The Rijndael Block Cipher

Vincent Leith
MATH 2700.002

May 3, 2010

A brief look at the mathematics behind the Rijndael Block Cipher.

1 Introduction

The Rijndael Block Cipher was brought about by Joan Daemen and Vincent Rijmen when they attempted to compete in the competition set up by the American National Institute of Standards of Technology back in 1997. The competition was set up to hopefully find a new Advanced Encryption Standard to be used for the protection of sensitive data. Fifteen entered the competition and the group of Rijndael, Joan and Vincent, were chosen as the best choice for the new AES standard with their encryption algorithm.

2 Preliminary Mathematics

The Rijndael Block Cipher uses finite field arithmetic in the field of GF( 2^8 ) or the Galois Field. For Rijndael we show this field as a polynomial where each b is a bit in a byte that can contain the binary value of either 1 or 0.

\[ b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \]

The values used in Rijndael are displayed in hexadecimal value form and each hexadecimal value correspond to a polynomial representation.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

**Example:** Here is a byte containing the hexadecimal value of 57 and it corresponding binary and polynomial representations

- Binary = 01010111
- Polynomial = \( x^6 + x^4 + x^2 + x + 1 \)

- **Addition** When using the polynomial representation for addition the sum of two elements in the polynomial is the coefficients of the two terms being given by the sum modulo 2. This is done by using simple bitwise EXOR at a byte level ( shown by \( \oplus \)).
Example: $57 + 83 = D4$, shown below in both polynomial notation along with the binary notation:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

$$01010111 \oplus 10000011 = 11010100$$

This method also works the same for subtraction.

- **Multiplication** In polynomial representation for multiplication in the Galois field the polynomials need to be multiplied by an irreducible binary polynomial modulo of degree 8. The polynomial used in Rijndael is called $m(x)$ and shown by

$$m(x) = (x^8 + x^4 + x^3 + x + 1)$$

Example: $53 \cdot CA = 01$,

$$(x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x) = x^{13} + x^{12} + x^9 + x^7 + x^{11} + x^{10} + x^7 + x^5 + x^8 + x^7 + x^4 + x^2 + x^7 + x^6 + x^3 + x$$

Next step is to multiply by the modulo $m(x)$,

$$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x \mod (x^8 + x^4 + x^3 + x + 1) = 1$$

To work the modulo we are going to go through the binary steps,

$$11111101111110 \ mod 100011011 = 1$$

$$
\begin{array}{c}
1111101111110 \\
\oplus 100011011 \\
1100000111110 \\
\oplus 100011011 \\
10110101110 \\
\oplus 100011011 \\
0101110110 \\
\oplus 100011011 \\
00011010 \\
\oplus 100011011 \\
00000001
\end{array}$$

2
Using EXOR arithmetic we can use the modulo to reduce the binary value down to a 8 byte hexadecimal value once again thus making the answer 01.

If we multiply b(x) by the polynomial x, we have:

\[ b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \]

The reduced result is achieved by using modulo m(x). By multiplying by x at the byte level we can shift left and then use a conditional bitwise EXOR. This is done by \( b = xtime(a) \) to complete in xtime.

**Example:** 57•13 = FE.

Multiplication by x (using the hexadecimal notation),

\[
\begin{align*}
57 \cdot 02 &= xtime(57) = \text{AE} \\
57 \cdot 04 &= xtime(\text{AE}) = 47 \\
57 \cdot 08 &= xtime(47) = 8E \\
57 \cdot 10 &= xtime(8E) = 07
\end{align*}
\]

The following multiplication explained through binary arithmetic,

\[
\begin{array}{cccc}
57\cdot 02 & 57\cdot 04 & 57\cdot 08 & 57\cdot 10 \\
10101110 & 101011100 & 1010111000 & 10101110000 \\
⊕ 100011011 & ⊕ 100011011 & ⊕ 100011011 & \\
001000111 & 0010001110 & 00100011100 & ⊕ 1000110111 \\
& & & 000000111
\end{array}
\]

Finally using EXOR with the hexadecimal xtime value equivalent to the hexadecimal value 13,

\[ 57 \cdot 13 = 57 \cdot (01 ⊕ 02 ⊕ 10) = 57 ⊕ \text{AE} ⊕ 07 = \text{FE} \]

### 3 Polynomials with coefficients in GF(2^8)

Polynomials can be defined with coefficients in the Galois field and in this we can display a 4 byte vector using a polynomial degree 4 or below.

- **Addition** As before we use EXOR arithmetic for adding the polynomials by adding two of the vectors with a bitwise EXOR.

- **Multiplication** More complicated than addition due to it easily exceeding a 4 byte vector space. By using the modulo M(x) the result can be reduced to a polynomial with degree below 4.
  
  The modulo is as follows, \( M(x) = x^4 + 1 \)
Now assume that there are two polynomials in the Galois field,

\[ a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad \text{and} \quad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \]

The product of \( c(x) = a(x)b(x) \) is as follows,

\[ c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \]

\[
\begin{align*}
    c_0 &= a_0 \cdot b_0 \\
    c_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\
    c_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\
    c_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 \\
    c_4 &= a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
    c_5 &= a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
    c_6 &= a_3 \cdot b_3 
\end{align*}
\]

But \( c(x) \) can no longer be represented by a 4 byte vector so now we will implement the modulo \( M(x) \) of \( a(x) \) and \( b(x) \),

\[ d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \]

where \( d(x) = a(x) \otimes b(x) \)

\[
\begin{align*}
    d_0 &= a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
    d_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
    d_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3 \\
    d_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 
\end{align*}
\]

To simplify this operation we can write \( a(x) \) as a matrix due to it being a fixed polynomial. The matrix would simply be multiplied by the \( b(x) \) matrix,

\[
\begin{bmatrix}
    d_0 \\
    d_1 \\
    d_2 \\
    d_3 
\end{bmatrix} =
\begin{bmatrix}
    a_0 & a_3 & a_2 & a_1 \\
    a_1 & a_0 & a_3 & a_2 \\
    a_2 & a_1 & a_0 & a_3 \\
    a_3 & a_2 & a_1 & a_0 
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    b_3 
\end{bmatrix}
\] (1)

4 Rijndael’s S-box

The s-box is used in the ByteSub step of the encryption and is generated by determining the multiplicative inverse for a given number in the Galois field, zero would be set to zero. The multiplicative inverse is transformed using the following affine transformation:

\[
\begin{bmatrix}
    1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
    1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
    1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
    1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
    0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
    0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
    0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 
\end{bmatrix}
\begin{bmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
    x_5 \\
    x_6 \\
    x_7 
\end{bmatrix}
= \begin{bmatrix}
    1 \\
    1 \\
    0 \\
    0 \\
    1 \\
    1 \\
    1 \\
    0 
\end{bmatrix}
\] (2)
where the x vector is the multiplicative inverse and being added to by the hexadecimal value 63.

For the affine transformation where we use the previous mentioned matrix multiplied by the element a,

\[ a = x^7 + x^6 + x^3 + x \]

to calculate the following:

\[
\begin{align*}
    a_0 &= a_0 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus 1 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1 \\
    a_1 &= a_0 \oplus a_1 \oplus a_5 \oplus a_6 \oplus a_7 \oplus 1 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0 \\
    a_2 &= a_0 \oplus a_1 \oplus a_2 \oplus a_6 \oplus a_7 \oplus 0 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
    a_3 &= a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_7 \oplus 0 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
    a_4 &= a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus 0 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0 \\
    a_5 &= a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus 1 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1 \\
    a_6 &= a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus 1 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1 \\
    a_7 &= a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus 0 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1 \\
\end{align*}
\]

Thus \( a' = x^7 + x^6 + x^5 + x^3 + x^2 + 1 = 11101101 = \text{ED} \)

The matrix multiplication for the S-box can now be calculated by the following algorithm:

1. Store the multiplicative inverse of the input number in two 8-bit unsigned temporary variables: s and x.
2. Rotate the value s one bit to the left; if the value of s had a high bit (eighth bit from the right) of one, make the low bit of s one; otherwise the low bit of s is zero.
3. Exclusive or the value of x with the value of s, storing the value in x
4. For three more iterations, repeat steps two and three; steps two and three are done a total of four times.
5. The value of x will now have the result of the multiplication.

Following this algorithm we generate the following S-box using the hexadecimal value 63.
And now the table can be easily read for replacing any hexadecimal value with its S-box inverse.

**Example:** 9A is now read as B8 after the S-box inverse is applied.

## 5 ByteSub Step

The ByteSub step is used on each value in the matrix by updating its value using an 8-bit substitution box, Rijndael’s S-box. The point of this operation to the matrix is provide a non-linear encryption to the chiper which keeps the chiper from being broken easily in a linear fashion as is done nowadays with block chipers.

Using the Galois field gives the chiper a strong non-linear property keeping simple algebraic attacks a bay. The ByteSub step is assisted by the S-box which as seen before is built using affine transformations.

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

This is arbitrary based upon whatever the matrix is being added by. This step is denoted by the statement \textit{ByteSub(State)}.

## 6 ShiftRow Step

The ShiftRow operation acts directly on the rows of the current state of the matrix. The rows are cyclically shifted over different offsets depending on the block length of the array.
The table show the different offsets for the matrix based off of its block length where a matrix would be shifted to the left by $C_1$ for Row 1, $C_2$ for Row 2, $C_3$ for Row 3, and Row 0 is not shifted. $Nb$ is equal to the block length of the matrix and each value corresponds to the number of bits used in the block divided by 32, i.e. $128 \div 32 = 4$.

**Example:** A $Nb = 4$ (128 bit) matrix before and after the ShiftRow Step:

<table>
<thead>
<tr>
<th>Before ShiftRow</th>
<th>After ShiftRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4 8 12</td>
<td>0 4 8 12</td>
</tr>
<tr>
<td>1 5 9 13</td>
<td>5 9 13 1</td>
</tr>
<tr>
<td>2 6 10 14</td>
<td>10 14 2 6</td>
</tr>
<tr>
<td>3 7 11 15</td>
<td>15 3 7 11</td>
</tr>
</tbody>
</table>

As we can see this is a simple row shift of the contents of each row by just a value of $C_1$, $C_2$, $C_3$ all of which vary depending on whether the matrix is representative of a 128, 192, or 256 bit block of values.

To get the inverse of this step simply apply the cyclic shift on the bottom three rows using Row 1 - $C_1$, Row 2 - $C_2$, and Row 3 - $C_3$ respectively.

### 7 MixColumn Step

This step is usually performed along with the ShiftRow step and is the main source of diffusion in the Jjndael cipher. The columns of the matrix are though of as polynomials in the Galois field and are multiplied modulo

$$x^4 + 1$$

with a fixed polynomial $c(x)$,

$$c(x) = 03x^3 + 01x^2 + 01 + x + 02$$

Now the MixColumn step can be performed by multiplying a coordinate vector in the Galois field by the following matrix:

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

(4)

Which can be written as the following for easei computation:

$$b_0 = 2a_0 \oplus 3a_1 \oplus a_2 \oplus a_3$$
$$b_1 = 2a_1 \oplus 3a_2 \oplus a_3 \oplus a_0$$
$$b_2 = 2a_2 \oplus 3a_3 \oplus a_0 \oplus a_1$$
$$b_3 = 2a_3 \oplus 3a_0 \oplus a_1 \oplus a_2$$
Example: The multiplication between a binary string 11001010 and 3 in the Galois field and applying module m(x) to reduce the new value to a 8 byte hexadecimal value:

\[
\begin{array}{c}
11001010 \\
\otimes 11 \\
11001010 \\
\oplus 11001010 \\
101011110 \\
\oplus 100011011 \\
1000101
\end{array}
\]

The new value, after applying the MixColumn step, is 45.

Using this step, denoted as \texttt{MixColumn(State)}, along with the \texttt{RowShift(State)} we can achieve a reliable diffusion for our cipher text.

8 AddRoundKey Step

A very simple step but can be seen as the final nail in the coffin. By adding the RoundKey to the transformed matrix we further distort the matrix causing it to be undecipherable without the formentioned key.

This step is done by applying bitwise EXOR arithmetic to the current matrix with the supplied RoundKey to generate a newly encrypted matrix that can only be unlocked by using the key.

9 The Round Transformation

Now all of the steps come together, to save space we will only give a general explanation of how the steps work. To get the desired output of \( e \) for the round input of \( a \).

\[
a_{i,j}
\]

denotes the \( i \)th row and the \( j \)th column. The first step is adding the RoundKey then the MixColumn Step,

\[
\begin{bmatrix}
    c_{0,j} \\
    c_{1,j} \\
    c_{2,j} \\
    c_{3,j}
\end{bmatrix}
\oplus
\begin{bmatrix}
    k_{0,j} \\
    k_{1,j} \\
    k_{2,j} \\
    k_{3,j}
\end{bmatrix}
\text{ and }
\begin{bmatrix}
    d_{0,j} \\
    d_{1,j} \\
    d_{2,j} \\
    d_{3,j}
\end{bmatrix}
\oplus
\begin{bmatrix}
    02 & 03 & 01 & 01 \\
    01 & 02 & 03 & 01 \\
    02 & 01 & 02 & 03 \\
    03 & 01 & 01 & 02
\end{bmatrix}
\begin{bmatrix}
    c_{0,j} \\
    c_{1,j} \\
    c_{2,j} \\
    c_{3,j}
\end{bmatrix}
\]

This is followed by the ShiftRow step and then the ByteSub transformation,

\[
\begin{bmatrix}
    c_{0,j} \\
    c_{1,j} \\
    c_{2,j} \\
    c_{3,j}
\end{bmatrix}
\text{ and } b_{i,j} = S \begin{bmatrix}
    a_{i,j}
\end{bmatrix}
\]

In this step the column indices need to be substituted with modulo Nb (either 4, 6, or 8).
\[
\begin{bmatrix}
e_{0,j} \\
e_{1,j} \\
e_{2,j} \\
e_{3,j}
\end{bmatrix} =
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 
\end{bmatrix}
\begin{bmatrix}
S \left[ a_{0,j} \right] \\
S \left[ a_{1,j} - C_1 \right] \\
S \left[ a_{2,j} - C_2 \right] \\
S \left[ a_{3,j} - C_3 \right]
\end{bmatrix}
\oplus
\begin{bmatrix}
k_{0,j} \\
k_{1,j} \\
k_{2,j} \\
k_{3,j}
\end{bmatrix}
\tag{7}
\]

Next step is simple matrix multiplication expressed here as a linear combination of vectors:

\[
\begin{bmatrix}
e_{0,j} \\
e_{1,j} \\
e_{2,j} \\
e_{3,j}
\end{bmatrix} =
\begin{bmatrix}
02 \\
01 \\
03
\end{bmatrix}
\begin{bmatrix}
03 \\
02 \\
01
\end{bmatrix}
\begin{bmatrix}
01 \\
03 \\
01
\end{bmatrix}
\begin{bmatrix}
01 \\
02 \\
01
\end{bmatrix}
\oplus
\begin{bmatrix}
k_{0,j} \\
k_{1,j} \\
k_{2,j} \\
k_{3,j}
\end{bmatrix}
\tag{8}
\]

The S values being used are vectors obtained by performing a table lookup on the input bytes in the S-box table. The rest is simple matrix multiplication based upon the values generated by the data set to be encrypted and the steps that went into encrypting it.

A look at the whole process in order is shown by:

- AddRoundKey(State)
- ByteSub(State)
- ShiftRows(State)
- MixColumn(State)
- AddRoundKey(State)
- ByteSub(State)
- ShiftRows(State)
- AddRoundKey(State)

To state to show all the steps and math incorporated in this execution would in itself take up as many pages as it has taken to explain how it works. So for space sake the practical execution of the Round Transformation is left up to the reader to calculate. Though that is the reason that it was designed to be written in code to avoid tedious calculations.

By using the steps multiple times the encryption can be considered in the terms of the AES one of the most secure ciphers that is in use today.

10 Conclusion

The Rijndael Block Cipher is incredible throughout in terms of runtime, protection, resistance to attack and many other such necessary issues that need to be addressed when designing a cipher. The Rijndael is still in use to this day, used in encrypting e-mails, important legal documents, etc. The math behind the Rijndael is simple once figured out but the combination of how the different steps act upon each other allows of the strength it has. The reader is encouraged to run and implement a coded version of Rijndael to see it in action as that is the best way to appreciate its purpose.

11 C Sharp Implementation

```csharp
using System;
```
using System.Text;
using System.IO;

class RijndaelSample

    static void Main()
    try
        // Create a new Rijndael object to generate a key
        // and initialization vector (IV).
        Rijndael RijndaelAlg = Rijndael.Create();

        // Create a string to encrypt.
        string sData = "Here is some data to encrypt."
        string FileName = "C:\Text.txt";

        // Encrypt text to a file using the file name, key, and IV.
        EncryptTextToFile(sData, FileName, RijndaelAlg.Key, RijndaelAlg.IV);

        // Decrypt the text from a file using the file name, key, and IV.
        string Final = DecryptTextFromFile(FileName, RijndaelAlg.Key, RijndaelAlg.IV);

        // Display the decrypted string to the console.
        Console.WriteLine(Final);
    catch (Exception e)
        Console.WriteLine(e.Message);
    Console.ReadLine();

    public static void EncryptTextToFile(String Data, String FileName, byte[] Key, byte[] IV)
    try
        // Create or open the specified file.
        FileStream fStream = File.Open(FileName, FileMode.OpenOrCreate);

        // Create a new Rijndael object.
        Rijndael RijndaelAlg = Rijndael.Create();

        // Create a CryptoStream using the FileStream
        // and the passed key and initialization vector (IV).
        CryptoStream cStream = new CryptoStream(fStream, RijndaelAlg.CreateEncryptor(Key, IV), CryptoStreamMode.Write);

        // Create a StreamWriter using the CryptoStream.
        StreamWriter sWriter = new StreamWriter(cStream);
try
    // Write the data to the stream
    // to encrypt it.
    sWriter.WriteLine(Data);

catch (Exception e)
    Console.WriteLine("An error occurred: 0", e.Message);

finally
    // Close the streams and
    // close the file.
    sWriter.Close();
cStream.Close();
fStream.Close();

catch (CryptographicException e)
    Console.WriteLine("A Cryptographic error occurred: 0", e.Message);

catch (UnauthorizedAccessException e)
    Console.WriteLine("A file error occurred: 0", e.Message);

public static string DecryptTextFromFile(String FileName, byte[] Key, byte[] IV)
try
    // Create or open the specified file.
    FileStream fStream = File.Open(FileName, FileMode.OpenOrCreate);

    // Create a new Rijndael object.
    Rijndael RijndaelAlg = Rijndael.Create();

    // Create a CryptoStream using the FileStream
    // and the passed key and initialization vector (IV).
    CryptoStream cStream = new CryptoStream(fStream, RijndaelAlg.CreateDecryptor(Key, IV), CryptoStreamMode.Read);

    // Create a StreamReader using the CryptoStream.
    StreamReader sReader = new StreamReader(cStream);

    string val = null;

    try
    // Read the data from the stream
    // to decrypt it.
    val = sReader.ReadLine();

    catch (Exception e)
```csharp
Console.WriteLine("An error occurred: 0", e.Message);

finally

    // Close the streams and
    // close the file.
    sReader.Close();
    cStream.Close();
    fStream.Close();

    // Return the string.
    return val;

catch (CryptographicException e)
    Console.WriteLine("A Cryptographic error occurred: 0", e.Message);
    return null;

catch (UnauthorizedAccessException e)
    Console.WriteLine("A file error occurred: 0", e.Message);
    return null;
```

12 References

1. Daemen, John, Rijmen, Vincent. **AES Proposal: Rijndael.**

2. Savard, John J. G. . **The Advanced Encryption Standard (Rijndael).**

