

1. Introduction

Securing the Internet presents great challenges and research opportunities. Potential applications such as Internet voting, universally available medical records and ubiquitous e-commerce are all being hindered because of serious security and privacy concerns. The epidemic of hacker attacks on personal computers and web sites only highlights the inherent vulnerability of the current computer and network infrastructure.

There various implementation for AES support the fact that different application required different implementation for the same algorithm. Previously many hardware implementation were proposed and was implemented they are 128,192,256 bit. Some application has strict area requirement and a compact AES implementation will be very useful to provide security as in the some embedded system cases. On the other side, some application highly needed the most level of security that can be obtained without carrying about the area /time limitation.

The first part of this chapter is devoted to Mathematical preliminaries required in encryption and decryption methods, in which we will present this area and explain how the benefits of math are exploited in cryptography. The second part is a detailed study of the next variation of AES algorithm called as 512 bit AES, it will be an introduction to the design of the method that we are going to propose in the next chapter.

2. Mathematical preliminaries

2.1. Addition

The addition of two elements in a finite field is achieved by “adding” the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed with the XOR operation (denoted by \oplus) modulo 2, so that $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, and $0 \oplus 0 = 0$. Consequently, subtraction of polynomials is identical to addition of polynomials. Alternatively, addition of finite field elements can be described as the modulo 2 addition of corresponding bits in the byte. For two bytes $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ and $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$, the sum is $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$, where each $c_i = a_i \oplus b_i$ $\{c_7 = a_7 \oplus b_7, c_6 = a_6 \oplus b_6, \dots, c_0 = a_0 \oplus b_0\}$. For example, the following expressions are equivalent to one another:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \text{ (Polynomial Notation).}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad (\text{binary notation}).$$

$$\{57\} \oplus \{83\} = \{d4\} \quad (\text{hexadecimal notation}). [18].$$

2.2. Multiplication

In the polynomial representation, multiplication in $GF(2^8)$ (denoted by \bullet) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is

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

or $\{11b\}$ in hexadecimal notation.

For example, $\{57\} \bullet \{83\} = \{c1\}$, because

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

And

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

The modular reduction by $m(x)$ ensures that the result will be a binary polynomial of degree less than 8, and thus can be represented by a byte. Unlike addition, there is no simple operation at the byte level that corresponds to this multiplication.

The multiplication defined above is associative, and the element $\{01\}$ is the multiplicative identity. For any non-zero binary polynomial $b(x)$ of degree less than 8, the multiplicative inverse of $b(x)$, denoted $b^{-1}(x)$, can be found as follows: the extended Euclidean algorithm is used to compute polynomials $a(x)$ and $c(x)$ such that

$$b(x)a(x) + m(x)c(x) = 1. \quad (2)$$

Hence, $a(x) \bullet b(x) \bmod m(x) = 1$, which means

$$b^l(x) = a(x) \bmod m(x). \quad (3)$$

Moreover, for any $a(x)$, $b(x)$ and $c(x)$ in the field, it holds that

$$a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x).$$

It follows that the set of 256 possible byte values, with XOR used as addition and the multiplication defined as above, has the structure of the finite field $GF(2^8)$. [18].

2.3. Polynomials with coefficients in $GF(2^8)$

Four-term polynomials can be defined - with coefficients that are finite field elements - as:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad (4)$$

which will be denoted as a word in the form $[a_0, a_1, a_2, a_3]$. Note that the polynomials in this section behave somewhat differently than the polynomials used in the definition of finite field elements, even though both types of polynomials use the same indeterminate, x . The coefficients in this section are themselves finite field elements, i.e., bytes, instead of bits; also, the multiplication of four-term polynomials uses a different reduction polynomial, defined below. The distinction should always be clear from the context.

To illustrate the addition and multiplication operations, let

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \quad (5)$$

define a second four-term polynomial. Addition is performed by adding the finite field coefficients of like powers of x . This addition corresponds to an XOR operation between the corresponding bytes in each of the words – in other words, the XOR of the complete word values.

Thus, using the equations of (4) and (5),

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \quad (6)$$

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

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \quad (7)$$

where

$$\begin{aligned}
 c_0 &= a_0 \cdot b_0 & c_4 &= a_3 \cdot b_1 + a_2 \cdot b_2 + a_1 \cdot b_3 \\
 c_1 &= a_1 \cdot b_0 + a_0 \cdot b_1 & c_5 &= a_3 \cdot b_2 + a_2 \cdot b_3 \\
 c_2 &= a_2 \cdot b_0 + a_1 \cdot b_1 + a_0 \cdot b_2 & c_6 &= a_3 \cdot b_3 \\
 c_3 &= a_3 \cdot b_0 + a_2 \cdot b_1 + a_1 \cdot b_2 + a_0 \cdot b_3
 \end{aligned} \tag{8}$$

The result, $c(x)$, does not represent a four-byte word. Therefore, the second step of the multiplication is to reduce $c(x)$ modulo a polynomial of degree 4; the result can be reduced to a polynomial of degree less than 4. For the AES algorithm, this is accomplished with the polynomial $x^4 + 1$, so that

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4} \tag{9}$$

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

$$D(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \tag{10}$$

With

$$\begin{aligned}
 d_0 &= (a_0 \cdot b_0) + (a_3 \cdot b_1) + (a_2 \cdot b_2) + (a_1 \cdot b_3) \\
 d_1 &= (a_1 \cdot b_0) + (a_0 \cdot b_1) + (a_3 \cdot b_2) + (a_2 \cdot b_3) \\
 d_2 &= (a_2 \cdot b_0) + (a_1 \cdot b_1) + (a_0 \cdot b_2) + (a_3 \cdot b_3) \\
 d_3 &= (a_3 \cdot b_0) + (a_2 \cdot b_1) + (a_1 \cdot b_2) + (a_0 \cdot b_3)
 \end{aligned} \tag{11}$$

When $a(x)$ is a fixed polynomial, the operation defined in equation (10) can be written in matrix form as:

$$\begin{pmatrix} \mathbf{d0} \\ \mathbf{d1} \\ \mathbf{d2} \\ \mathbf{d3} \end{pmatrix} = \begin{pmatrix} \mathbf{a0} & \mathbf{a3} & \mathbf{a2} & \mathbf{a1} \\ \mathbf{a1} & \mathbf{a0} & \mathbf{a3} & \mathbf{a2} \\ \mathbf{a2} & \mathbf{a1} & \mathbf{a0} & \mathbf{a3} \\ \mathbf{a3} & \mathbf{a2} & \mathbf{a1} & \mathbf{a0} \end{pmatrix} \begin{pmatrix} \mathbf{b0} \\ \mathbf{b1} \\ \mathbf{b2} \\ \mathbf{b3} \end{pmatrix} \tag{12}$$

Because $x^4 + 1$ is not an irreducible polynomial over $GF(2^8)$, multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that does have an inverse:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \quad (13)$$

$$a^{-1}(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. [18]. \quad (14)$$

3. AES Algorithm Using 512 Bit Key

The original AES was selected through a public process that was in fact a contest conducted by the NIST, the US Government's official standards organization. Fifteen candidates submitted symmetric encryption algorithms that met NIST requirements. Five of these contestants made it into the AES finals. The five finalists were all regarded to have similar security, but the submission from Rijndael was selected to become AES as it offered the best performance across all architectures.

The new AES 512 bit can be used when higher level of security throughput are required without increasing overall design area as compared to the original 128-bit AES algorithm. The new algorithm consist of the structure, which is similar to the original AES algorithm. It has a slight difference in the plaintext size and key size using input of 512 bit instead of 128 bit, which has impact on the whole algorithm structure as it will be discussed in details later, the procedure to generate the new 512 bit key will be presented as well. The AES algorithm consist of four major operations are performed during each round: byte substitution, shifting rows, mixing columns and finally adding the round key.

3.1. Cipher

The 512-bit AES algorithm has four main different byte based transformations. The first transformation is the byte substitution, which substitutes the value of 512 bit, and this is achieved by using parallel s-boxes. The second transformation is shifting rows that shift the rows of the output from previous step by an offset equal to the row numbered. The third transformation is mixing column, where each column of the output from previous step is multiplied by different value. The final transformation in the round is adding round key to the result of this round.

3.1.1. SubBytes()

The 512 bits input plaintext are organized in array of 64 bytes and are substituted by values obtained from substitution boxes. This is done to achieve more security according to diffusion-confusion Shannon's principles for cryptographic algorithm design. [19].

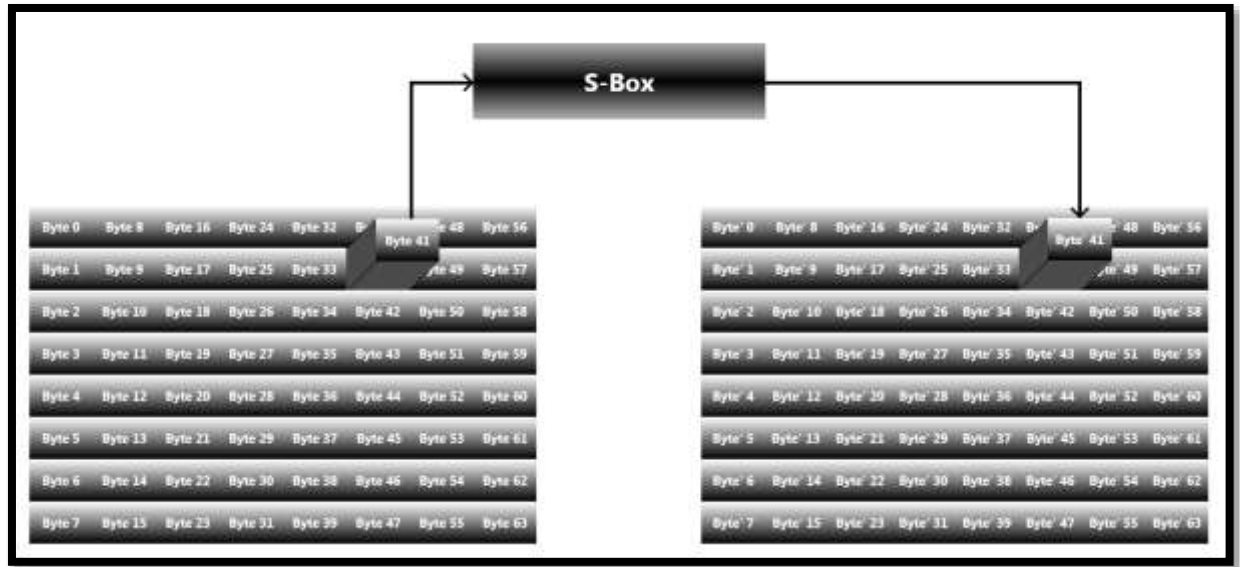


Figure II.1: Byte substitution.

The S-Box used in SubBytes() transformation of 512-bit AES is the same one used in SubBytes() transformation for 128-bit AES, it is presented in hexadecimal form in Fig II.2. [19]. For example, if $S_{1,1} = \{53\}$, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Fig II.2 this would be result in $S'_{1,1}$ having a value of {ed}.

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

Figure II.2: Substitution values in hexadecimal form.

3.1.2. ShiftRows()

After the original 512-bit data is substituted with values from the S-boxes, the rows of the resulting matrix are shifted in a process called Shift Row transformation. What happened in this part is that the bytes in each row in the input data matrix will be rotated left. The number of left rotations is not the same in each row, and it can be determined by the row number. For example, row number zero is not shifted, the first row is shifted by one byte, and so on. [19].

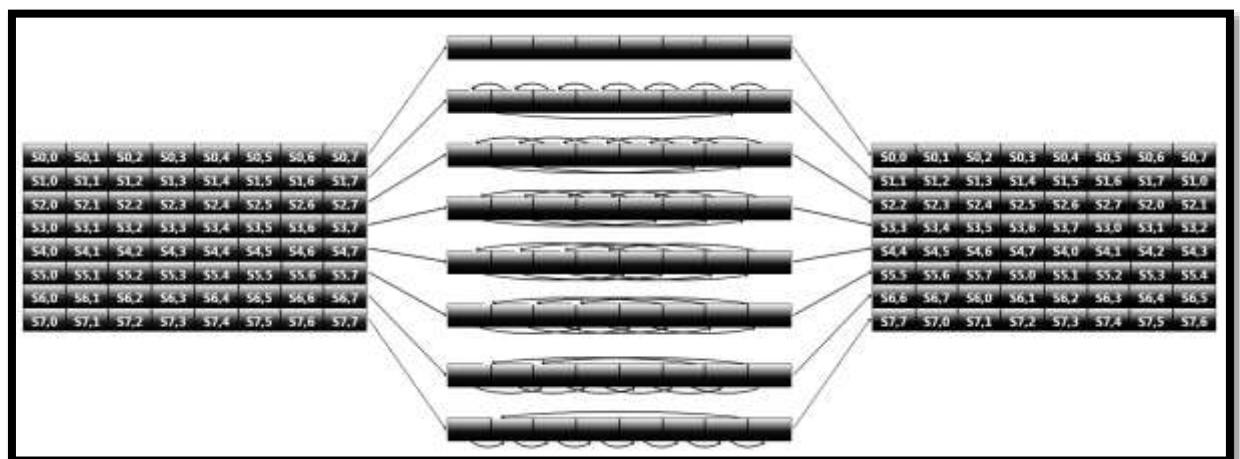


Figure II.3: Shift rows.

3.1.3. MixColumns()

Now, and after the rows of the input data are rotated left by different offsets, an operation must be applied to the columns of the data matrix. The Mix Column transformation multiplies the columns of the data matrix by a pre-defined matrix. The AES-512 and original AES process the data in bytes basis. Each byte is considered as polynomials over GF (2⁸) with 8 terms. [19].

$$\begin{pmatrix} 02 & 01 & 03 & 01 & 01 & 01 & 01 & 01 \\ 01 & 03 & 01 & 01 & 01 & 01 & 01 & 03 \\ 03 & 01 & 01 & 01 & 01 & 01 & 02 & 01 \\ 01 & 01 & 01 & 01 & 01 & 02 & 01 & 03 \\ 01 & 01 & 01 & 01 & 02 & 01 & 03 & 01 \\ 01 & 01 & 01 & 02 & 01 & 03 & 01 & 01 \\ 01 & 01 & 02 & 01 & 03 & 01 & 01 & 01 \\ 01 & 02 & 01 & 03 & 01 & 01 & 01 & 01 \end{pmatrix} \begin{pmatrix} S_{0,0} \\ S_{1,0} \\ S_{2,0} \\ S_{3,0} \\ S_{4,0} \\ S_{5,0} \\ S_{6,0} \\ S_{7,0} \end{pmatrix} = \begin{pmatrix} S_{0,0} \\ S_{1,0} \\ S_{2,0} \\ S_{3,0} \\ S_{4,0} \\ S_{5,0} \\ S_{6,0} \\ S_{7,0} \end{pmatrix} \quad (15)$$

The Mix Column operation (shown in Figure 3) multiplies the columns in the data matrix with a fixed polynomial of $a(x)$, given by:

$$a(x) = [02]x^7 + [01]x^6 + [03]x^5 + [01]x^4 + [01]x^3 + [01]x^2 + [01]x^1 + [01]x^0 \quad (16)$$

The multiplication result is taken (modulo $p(x) = x^8 + 1$) to keep the resulting polynomial with degree less than 8.

In the inverse of the Mix Column transformation, the input array is multiplied with the inverse of the polynomial $a(x)$, denoted as $a^{-1}(x)$, which is given by:

$$a^{-1}(x) = [0E]x^7 + [01]x^6 + [09]x^5 + [01]x^4 + [0D]x^3 + [01]x^2 + [0B]x^1 + [01]x^0 \quad (17)$$

3.1.4. AddRoundKey()

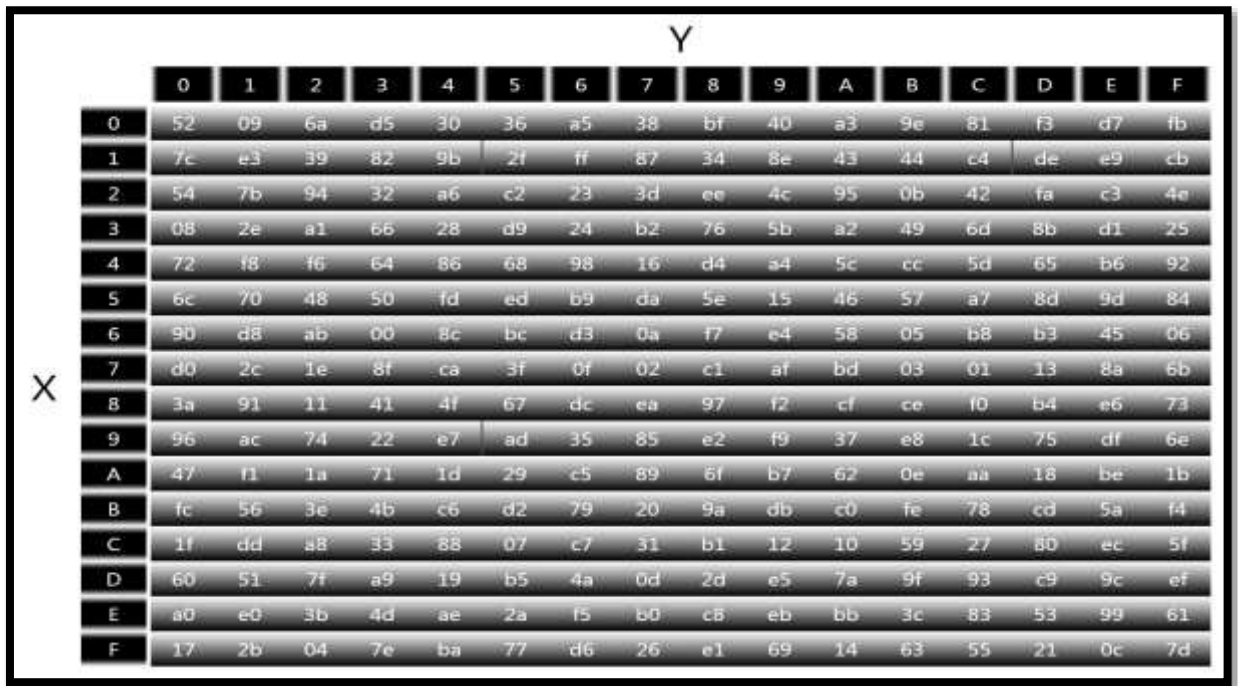
To make relationship between the key and the cipher text more complicated and to satisfy the confusion principal, the AddRoundKey() operation is performed. This addition step takes the resulting data matrix from the previous step and performs on it a bitwise XOR operation with the sub key of that specific round (addition operation in GF (2ⁿ)). We must mention that the round key is 512 bits that is arranged in a square matrix of eight columns where each column has 8 bytes. [19].

3.2. Inverse cipher

The Cipher transformations in Sec. 5.1 can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher are `InvShiftRows()`, `InvSubBytes()`, `InvMixColumns()`, and `AddRoundKey()`, process the State and are described in the following subsections.

3.2.1. `InvSubBytes()`

It 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 followed by taking the multiplicative inverse in $GF(2^8)$. [18]. The inverse S-box used in the `InvSubBytes()` transformation is presented in Fig II.4:



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

Figure II.4: Inverse values of `SubBytes()` function in hexadecimal form.

3.2.2. InvShiftRows()

It is the inverse of the ShiftRows() transformation (shifting the rows). the bytes in each row in the input data matrix will be rotated right. The number of left rotations is not the same in each row, and it can be determined by the row number. For example, row number zero is not shifted; the first row is shifted by one byte, and so on. [18].

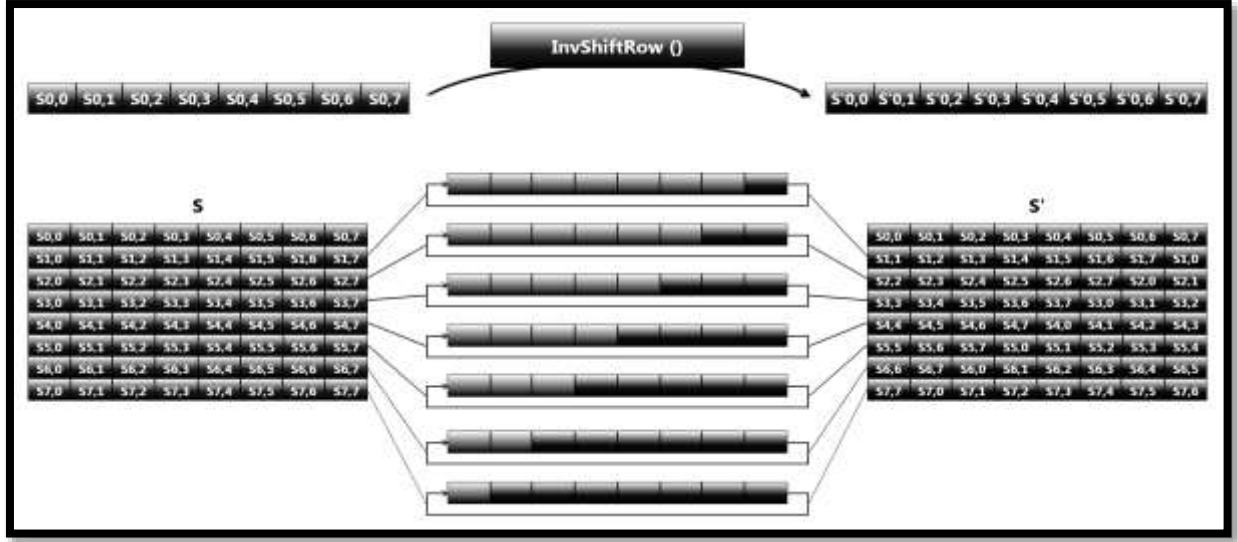


Figure II.5: Inverse of ShiftRows function.

3.2.3. InvMixColumns()

InvMixColumns() is the inverse of the MixColumns() transformation. It operates on the State column-by-column, treating each column as an eight-term polynomial as described in Sec. 2.3. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by [18].

$$a^{-1}(x) = [0E]x^7 + [01]x^6 + [09]x^5 + [01]x^4 + [0D]x^3 + [01]x^2 + [0B]x^1 + [01]x^0 \quad (18)$$

3.2.4. Inverse of AddRoundKey()

AddRoundKey(), which was described in Sec. 3.1.4, is its own inverse, since it only involves an application of the XOR operation. [18].

4. Key management

The 128-bit AES key is considered secure compared to other existing symmetric cipher algorithm. It is widely used in many application where the security is very important. The 512-bit AES algorithm provides even more security and double throughput, more security comes from using larger key size, and more throughput comes from using four times larger block size than the block size used in the original AES.

4.1. Key length requirements

Previously many hardware implementations were proposed and implemented; the keys are 128, 192, and 256 bits. There various implementation for AES support the fact that different application required different implementation for the same algorithm. Some application has strict area requirement and a compact AES implementation will be very useful to provide security as in the some embedded system cases. On the other side, some application highly needed the most level of security that can be obtained without carrying about the area and the time limitation. An implementation of the original AES algorithm supports at least one of the three key lengths: 128, 192, or 256 bits. The new 512-bit AES algorithm supports only one key length fixed by 512-bits, so it avoids any differences about key in order to promote the interoperability of algorithm implementations. [19].

4.2. Key expansion and rounds

The 512-bit input key of the new AES-512 algorithm is used to generate ten sub-keys for each of the ten AES rounds. The round keys expansion process involves arranging the original 512-bits input key into eight words of eight bytes each. After that, the round keys expansion is performed according to the following equations:

$$W(I) = W(i-8) \text{ XOR } W(I-1) \text{ } I \text{ is not a multiple of } 8 \quad (19)$$

$$W(I) = W(i-8) \text{ XOR } T(W(I-1)) \text{ } I \text{ is a multiple of } 8 \quad (20)$$

Where the $T(I)$ transformation is defined as:

$$T(I) = \text{ByteSub}(\text{ShiftLeft}(W(I))) \text{ XOR } \text{RoundConst} \quad (21)$$

The round constant is defined by the following equation:

$$\text{RoundConst} = 00000010^{(i-8)/8} \quad (22)$$

I is the round number.

Table II.1 shows the round constants for all rounds in AES-512.

Round	I	Round Constant
1	8	0100000000000000
2	16	0200000000000000
3	24	0400000000000000
4	32	0800000000000000
5	40	1000000000000000
6	48	2000000000000000
7	56	4000000000000000
8	64	8000000000000000
9	72	1B00000000000000
10	80	3600000000000000

Table II.1: Round constant for 512-bit AES rounds.

The round structure of the AES-512 algorithm (shown in Figure II.6) uses the transformation defined in the previous section. First, byte substitution is performed on 512 bits data, followed by row rotation according to the row number, where 0-7 left rotations are performed in this step. Then, the columns are multiplied by the new defined matrix column by column in the Mix Column transformation (except in the 10th round). The last operation will be the bitwise XORing with the round key expanded using the key expansion process. The output at of the 10th round will be the 512-bit encrypted message. [19].

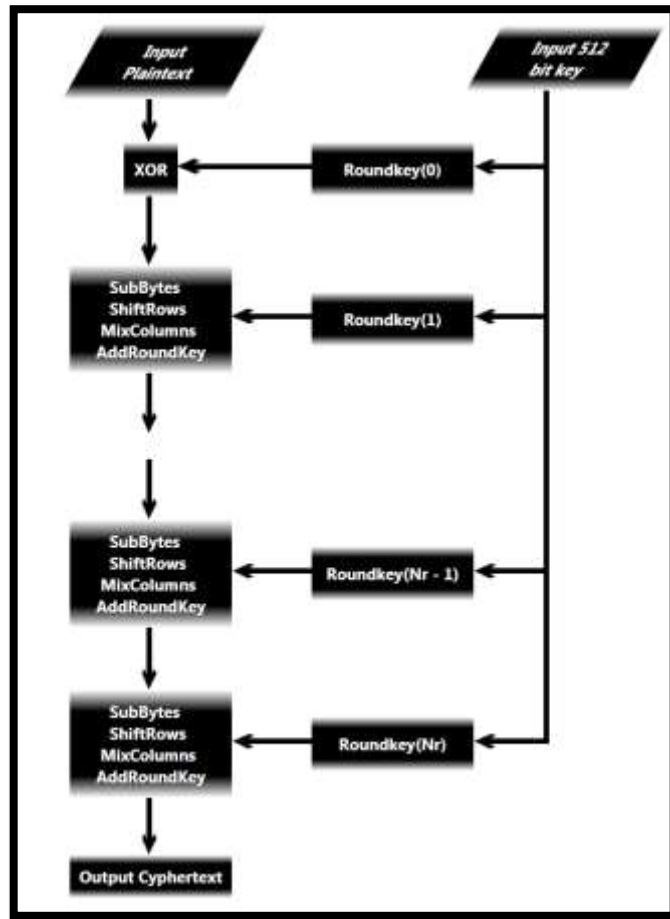


Figure II.6: Round Structure of AES.

5. Advantages and disadvantages of 512-bits AES

While the data is encrypted with 512-bit AES, the number of sub-keys is increased to ten keys and minimize the rounds of processing comparing with the original AES with its different key sizes. The big Key size provides a high level of security to the data because it is difficult for the hacker/attacker to retrieve an information encrypted by using a big key length.

The only disadvantage of 512-bit AES is the need for more design area. It takes a lot of space more than the original AES, because its key block size and data block size are bigger than the others in original AES by four times.

6. Complexity analysis

The safety evaluation of a block cipher algorithm is a very difficult task, because it requires the calculation of the complexity of the currently known attacks, including

exhaustive research, linear cryptanalysis and differential cryptanalysis. If the complexity of the exhaustive search is very easy to evaluate, the complexity of statistical attacks is very difficult to calculate since it requires a thorough knowledge of the development of these attacks. However, to design a symmetric cipher algorithm resistant to block these attacks, it is highly advisable to strengthen the algorithm by permutations (at the beginning, the middle and the end) as well as substitutions. Because the statistical attacks are attacks that follow the running encryption / decryption of several couples (plaintext, ciphertext) in all procedures of algorithms to find a statistical relationship between plaintext and ciphertext, But the existence of these permutations and substitutions (after several turns) makes this more difficult strategy to implement.

The exhaustive search complexity depends essentially on the size of the used key, if this size is L in this case the complexity is $O(2^{L-1})$.

For a 128-bit key, the complexity of this attack is $O(2^{128})$, the number of possible keys is:

340282366920938463463374607431768211456 key.

Whereas with a key of 256 bits, the complexity is $O(2^{256})$ That is to say:

115792089237316195423570985008687907853269984665640564039457584007913129639
936 possible keys.

In the case of the 512-bit AES algorithm, we have four 128-bit keys ($4 * 128\text{-bit} = 512$). Therefore, the complexity will be $O(2^{512})$. It means that the number of possible keys is much much bigger than the previous numbers mentioned in his section.

7. Experiences and results (memory space and encryption time)

When it comes to the criteria of the memory space, there are two ways to effect an ideal comparison. The first one is to measure the memory space spent by all the functions of the encryption process. In this case, to encrypt one data block of 512-bit size, the 512-bit AES uses four functions that occupy together 2048 bits ($512\text{-bit} \times 4$). While the 128-bit AES allows to occupy only 512-bits ($128\text{-bit} \times 4$), which is four times less than the other one. The second one is to measure the memory space spent by each function of the encryption process. In this case, we have to take in consideration the size of data block that each function consume at the same time for each algorithm. In 512-bit AES, all the functions work with 512-bit data block size and produce 512-bit data block size. While in 128-bit AES, all the

functions work with 512-bit data block size and produce 512-bit data block size. Therefore, as we can see, the original AES is better than the 512-bit AES when it comes to the criteria of the memory consummation.

When it comes to the encryption time, The 128-bit AES encrypts the data spending time less than what the 512-bit AES spend (almost the half of time), and the difference is getting bigger in each time the data gets bigger. The only thing that makes the 512-bit AES better than the 128-bit AES is the level of security that it provides.

8. Explanatory example

The plaintext: « I have a secret ! ».

The key:

« efefaaaafbfb43434d3333858545454d02027f7f5050f9f99fa8a851513c3c9fffbfb4343efefaaaa8545454d4d3333855050f9f902027f7f513c3c9f9fa8a851 ».

Before everything, zeros must be added to complete the 512 bits, than, the plaintext must be converted to hexadecimal. We will have this:

[illegible]

First step is SubBytes() transformation, which is transforming data using an S-Box, the result is:

[illegible]

Second step is shifting rows to the left side, as a result we have:

[illegible]

Third step is the multiplication process, which is called MixColumns() transformation, the result is:

« 897d655d00000000bdb33e00000000bb5cc13e00000000c52153f300000000c52143f3000000001e80643d000000043fa593f00000000013cdf0a800000000 ».

Fourth step is adding the key (AddRoundKey() transformation):

« 6090c0f0f0f04040f08000b080404040b050b0405050f0f05080f0a05030309030d000b0e0e0a0a090c020704030308010a0a0c00000707040f0c03090a0a050 ».

Finally, converting the block to characters, obviously we see that data is completely changed, and ofcourse it is not readable:

« ` • Àððð@ @ð€ 0°€ @ @ @°P°@PPððP€ ð P00 • 0Ð0°àà • À p@00€ À00pp@ðÀ0 • P ».

9. Conclusion

Due to the increasing needs for secure communications, a more safe and secure cryptographic algorithms has to be proposed and implemented. The Advanced Encryption Standard (AES-128bit) is widely used nowadays in many applications.

In this chapter, we presented a Mathematical preliminaries required in encryption and decryption methods, in which we presented this area and explained how the benefits of math are exploited in cryptography. Then, we presented the 512-bit AES-512 that works with 512-bit input block and 512-bit key size compared with 128-bit in the original AES-128 algorithm.

The extra increase in area can be tolerated and makes the proposed algorithm ideal for applications in which high level of security is required such as in multimedia communications.