An Approach to Security Using Rijndael Algorithm

Srinivasan Nagaraj
Sr.Asst. Professor
Dept. of CSE, GMRIT
Rajam – 532127, AP, India

Kishore Bhamidipati
Asst. Professor
Dept. of IT, GMRIT
Rajam – 532127, AP, India

G Apparao
Asst. Professor
Dept. of CSE, GITAM Univ.
Vizag – 530045, AP, India

ABSTRACT
The existing system consisted of files with literally no file security. The main issue of Reading or tapping data is secrecy and confidentiality. Confidentiality has always played an important role in diplomatic and military matters. Often Information must be stored or transferred from one place to another without being exposed to an opponent or enemy. The main aim of presenting this paper is to encrypt a java file using Rijndael Algorithm. The first aspect that has to be considered in our paper is file security and the need for file security. Key management is also related to Confidentiality. This deals with generating, distributing and storing keys .File security must be implemented so as to eliminate the problems like unauthorized access, execution of commands illicitly, destructive behavior and confidentiality reaches.

Keywords
Bytes, Cipher, state, finite field Addition, multiplication.

1. INTRODUCTION
Standards like Rijndael was to be implemented due to the following factors against which several security measures had to be taken up like. Reading or tapping data, Manipulating and modifying data and Illegal use of files, Corrosion of data files, Distortion of data transmission and Disturbance of the operation of equipment or systems. Also Computer files and networks must be protected against intruders and Unauthorized. That includes File Security, Cryptography and Private-Key-Encryption, Key Management.

1.1 Rijndael Features
Designed to be efficient in both hardware and software across a variety of platforms. Uses a variable block size, 128, 192, 256-bits, key size of 128, 192-, or 256- bits.

Variable number of rounds (10,12, 14)

128-bit round key used for each round:

– 128 bits = 16 bytes = 4 words

– needs Nr+1 round keys for Nr rounds – needs 44 words for 128-bit key (10 rounds)

2. IMPLEMENTING RIJNDAEL

2.1 Notation and Conventions

2.2 Rijndael Inputs and Outputs
The input, the output and the cipher key for Rijndael are each bit sequences containing 128, 192 or 256 bits with the constraint that the input and output sequences have the same length. A bit is a binary digit, 0 or 1, while the term 'length' refers to the number of bits in a sequence. In general the length of the input and output sequences can be any of the three allowed values but for the Advanced Encryption Standard (AES) the only length allowed is 128. However, both Rijndael and AES allow cipher keys of all three lengths. The individual bits within sequences will be enumerated starting at zero and increasing to one less than the length of the sequence. The number i associated with a bit, called its index, is hence in one of the three ranges 0 £ i < 128, 0 £ i < 192 or 0 £ i < 256 depending on the length of the particular sequence in question.

2.3 Bytes
A byte in Rijndael is a group of 8 bits and is the basic data unit for all cipher operations. Such bytes are interpreted as finite field elements using polynomial representation, where a byte b with bits b0 b1 …b7represents the finite field element:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i$$

The values of bytes will be presented in binary as a concatenation of their inputs (0 or 1) between braces. Hence {011000011} identifies a specific finite field element. Unless specifically indicated, bit patterns will be presented with higher numbered bits to the left. It is also convenient to denote byte values using hexadecimal notation, with each of two groups of four bits being denoted by a character as follows

bit pattern	character	bit pattern	character	bit pattern	character	bit pattern	character
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

Hence the value {011000011} can also be written as {63}, where the character denoting the 4-bit group containing the higher numbered bits is again to the left. Some finite field operations utilize a single additional bit (b8) to the left of an 8-bit byte. Where this bit is present it will appear immediately to the left of the left brace, for example, as in 1{1b}.

2.4 Arrays of Bytes
All input, output and cipher key bit sequences are represented as one-dimensional arrays of bytes where byte n consists of bits 8n to 8n+7 from the sequence with bit 8n+i in the sequence mapped to bit 7-i in the byte for 0 <= i < 8. For a sequence denoted by the symbol a, the n'th byte will be referred to using either of the two notations an or a[n], with n in one of the ranges 0 <=n < 16, 0 <=n < 24 or 0 <=n < 32.

2.5 The Rijndael State
Internally Rijndael operates on a two dimensional array of bytes called the state that contains 4 rows and Nc columns, where Nc is the input sequence length divided by 32. In this state array, denoted by the symbol s, each individual byte has two indexes: its row number r, in the range 0 <=r < 4, and its column number c, in the range 0 <=c < Nc, hence allowing it to be referred to either as c r s , or s[r, c]. For AES the range for c is 0 <=c < 4 since Nc has a fixed value of 4. At the start (end) of an encryption or decryption operation the bytes of the cipher input (output) are copied to (from) this state array in the order shown in Figure 1.

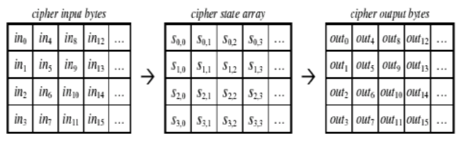


Figure 1 – Input to, and output from, the cipher state array

Hence at the start of encryption or decryption the input array in is copied to the state array according to the scheme:

s[r, c] = in[r + 4c] for 0 £ r < 4 and 0 £ c < Nc

and when the cipher is complete the state is copied to the output array out according to:

out[r + 4c] = s[r, c] for 0 £ r < 4 and 0 £ c < Nc

Arrays of 32-bit Words

The four bytes in each column of the state can be thought of as an array of four bytes indexed by the row number r or as a single 32-bit word (bytes within all 32-bit words will always be enumerated using the index r). The state can hence be considered as a one dimensional array of words for which the column number c provides the array index. The key schedule for Rijndael, described below, is an array of 32-bit words, denoted by the symbol k, with the lower elements initialized from the cipher key input so that byte 4i+r of the key is copied into byte r of key schedule word k[i]. The cipher iterates through a number of cycles, called rounds, each of which uses Nc words from this key schedule. Hence the key schedule can also be viewed as an array of round keys, each of which consists of an Nc word sub-array. Hence word c of round key n, which is k[Nc * n + c], will also be referred to using two dimensional array notation as either k[n,c] or kn,c . Here the round key for round n as a whole, an Nc word sub-array, will sometimes be referred to by replacing the second index with '-' as in k[n,-] and -, n k.

3. ECRYPTION

3.1 Finite Field Addition

The addition of two finite field elements is achieved by adding the coefficients for corresponding powers in their polynomial representations, this addition being performed in GF(2), that is, modulo 2, so that 1 + 1 = 0. Consequently, addition and subtraction are both equivalent to an exclusive-or operation on the bytes that represent field elements. Addition operations for finite field elements will be denoted by the symbol Å. For example, the following expressions are equivalent.

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)\equiv x^7+x^6+x^4+x^2$$

(polynomial notation)

{01010111} Å {10000011} _ {11010100}

(binary notation)

{57} Å {83} _ {d4}   (Hex Notation)

3.2 Finite Field Multiplication

Finite field multiplication is more difficult than addition and is achieved by multiplying the polynomials for the two elements concerned and collecting like powers of x in the result. Since each polynomial can have powers of x up to 7, the result can have powers of x up to 14 and will no longer fit, within a single byte. This situation is handled by replacing the result with the remainder polynomial after division by a special eighth order irreducible polynomial, which, for Rijndael, is:

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

Since this polynomial has powers of x up to 8 it cannot be represented by a single byte and will be written as either 1{00011011} or 1{1b} as indicated earlier. This process is illustrated in the following example of the product {57} · {83} _ {c1} (where · is used to represent finite field multiplication):

$$\begin{array}{ll}
(x^6+x^4+x^2+x+1)\bullet(x^7+x+1)\rightarrow & \\
(x^6+x^4+x^2+x+1)\bullet x^7= & x^{13}+x^{11}+x^9+x^8+x^7+ \\
(x^6+x^4+x^2+x+1)\bullet x= & \qquad\qquad x^7\quad +x^5\quad +x^3+x^2+x \\
(x^6+x^4+x^2+x+1)\bullet 1= & \qquad\qquad\quad x^6\quad +x^4\quad +x^2+x+1 \\
\hline
 & x^{13}+x^{11}+x^9+x^8\quad +x^6+x^5+x^4+x^3\quad +1
\end{array}$$

This intermediate result is now divided by m(x) above:

$$\begin{array}{ll}
 & x^{13}+x^{11}+x^9+x^8\quad +x^6+x^5+x^4+x^3\quad +1 \\
(x^8+x^4+x^3+x+1)\bullet x^5= & x^{13}\quad +x^9+x^8\quad +x^6+x^5 \\
\hline
\text{subtract to give intermediate remainder} & x^{11}\qquad\qquad +x^4+x^3\quad +1 \\
(x^8+x^4+x^3+x+1)\bullet x^3= & x^{11}\quad +x^7+x^6\quad +x^4+x^3 \\
\hline
\text{subtract to give the final remainder} & \qquad x^7+x^6\qquad\qquad +1
\end{array}$$

Multiplication is associative, and there is a neutral element {01}; for any binary polynomial b(x) of degree less than 8, the extended Euclidean algorithm can be used to compute polynomials a(x) and c(x), such that:

$$b(x)\bullet a(x)\oplus m(x)\bullet c(x)=1$$
$$a(x)\bullet b(x)\bmod m(x)=1$$

Which shows that the polynomials a(x) and b(x) are mutual inverses. Furthermore:

$$a(x)\bullet(b(x)\oplus c(x))=a(x)\bullet b(x)\oplus a(x)\bullet c(x)$$

It hence follows that the set of 256 byte values, with the XOR as addition and multiplication as defined above has the structure of the finite field GF(256).

3.3 Multiplication by Repeated Shifts

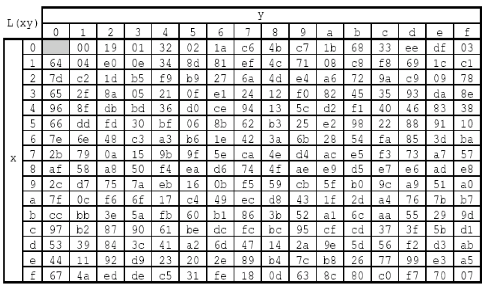
The finite field element {00000010} is the polynomial x, which means that multiplying another element by this value increases all its powers of x by 1. This is equivalent to shifting its byte representation up by one bit so that the bit at position i move to position i+1. If the top bit is set prior to this move it will overflow to create an x8 term, in which case the modular polynomial is added to cancel this additional bit, leaving a result that fits within a single byte. For example, multiplying {11001000} by x, that is {00000010}, the initial result is 1{10010000}. The 'overflow' bit is then removed by adding 1{00011011}, the modular polynomial, using an exclusive-or operation to give a final result of {10001011}.By repeating this process, a finite field element can be multiplied by all powers of x from 0 to 7. Multiplication of this element by any other field element can then be achieved by adding the results for the appropriate powers of x. For example, Table 1 carries out this calculation for the product of the field elements 57} and {83} to give {c1}.

p	$\{57\}\bullet x^p$	$\oplus\ m(x)$	$\{57\}\bullet x^p$	{83}	⊕ to result	result
0	{01010111}		{01010111}	1	{01010111}	{01010111}
1	{10101110}		{10101110}	1	{10101110}	{11111001}
2	1{01011100}	1{00011011}	{01000111}	0		
3	{10001110}		{10001110}	0		
4	1{00011100}	1{00011011}	{00000111}	0		
5	{00001110}		{00001110}	0		
6	{00011100}		{00011100}	0		
7	{00111000}		{00111000}	1	{00111000}	{11000001}

Table 1 – Finite field multiply {57} • {83}

3.4 Finite Field Multiplication Using Tables

When certain finite field elements (known as generators) are repeatedly multiplied to produce a list of their powers, gp, they progressively generate all 255 non-zero elements in the field. When p reaches 256 the original field element recurs, indicating that g255 is equal to {01}. The p values for each field element can be thought of as logarithms and these provide a way of converting multiplication into addition. Hence the two elements a = g a and b = g b have the product a · b = g a + b. With a 'logarithm' table listing the power of the generator for each finite field element can hence find the powers a and b corresponding to the elements a and b and add these values to find the power of g for the result. A reverse table can then be used to look up the product element. Since the two initial power values can each be as high as 255, their sum may be greater than 255 but if this occurs, 255 can be subtracted from the value to bring it into the range of the tables because g255 = {01}. Although decimal exponents have been used in this explanation, all exponents in what follows are in hexadecimal notation.



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

Table 2 – 'Logs' – L values such that {xy} = $\{03\}^L$ for a given a finite field element {xy}

For the Rijndael field {03} is a generator that yields Table 2 . Using the previous example, Table 2 shows that {57} = {03}(62) and {83} = {03}(50) (where the brackets on the exponents identify them as hexadecimal numbers). This gives the product as {57} · {83} = {03}(62) + (50) and since (62) + (50) = (b2) in hexadecimal, These tables can also be used to find the inverses of field elements since g(X) has the inverse g(ff)-(X). Hence the element {af} = {03}(b7) has the inverse g(ff)-(b7) = g(48) = {62}. All elements except {00} have inverses.

3.5 Polynomials with Coefficients in GF (256)

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

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

where the four coefficients, each represented by a byte, will be denoted as a 32-bit word in the form [a3 , a2 , a1 , a0]. With a second polynomial:

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

addition can be performed by adding the finite field coefficients of like powers of x, which corresponds to an XOR operation between the corresponding bytes in each of the words or an XOR of the complete 32-bit word values Multiplication is achieved by algebraically expanding the polynomial product and collecting like powers of x to give:

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

where:

$$c_0 = a_0 \bullet b_0 \qquad c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$
$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \qquad c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$
$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \qquad c_5 = a_3 \bullet b_3$$
$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

with · and Å representing finite field multiplication and addition (XOR) respectively. This result requires six bytes to represent its coefficients but it can be reduced modulo a degree 4 polynomial to produce a result that is of degree less than 4. In Rijndael the polynomial used is x4 + 1 and reduction produces the following polynomial coefficients:

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$
$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$
$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$
$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

If one of the polynomials is fixed, this can conveniently be written in matrix form as:

$$\begin{bmatrix} d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_3 & a_0 & a_1 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_1 & a_2 & a_3 & a_0 \end{bmatrix} \begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix}$$

Because x4 + 1 is not an irreducible polynomial, not all polynomial multiplications are invertible. For Rijndael, however, a polynomial that has an inverse has been chosen:

$$a(x) = \{03\}\,x^3 + \{01\}\,x^2 + \{01\}\,x + \{02\}$$
$$a^{-1}(x) = \{0b\}\,x^3 + \{0d\}\,x^2 + \{09\}\,x + \{0e\}$$

Another polynomial that Rijndael uses has a0 = a2 = a3 = {00} and a1 = {01}, which is the polynomial x. Inspection of above will show that its effect is to form the output word by rotating the bytes in the input word so that [b3 , b2 , b1 , b0] is transformed into [b2 , b1 , b0 , b3], with bytes moving to higher index positions and the top byte wrapping round to the lowest position. Higher powers of x correspond to the other cyclic permutations of the four bytes within a 32-bit word. The RootWord function that is used in the key schedule corresponds to x3.

3.6 The Cipher

At the start of the cipher the cipher input is copied into the internal state using the conventions described in above. An initial round key is then added and the state is then transformed by iterating a round function in a number of cycles. The number of cycles Nn varies with the key length and block size. On completion the final state is copied into the cipher output using the same conventions. The round function is parameterized using a round key which consists of an Nc word sub array from the key schedule. The latter is considered either as a one-dimensional array of 32-bit words or an array of round keys with a structure and initialization as described in above. In general the length of the cipher input, the cipher output and the cipher state, Nc, measured in multiples of 32 bits, is 4, 6 or 8 but the AES standard only allows a length of 4. The length of the cipher key, Nk, again measured in multiples of 32 bits, is also 4, 6 or 8, all of which are allowed by both Rijndael and the AES standard. Here the key schedule is treated as an array of Nn + 1 individual round keys, each of which is itself an array of Nc words. The number of rounds for the cipher (Nn) varies with the block length and the key length as shown in the following table.

Nn		Nc		
		4	6	8
Nk	4	10	12	14
	6	12	12	14
	8	14	14	14

Table 3 – The number of rounds as a function of block and key size

3.7 The SubBytes Transformation

The SubBytes transformation is a non-linear byte substitution that acts on every byte of the state in isolation to produce a new byte value using an S-box substitution table. The action of this transformation is illustrated in Figure 2 for a block size of 6.

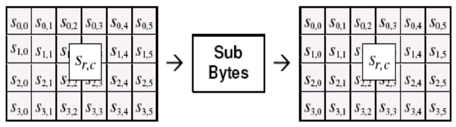


Figure 2 – SubBytes acts on every byte in the state in isolation

This substitution, which is invertible, is constructed by composing two transformations:

First the multiplicative inverse in the finite field described earlier (with element {00} mapped to itself).

Second the affine transformation over GF(2) defined by:

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

For 0 £ i < 8 where bi is bit i of the byte and ci is bit i of a byte c with the value {63} or {01100011}. Here and elsewhere a prime on a variable on the left of an equation indicates that its value is to be updated with the value on the right. In matrix form the latter component of the S-box transformation can be expressed as:

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

The final result of this two stage transformation is given in the following table.

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

Table 4 – The Substitution Table – Sbox[xy] (in hexadecimal)

3.8 The ShiftRows Transformation

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

$$s_{r,c}' = s_{r,[c+h(r,Nc)] \bmod Nc} \quad \text{for } 0 \le c < Nc \text{ and } 0 < r < 4$$

Where the shift amount h(r, Nc) depends on row number r and block length as follows:

h(r, Nc)		row (r)		
		1	2	3
Nc	4	1	2	3
	6	1	2	3
	8	1	3	4

Table 5 - Shift offsets for different rows and block lengths

This has the effect of moving bytes to lower positions in the row except that the lowest bytes wrap around into the top of the row (note that a prime on a variable indicates an updated value). The action of this transformation is illustrated in Figure 3 for a cipher block size of 6.

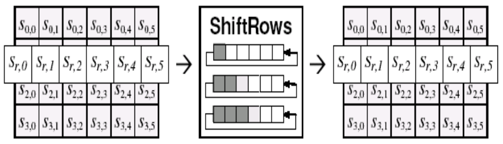


Figure 3 – ShiftRows acts independently on rows in the state

3.9 The MixColumns Transformation

The MixColumns transformation acts independently on every column of the state and treats each column as a four-term polynomial.

In matrix form the transformation used given in equation, where all the values are finite field elements as discussed in

$$\begin{bmatrix} s_{3,c}' \\ s_{2,c}' \\ s_{1,c}' \\ s_{0,c}' \end{bmatrix} = \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix} \begin{bmatrix} s_{3,c} \\ s_{2,c} \\ s_{1,c} \\ s_{0,c} \end{bmatrix} \quad \text{for } 0 \le c < Nc$$

The action of this transformation is illustrated in Figure 4 for a cipher block size of 6.

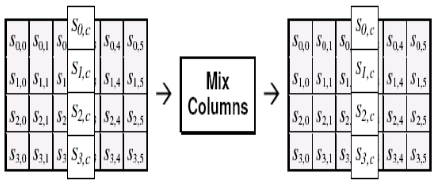


Figure 4 – MixColumns acts independently on each column in the state

3.10 The Xor RoundKey Transformation

In the Xor RoundKey transformation Nc words from the key schedule (the round key described later) are each added (XOR' d) into the columns of the state so that:

$$[b_{3c}', b_{2c}', b_{1c}', b_{0c}'] = [b_{3c}, b_{2c}, b_{1c}, b_{0c}] \oplus [k_{round,c}] \quad \text{for } 0 \le c < Nc$$

where the round key words K round, c (shortened to k r c in the diagram below) will be described later. The round number, round, is in the range 0 £ round £ Nn, with the value of 0 being used to denote the initial round key that is applied before the round function.

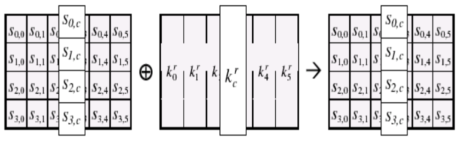


Figure 5 – Words from the key schedule are XOR'd into columns in the state

The action of this transformation is illustrated in Figure 5 for a cipher block size of 6. The byte address within each word of the key schedule is that described in above.

The Key Schedule

The round keys are derived from the cipher key by means of a key schedule with each round requiring Nc words of key data which, with an additional initial set, makes a total of Nc (Nn + 1) words, where Nn is the number of cipher rounds. This key schedule is considered either as a one dimensional array k of Nc (Nn + 1) 32-bit words with an index I in the range 0 £ i < Nc (Nn + 1) or as a two dimensional array k [n,c] of Nn + 1 round keys, each or which individually consists of a sub-array of Nc words.

The expansion of the input key into the key schedule proceeds according to the following pseudo code. The function SubWord(x) gives an output word for which the S-box substitution has been individually applied to each of the four bytes of its input x. The function RotWord(x) converts an input word [b3,b2,b1,b0] to an output [b0,b3,b2,b1] . The word array Rcon[i] contains the values [0, 0, 0, xi-1] with xi-1 being the powers of x in the field GF(256) discussed above (note that the index i starts at 1).

Note that this key schedule, which is illustrated in Figure 6 for Nk = 4 and Nc = 6, can be generated 'on-the fly' if necessary using a buffer of max(Nc, Nk) words. It can also be split into separate, somewhat simpler, key schedules for Nk _ 6 and Nk > 6 respectively.

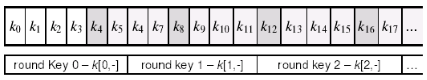

k_0 k_1 k_2 k_3 k_4 k_5 k_4 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15} k_{16} k_{17} ...
round Key 0 – k[0,-]
round key 1 – k[1,-]
round key 2 – k[2,-]
...

Figure 6 – The key schedule and round key selection for *Nk* = 4 and *Nc* = 6

4. DECRYPTION

4.1 The Inverse Cipher

The inversion of the cipher code presented above is straightforward.

4.2 The Inverse ShiftRows Transformation

The Inverse ShiftRows transformation operates individually on each of the last three rows of the state cyclically shifting the bytes in the row such that

$$s'_{r,[c+h(r,Nc)]\bmod Nc} = s_{r,c} \quad \text{for } 0 \le c < Nc \text{ and } 0 < r < 4$$

where the cyclic shift values h(r, Nc) are given in Table 6.

4.3 The Inverse SubBytes Transformation

The inverse S-box table needed for the inverse Inverse SubBytes transformation is given above.

Table6 gives the full inverse S-box, the inverse of the affine tranformation (3.1.1) being:

$$b'_i = b_{(i+2)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus d_i \text{ , where byte } d = \{05\} \qquad (5.2.1)$$

hex		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
x	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

Table 6 - The Inverse Substitution Table – InvSbox[xy] (in hexadecimal)

4.4 The Inverse Xor RoundKey Transformation

The XorRoundKey transformation is its own inverse.

4.5 The Inverse MixColumns Transformation

The InvMixColumns transformation acts independently on every column of the state and treats each column as a four-term polynomial as described above. In matrix form the transformation used is given in equation, where all the values are finite field elements as discussed above.

$$\begin{bmatrix} s'_{3c} \\ s'_{2c} \\ s'_{1c} \\ s'_{0c} \end{bmatrix} = \begin{bmatrix} 0e & 09 & 0d & 0b \\ 0b & 0e & 09 & 0d \\ 0d & 0b & 0e & 09 \\ 09 & 0d & 0b & 0e \end{bmatrix} \begin{bmatrix} s_{3c} \\ s_{1c} \\ s_{1c} \\ s_{0c} \end{bmatrix} \quad \text{for } 0 \le c < Nc$$

5. CONCLUSION

Cryptography has got it's own depth in computer science. Better algorithms are the future requirement for Cryptography. Like any technology, encryption software isn't perfect. Even the best products consume both processor speed and storage space. Bad encryption software can be confusing to use or easily compromised. The main aim of presenting this paper is to encrypt a java file using Rijndael Algorithm. The first aspect that has to be considered in our paper is file security and the need for file security. Key management is also related to Confidentiality. This deals with generating, distributing and storing keys File security must be implemented so as to eliminate the problems like unauthorized access, execution of commands illicitly, destructive behavior and confidentiality reaches. This can be implemented in future in various projects regarding computer security Like digital certificates, electronic signatures and hyper Encryption.

6. REFERENCES

[1] James, N., E. Barker, L. Bassham, W. Burr, M.Dworkin, J. Foti and E. Roback, 2000. Report on the development of the advanced encryption standard (AES). Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce.

[2] Srdjan, C., L. Buttyan and J.-P. Hubaux, 2003. Self-organized public-key management for mobile ad hoc Networks. IEEE Trans. Mobile Computing: pp: 52-64.

[3] Philip, R., M. Bellare and J. Black OCB, 2003. A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Information System and Security, pp: 365-403.

[4] Mary, R.T., A. Essiari and S. Mudumbai, 2003. Certificate-based authorization policy in a PKI environment. ACM Trans. Information System and Security, pp:566-88.

[5] D. Alessandri, C. Cachin, M. Dacier, et al, "Towards a Taxonomy of Tntrusion Detection Systems and Attacks. MAFTIA deliverable D3". 2001

[6] Custom Attack Simulation Language, Secure Networks. 1998.

[7] L.Me.Gassata, "A genetic algorithm as an alternative tools for security audit trails analysis", *RAID'98*, 1998.

[8] S.Kumar, *Classification and Detection of Computer Instructions. PhD thesis,* Purdue University, 1995.