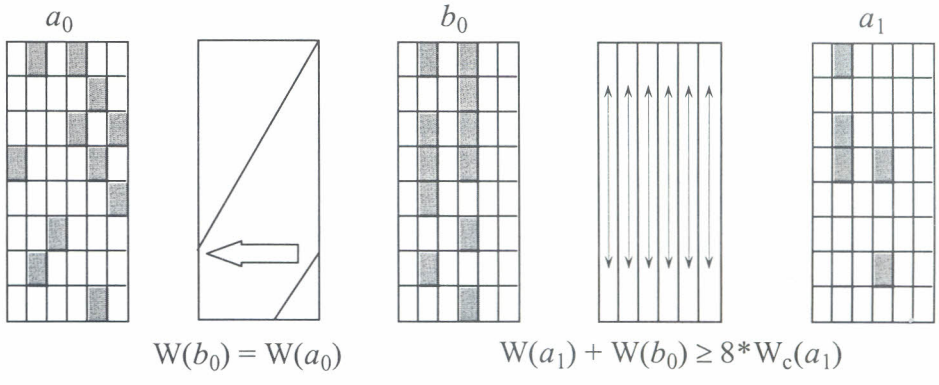

Figure 3. Illustration of Theorem 1 with Q = 2

From this it follows that any two-round trail has at least 8 active S-boxes.

Lemma 1. *In a two-round trail, the sum of the number of active columns at its input and the number of active columns at its output is at least Nb+1. In other words, the sum of the columns weights of a_0 and a_2 is at least Nb+1.*

Proof. `ShiftRows` moves all bytes in a column of a_i to different columns in b_i and vice versa. It follows that the column weight of a_i is lower bounded the byte weights of the individual columns of b_i and the number of columns in a state (Nb). Likewise the column weight of b_i is lower bounded by the byte weights of the individual columns of a_i and the number of columns in a state (Nb).
In a trail, at least one column of a_1 (or equivalently b_0) is active. Let this column be denoted by "column g". Because `MixColumns` has a branch number of 8, the sum of the byte weights of column g in b_0 and column g in a_1 is lower bounded by 8. The column weight of a_0 is lower bounded by the byte weight of column g of b_0 and the number of columns Nb. The column weight of b_1 is lower bounded by the byte weight of column g of a_1 and the number of columns Nb. It can be infered that the sum of the column weights of a_0 and b_1 is lower bounded by Nb+1. As the column weight of a_2 is equal to that of b_1, the lemma is proven. **QED**

Lemma 1 is illustrated in Figure 4.

Theorem 2. *Any trail over four rounds has at least 8(Nb+1) active bytes.*

Proof. By applying Theorem 1 on the first two rounds (1 and 2) and on the last two rounds (3 and 4), it follows that the byte weight of the trail is lower bounded by the sum of the column weight of a_1 and a_3 multiplied by 8. By applying Lemma 1, the sum of the column weight of a_1 and a_3 is lower bounded by $Nb+1$. From this it follows that the byte weight of the four-round trail is lower bounded by 8(Nb+1). **QED**

Theorem 2 is illustrated in Figure 5.

7.3. Truncated Differentials

The concept of truncated differentials was first published by Lars Knudsen [12]. The corresponding class of attacks exploit the fact that in some ciphers differential trails tend to cluster [9]. Clustering takes place if for certain sets of input difference patterns and output difference patterns, the number of differential trails is exceedingly large. The expected probability within that a differential trail stays the boundaries of the cluster

can be computed independently of the prop ratios of the individual differential trails. Ciphers in which all transformation operate on the state in well aligned blocks are prone to be susceptible to this type of attack. Since this is the case for 256/384/512-bit extended Rijndael-like Block Cipher, all transformations operating on bytes rather than individual bits, we investigated its resistance against "truncated differentials". For 6 rounds or more, no attacks faster than exhaustive key search have been found.

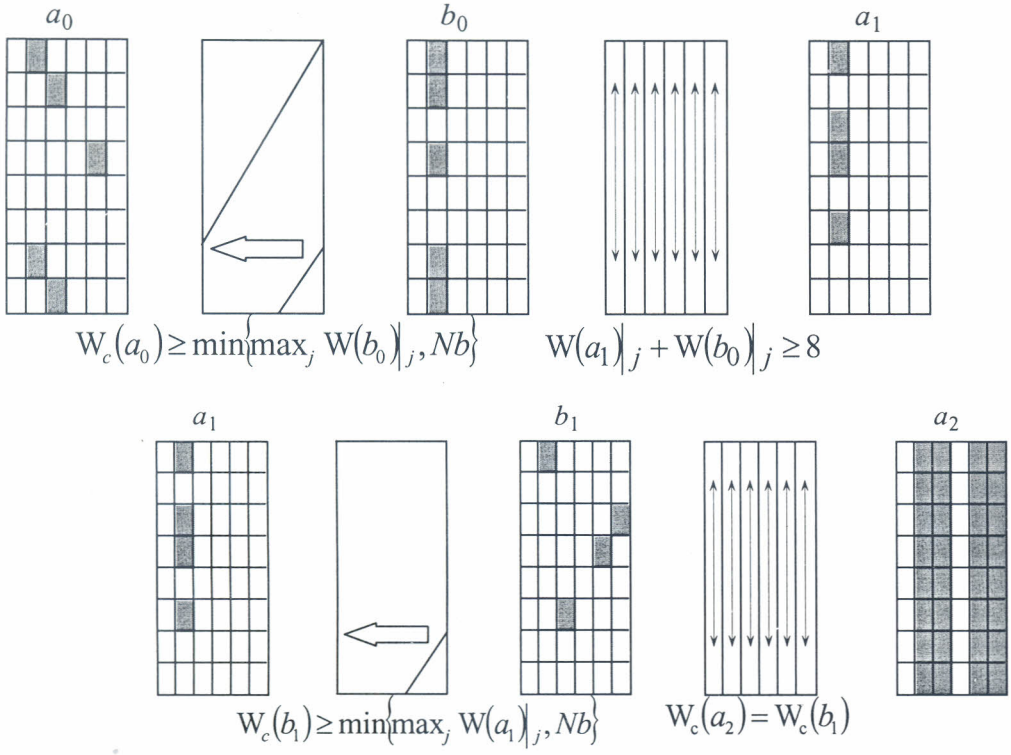

Figure 4. Illustration of Lemma 1 with one active column in a_1

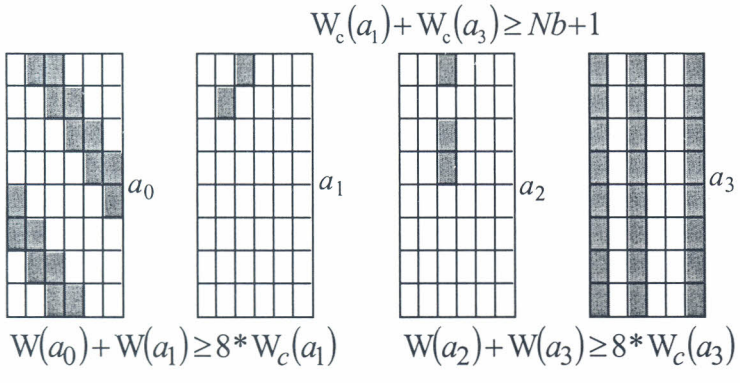

Figure 5. Illustration of Theorem 2

7.4. Interpolation Attacks

In this attack [14], the attacker constructs polynomials using cipher input/output pairs. This attack is feasible if the components in the cipher have a compact algebraic expression and can be combined to give expressions with manageable complexity. The basis of the attack is that if the constructed polynomials (or rational expressions) have a small degree, only few cipher input/output pairs are necessary to solve for the (key-dependent) coefficients of the polynomial. The complicated expression of the S-box in $GF(2^8)$, in combination with the effect of the diffusion layer prohibits these types of attack for more than a few rounds. The expression for the S-box is given by:

$$S(x)=\{63\}+\{8f\}x^{127}+\{b5\}x^{191}+\{01\}x^{223}+\{f4\}x^{239}+\{25\}x^{247}+\{f9\}x^{251}+\{09\}x^{253}+\{05\}x^{254}$$

7.5. Weak keys as in IDEA

The weak keys discussed in this subsection are keys that result in a block cipher mapping with detectable weaknesses. The best known case of weak keys are those of IDEA [9]. Typically, this weakness occurs for ciphers in which the non-linear operations depends on the actual key value. This is not the case for 256/384/512-bit extended Rijndael-like Block Cipher, where keys are applied using the XOR and all non-linearity is in the fixed S-box. In 256/384/512-bit extended Rijndael-like Block Cipher, there is no restriction on key selection.

7.6. Related-Key Attacks

In related-key attacks [11], the cryptanalyst can do cipher operations using different (unknown or partly unknown) keys with a chosen relation. The key schedule of 256/384/512-bit extended Rijndael-like Block Cipher, with its high diffusion and non-linearity, makes it very improbable that this type of attack can be successful for 256/384/512-bit extended Rijndael-like Block Cipher.

8. EXPERIMENTS AND PERFORMANCE

The cipher and its inverse take the same time Table 1 and Table 2 give the performance figures for the raw encryption implemented in C of the original Rijndael Block Cipher and the 256/384/512-bit extended Rijndael-like Block Cipher using the Microsoft Visual C++ 6.0 compiler. The experiments were performed on different platforms such as 200 MHz Pentium (running Microsoft Windows 98SE), 400 MHz Pentium II, and 733 MHz Pentium III running Microsoft Windows 2000 Professional.

		Speed (Mbits/sec)		
Key (bits)	Block (bits)	200 MHz Pentium	400 MHz Pentium II	733 MHz Pentium III
128	128	70.5	141.5	259.2
128	192	59.8	119.7	219.3
128	256	51.3	101.5	186.1

Table 1. Performance figures for the cipher

		Speed (Mbits/sec)		
Key (bits)	Block (bits)	200 MHz Pentium	400 MHz Pentium II	733 MHz Pentium III
256	256	27.4	56.4	103.4
256	384	23.3	47.5	87.1
256	512	20.2	42.0	76.9

Table 2. Performance figures for the cipher

From these tables, it can be seen that it is approximately four times slower to process each block when the length of the block and the cipher key are doubled. Although nearly half of the total speed of the cipher will be reduced, the space for exhaustive key search will be significantly enlarged.

9. CONCLUSION

256/384/512-bit extended Rijndael-like Block Cipher is expected, for all key and block lengths defined, to behave as good as can be expected from a block cipher with the given block and key lengths. This implies among other things, the following. The most efficient key-recovery attack for 256/384/512-bit extended Rijndael-like Block Cipher is exhaustive key search. Obtaining information from given plaintext-ciphertext pairs about other plaintext-ciphertext pairs cannot be done more efficiently than by determining the key by exhaustive key search. The expected effort of exhaustive key search depends on the length of the Cipher Key and is:

- for a 32-byte key, 2^{255} applications;
- for a 48-byte key, 2^{383} applications;
- for a 64-byte key, 2^{511} applications.

The rationale for this is that a considerable safety margin is taken with respect to all known attacks.

Basing on the same mathematical and cryptographic bases of the Rijndael Block Cipher, we also devise the 512/768/1024-bit extended Rijndael-like Block Cipher [1, 5]) and the generalized template of the Rijndael-like Block Cipher ([3, 6]) that can be used to generate ciphers with larger blocks and larger keys.

We are currently testing the optimized implementations of these extended versions of the Rijndael Block Cipher. Together with the original algorithm, these versions are being implemented in our applications including the plug-in for the Microsoft Outlook Express [15], the plug-in for the Eudora, the encoding/decoding modules for the digital maps and some other DSP equipments, the secured electronic mail system ([7, 15]), the security process in examinations [4]...

REFERENCES

[1] Duong Anh Duc, "*Some Optimizing Problems In Picturial Information Systems*", PhD Thesis (Draft), Department of Software Engineering, Faculty of Information Technology, University of Natural Sciences, VNUHCM, 2001.

[2] Duong Anh Duc, Tran Minh Triet, Luong Han Co, *The Advanced Encryption Standard*, 4th National Conference "Selected Topics in Information Technology", Hai Phong, Vietnam, 2001.

[3] Duong Anh Duc, Tran Minh Triet, Luong Han Co, The extended Rijndael-like Block Ciphers, *International Conference on Information Technology: Coding and Computing* - 2002, The Orleans, Las Vegas, Nevada, 2002 (accepted).

[4] Duong Anh Duc, Tran Minh Triet, Luong Han Co, The Advanced Encryption Standard And Its Application in the examination security in Vietnam, *International Conference on Information Technology: Coding and Computing* - 2002, The Orleans, Las Vegas, Nevada, 2002 (accepted).

[5] Duong Anh Duc, Tran Minh Triet, Luong Han Co, The extended version of the Rijndael Block Cipher, *Journal of Institute of Mathematics and Computer Sciences,* India, Dec. 2001.

[6] Duong Anh Duc, Tran Minh Triet, Luong Han Co, The extended versions of the Advanced Encryption Standard, Workshop on Applied Cryptology, Coding Theory and Data Iitegrrity, Singapore, 2001.

[7] Duong Anh Duc, Tran Minh Triet, Luong Han Co, Applying the Advanced Encryption Standard and its variants in Secured Electronic-Mail System in Vietnam, Workshop on Applied Cryptology, Coding Theory and Data Iitegrrity , Singapore, 2001.

[8] E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, *Journal of Cryptology* **4** (1) (1991) 3-72.

[9] J. Daemen, *Cipher and hash function design strategies based on linear and ifferential cryptanalysis*, Doctoral Dissertation, March 1995, K.U.Leuven.

[10] J. Daemen and V. Rijmen, *AES Proposal: Rijndael, AES Algorithm Submission*, September 3, 1999.

[11] J. Kelsey, B. Schneier and D. Wagner, *Key-schedule cryptanalysis of IDEA, GDES, GOST, SAFER, and Triple-DES, Advances in Cryptology, Proceedings Crypto '96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 237-252.

[12] L.R. Knudsen, *Truncated and higher order differentials*, Fast Software Encryption, LNCS 1008, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196-211.

[13] M. Matsui, Linear cryptanalysis method for DES cipher, *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386-397.

[14] T. Jakobsen and L.R. Knudsen, *The interpolation attack on block ciphers*, Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 28-40.

[15] Tran Minh Triet, Luong Han Co, "*Cryptography and Applications*", BSc Thesis, Department of Software Engineering, Faculty of Information Technology, University of Natural Sciences, VNUHCM, July 2001.

Received June 12, 2001

Faculty of Information Technology, University of Natural Sciences,
VNU, Ho Chi Minh City.