Modern Block Cipher Standards (AES)

Debdeep Mukhopadhyay

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
INDIA -721302

Objectives

- **Introduction to arithmetic of AES**
- **AES Algorithm**
  - **Sub Byte**
  - **Shift row**
  - **Mix Column**
  - **Add round Key**

Polynomials representation

An element of AES state matrix is of the form:

$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + ... + b_0$

x being called indeterminate of the polynomial,

and the $b_i \in \{0,1\}$ the coefficients.

The degree of a polynomial equals l if $b_j = 0, \ \forall j>l$,

and l is the smallest number with this property.

Operations on Polynomials

- **Addition:**

$$c(x) = a(x) + b(x) \Leftrightarrow c_i = a_i + b_i, 0 \leq i \leq n$$

Addition is closed

0 (polynomial with all coefficients 0) is the identity element.

The inverse of an element can be found by replacing each coefficient of the polynomial by its inverse.

In this case, it is same as b_i.

Example

Compute the sum of the polynomials denoted by 57 and 83.

In binary, 57=01010111, and 83=10000011.
In polynomial notations we have,

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

The addition can be implemented with the bitwise XOR instruction.

Multiplication

- **Associative**
- **Commutative**
- **Distributive wrt. addition of polynomials.**

In order to make the multiplication closed over the polynomials.
We select a polynomial m(x) of degree *l*, called the reduction polynomial.
The multiplication is then defined as follows:

$$c(x) = a(x).b(x) \Leftrightarrow c(x) \equiv a(x) \times b(x) \pmod{m(x)}$$

Irreducible Polynomial

- **A polynomial d(x) is irreducible over the field GF(p) iff there exist no two polynomials a(x) and b(x) with coefficients in GF(p) such that d(x)=a(x)b(x), where a(x) and b(x) are of degree > 0.**

The set of polynomials of degree 7 is called GF(2^8).

Example

Degree	Irreducible Polynomial
1	$(x+1), x$
2	(x^2+x+1)
3	(x^3+x^2+1), (x^3+x+1)
4	$(x^4+x^3+x^2+x+1)$, (x^4+x^3+1), (x^4+x+1)

Example of Multiplication

Compute the product of the elements 57 and 83 in GF(2^8)
57=01010111, and 83=10000011.
In polynomial notations we have,

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

and,

$$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1)$$
$$\equiv x^7 + x^6 + 1 \pmod{x^8 + x^4 + x^3 + x + 1}$$

Addition and Multiplication in GF(2^n)

- Addition can be implemented using only XOR operation.
- Multiplication can be implemented using shift-left and XOR operation.

Introduction to AES

- In 1999, NIST issued a new standard that said 3DES should be used
  - 168-bit key length
  - Algorithm is the same as DES
- 3DES had drawbacks
  - Algorithm is sluggish in software
  - Only uses 64-bit block size

Introduction to AES (Cont.)

- In 1997, NIST issued a CFP for AES
  - security strength >= 3DES
  - improved efficiency
  - must be a symmetric block cipher (128-bit)
  - key lengths of 128, 192, and 256 bits

Introduction of AES (cont.)

- First round of evaluation
  - 15 proposed algorithms accepted
- Second round
  - 5 proposed algorithms accepted
    - Rijndael, Serpent, 2fish, RC6, and MARS
- Final Standard - November 2001
  - Rijndael selected as AES algorithm

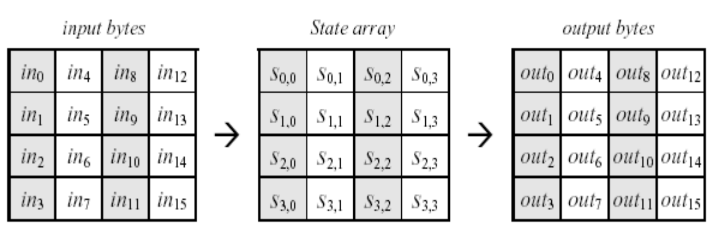
Rijndael Algorithm

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Difference between Rijndael and AES

- **Rijndael is a block cipher with both a variable block length and a variable key length.**
- **The block and key lengths can be independently fixed to any multiple of 32, ranging from 128 to 256 bits.**
- **The AES fixes the block length to 128 bits, and supports key lengths of 128, 192 and 256 bits.**

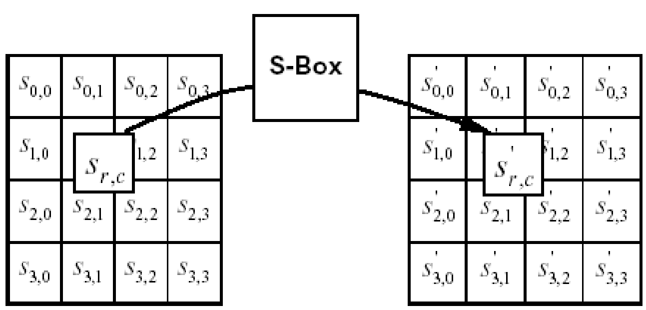
Rijndael Algorithm



Rijndael Algorithm

- In Rijndael, there are four round functions.
  (1) Byte Sub
  (2) Shift Row
  (3) Mix Columns
  (4) Add Round Key

Byte Sub



The AES SBox

- **Based on the mapping defined by K. Nyberg, published in Eurocrypt 1993.**
- **The input is an eight bit value, a. Here, a is in GF(2^8).**
- **The SBox is based on the mapping:**

The AES SBox

- **In addition no fixed points or opposite fixed points were desired.**
- **Hence an affine mapping was defined.**

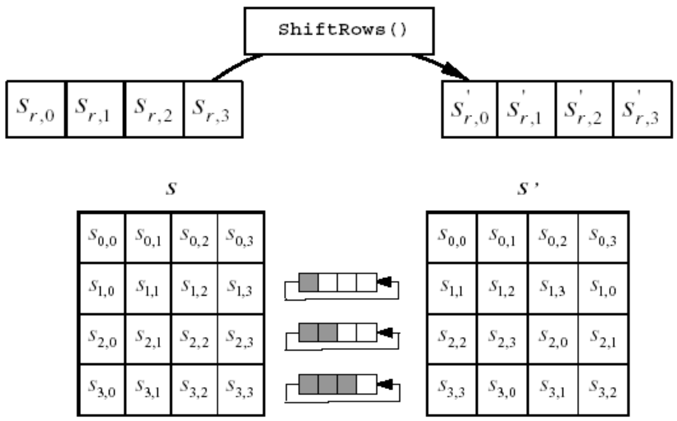
The AES S-Box Affine mapping

S-Box

For Examples, if $S_{1,1} = \{53\}$, then the substitution value would be determined by the intersection of row with index '5' and the column with index '3' in below figure.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	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
x	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

Shift Row



Mix Columns

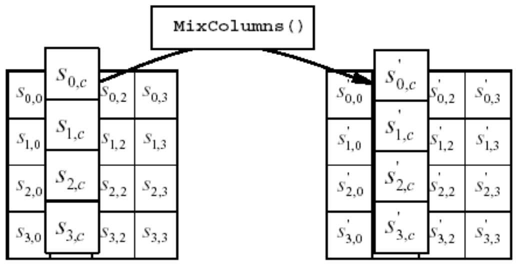
- **Mix Columns:**

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

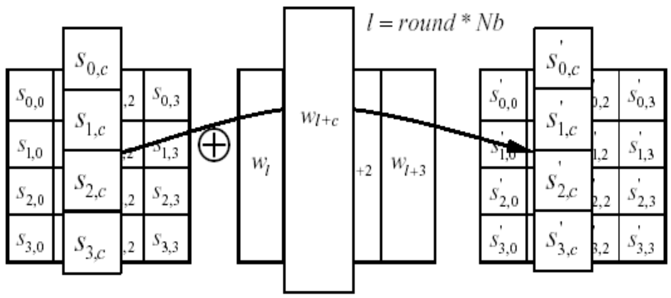
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$



Add Round Key



Modern Block Cipher Standards (AES) (contd.)

Debdeep Mukhopadhyay

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
INDIA -721302

Objectives

- The AES Key scheduling
- The AES Decryption function
- Implementation of the AES Round on modern processors

The AES KeyScheduling

- **Efficiency:**
  - Low working memory
  - Performance on a wide range of processors
- **Symmetry elimination:** use round constants to eliminate symmetricity
- **Diffusion:** High diffusion of cipher key differences into the expanded key
- **Non-linearity:** Exhibit high non-linearity to prevent the determination of differences in the expanded key from that of the input key.

Key Expansion

- The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule.
- The Key Expansion generates a total of ***Nb (Nr*** + 1) words: the algorithm requires an initial set of ***Nb*** words, and each of the ***Nr*** rounds requires ***Nb*** words of key data.
- Key Expansion includes the following functions :
  (1)**RotWord** : Takes a word [a_0,a_1,a_2,a_3] as input , performs a cyclic permutation, and returns the word [a_1,a_2,a_3,a_0]
  (2)**SubWord** : is a function that take a 4-bytes input word and applies the S-box to each of the four bytes to produce and output word.
  (3)**Rcon[i/NK**] : contains the values given by [x^{i-1}, {00},{00},{00}], with x^{i-1} being powers of x (x is denoted as {02}) in the field GF(2^8).

The Key Scheduling Algorithm for Nk=4

```
keyexpansion(byte key[4*Nk],word w[(Nr+1)*Nb],Nk)
word temp; i=0;
while (i<Nk)
  { w[i]={key[4i],key[4i+1]key[4i+2]key[4i+3]};
i=i+1;
}
```

The Key Scheduling Algorithm for Nk=4

```
while(i<Nb(Nr+1)){
   temp=w[i-1];
   if(i mod Nk = 0)
     temp= Subword(Rotword(temp)) xor Rcon[i/Nk];

    w[i]=temp xor w[i-Nk];
    i=i+1;
  }
```

The Round Constant

- Each round constant is a 4 byte value, where the right most three bytes are always 0.
- The left byte is equal to x^{i-1}, where x is an element in $GF(2^8)$
- The Round Constants can be either obtained from a table or computed by multiplication in $GF(2^8)$, where $m(x)=x^8+x^4+x^3+x+1$ is the reduction polynomial.

Powers of x in GF(2^8)

- $RC_1 = x^{1-1} = x^0$ = 0000 0001 = 01_{16}
- $RC_2 = x^{2-1} = x$ = 0000 0010 = 02_{16}
- $RC_3 = x^{3-1} = x^2$ = 0000 0100 = 04_{16}
- $RC_4 = x^{4-1} = x^3$ = 0000 1000 = 08_{16}
- $RC_5 = x^{5-1} = x^4$ = 0001 0000 = 10_{16}
- $RC_6 = x^{6-1} = x^5$ = 0010 0000 = 20_{16}
- $RC_7 = x^{7-1} = x^6$ = 0100 0000 = 40_{16}
- $RC_8 = x^{8-1} = x^7$ = 1000 0000 = 80_{16}
- $RC_9 = x^{9-1} = x^8$ = 0001 1011 = $1B_{16}$
- $RC_{10} = x^{10-1} = x^9$ = 0011 0110 = 36_{16}

Algorithm of Encryption process

```
Cipher (byte  in[4*Nb],byte out[4*Nb],word w[Nb*(Nr+1)]
begin
    byte state [4,Nb];
    state = in;
   AddRoundKey(state, w[0,Nb-1];
   for(round=1 to Nr-1)
   begin
      SubBytes(state);
      ShiftRow(state);
      MixColumn(state);
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1];
   end
```

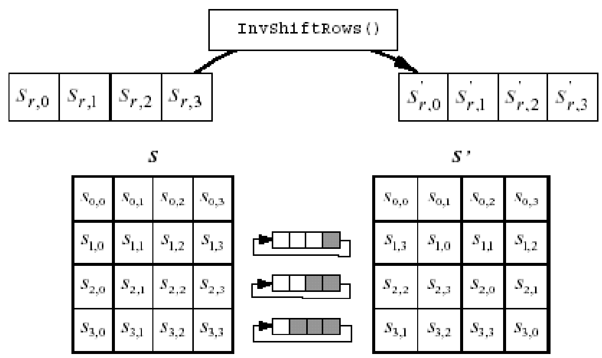
Last Round of AES encryption

```
SubBytes(state);
ShiftRow(state);

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1];

out=state;
```

Inverse Cipher (decryption)

- The cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES Algorithm. The individual of transformation used in the Inverse Cipher process the state.
  - InvshiftRows( )
  - InvSubBytes( )
  - InvMixColumn( )
  - AddRoundKey( )

InvShiftRows( )



Inverse S-Box

x \ 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

InvMixColumns

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Algorithm of Decryption process

```
InvCipher (byte  in[4*Nb],byte out[4*Nb],word w[Nb*(Nr+1)]
begin
    byte state [4,Nb];
    state = in;
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1];

   for(round= Nr-1 to 1)
   begin
      InvShiftRow(state);
      InvSubBytes(state);
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1];
     InvMixColumn(state);
   end
```

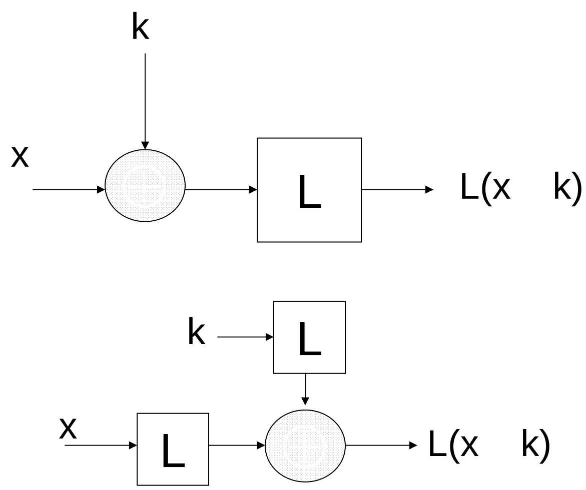
Last Round of AES decryption

```
InvShiftRow(state);
InvSubBytes(state);

AddRoundKey(state, w[0, Nb-1];

out=state;
```

Some Points

- The order of InvShift Rows and InvSubBytes is indifferent.
- The order of AddRoundKey and InvMixColumns can be inverted if the round key is adapted accordingly.

A Linear transformation can be pushed through an XOR



Encryption steps for two round AES variant

- AddRoundKey(State, ExpandedKey[0]);
- SubBytes(State);
- ShiftRow(State);
- MixColumn(State);
- AddRoundKey(State, ExpandedKey[1]);
- SubBytes(State);
- ShiftRow(State);
- AddRoundKey(State, ExpandedKey[2]);

Decryption steps for two round AES variant

- AddRoundKey(State, ExpandedKey[2]);
- InvShiftRow(State);
- InvSubBytes(State);
- AddRoundKey(State, ExpandedKey[1]);
- InvMixColumn(State);
- InvShiftRow(State);
- InvSubBytes(State);
- AddRoundKey(State, ExpandedKey[0]);

Equivalent Decryption steps for two round AES variant

- AddRoundKey(State, ExpandedKey[2]);
- InvSubBytes(State);
- InvShiftRow(State);
- InvMixColumn(State);
- AddRoundKey(State, EqExpandedKey[1]);
- InvSubBytes(State);
- InvShiftRow(State);
- AddRoundKey(State, ExpandedKey[0]);

Equivalent Decryption

- The equivalent key-scheduling can be obtained by applying InvMixColumns after the key-scheduling algorithm.
- This can be generalized to the full round AES.
- Thus we see that in the equivalent decryption the sequence of steps is similar.
  - This helps implementation

Implementation on modern processors

- Different steps of the round transformation can be combined in a single set of look up tables.
- This allows very fast implementation on processors with word length 32 or greater.

AES on the table!

Let the input of the round transformation be denoted by a, and the output of SubBytes by b.

$\therefore b_{i,j} = S_{RD}[a_{i,j}], 0 \le i < 4$ and $0 \le j < N_b$

Let the output of ShiftRows be denoted by c, and the output of MixColumns by d.

$$\therefore \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j+C_0} \\ b_{1,j+C_1} \\ b_{2,j+C_2} \\ b_{3,j+C_3} \end{bmatrix}, 0 \le \text{j} < \text{N}_b$$

$$\text{and, } \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 02 & 01 & 01 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}, 0 \le \text{j} < \text{N}_b$$

The above addition in the indices are done modulo N_b.

AES on the table!

Combining the above equations we have,

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 02 & 01 & 01 \end{bmatrix} \begin{bmatrix} S_{RD}[a_{0,j+C_0}] \\ S_{RD}[a_{1,j+C_1}] \\ S_{RD}[a_{2,j+C_2}] \\ S_{RD}[a_{3,j+C_3}] \end{bmatrix}, 0 \le \text{j} < \text{N}_b$$

$$\Rightarrow \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} S_{RD}[a_{0,j+C_0}] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} S_{RD}[a_{1,j+C_0}]$$

$$\oplus \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} S_{RD}[a_{2,j+C_0}] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} S_{RD}[a_{3,j+C_0}], 0 \le \text{j} < \text{N}_b$$

AES on the table!

Define 4 tables, T_0, T_1, T_2 and T_3.

$$T_0[a] = \begin{bmatrix} 02S_{RD}[a] \\ 01S_{RD}[a] \\ 01S_{RD}[a] \\ 03S_{RD}[a] \end{bmatrix}, T_1[a] = \begin{bmatrix} 03S_{RD}[a] \\ 02S_{RD}[a] \\ 01S_{RD}[a] \\ 01S_{RD}[a] \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} 01S_{RD}[a] \\ 03S_{RD}[a] \\ 02S_{RD}[a] \\ 01S_{RD}[a] \end{bmatrix}, T_3[a] = \begin{bmatrix} 01S_{RD}[a] \\ 01S_{RD}[a] \\ 03S_{RD}[a] \\ 02S_{RD}[a] \end{bmatrix}$$

Cost of the table(s)

- Each table has 256 entries of size 4 bytes. Thus each table is of 1 kB.
- Since AddRoundKey can be implemented by additional 32 bit XOR, AES round can be implemented with 4 kB of tables, with 4 table look ups and one XOR per column per round.
- Note that final round does not have a Mixcolumn step.
- Using some additional simple operations, the 4 tables can be reduced to 1. (How?)

Further Reading

- Douglas Stinson, *Cryptography Theory and Practice, 2nd Edition*, Chapman & Hall/CRC
- Joan Daemen, Vincent Rijmen, *"The Design of Rijndael",* Springer Verlag

Exercise

- Convince yourself that diffusion takes place very fast in AES.
  – How many rounds are necessary for a one byte diffusion to spread to the entire AES state matrix?

Next days topic

- Stream Ciphers

Number of rounds of AES-128

- Two rounds provide full diffusion
- Short cut attacks exist on 6 rounds of AES-128.
- As a conservative approach, two rounds of diffusion are provided at the beginning and two at the end, thus explaining the 10 rounds.

Number of rounds

- Number of rounds increased by 1 for every 32 bits additional key bits.
- The main reason is we need to avoid short cut attacks. Since with the increase in key size, the exhaustive key search grows exponentially, the short cut attacks will work for larger number of rounds than for AES-128.

Attacks on reduced variants

- Linear Cryptanalysis
- Differential Cryptanalysis
- Related key attacks
- Boomerang attacks
- Square attacks

When Nk>6…

```
while(i<Nb(Nr+1)){
   temp=w[i-1];
   if(i mod Nk = 0)
     temp= Subword(Rotword(temp)) xor Rcon[i/Nk];
   if(i mod Nk=4)
    temp = Subword(temp);

    w[i]=temp xor w[i-Nk];
    i=i+1;
  }
```

Key Expansion

Expansion of a 128-bit Cipher Key:

This section contains the key expansion of the following cipher key:

Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

for ***Nk*** = 4, which results in

w0 = **2b7e1516** w1 = **28aed2a6** w2 = **abf71588** w3 = **09cf4f3c**

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943

Key Expansion (192-bit Cipher Key)

This section contains the key expansion of the following cipher key:

Cipher Key = 8e 73 b0 f7 da 0e 64 52 c8 10 f3 2b
80 90 79 e5 62 f8 ea d2 52 2c 6b 7b

for $Nk = 6$, which results in

w_0 = 8e73b0f7 w_1 = da0e6452 w_2 = c810f32b w_3 = 809079e5

w_4 = 62f8ead2 w_5 = 522c6b7b

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i] = temp XOR w[i-Nk]
6	522c6b7b	2c6b7b52	717f2100	01000000	707f2100	8e73b0f7	fe0c91f7
7	fe0c91f7					da0e6452	2402f5a5
8	2402f5a5					c810f32b	ec12068e
9	ec12068e					809079e5	6c827f6b
10	6c827f6b					62f8ead2	0e7a95b9
11	0e7a95b9					522c6b7b	5c56fec2
12	5c56fec2	56fec25c	b1bb254a	02000000	b3bb254a	fe0c91f7	4db7b4bd

Key Expansion(256-bit Cipher Key)

This section contains the key expansion of the following cipher key:

Cipher Key = 60 3d eb 10 15 ca 71 be 2b 73 ae f0 85 7d 77 81
1f 35 2c 07 3b 61 08 d7 2d 98 10 a3 09 14 df f4

for $Nk = 8$, which results in

w_0 = 603deb10 w_1 = 15ca71be w_2 = 2b73aef0 w_3 = 857d7781

w_4 = 1f352c07 w_5 = 3b6108d7 w_6 = 2d9810a3 w_7 = 0914dff4

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i] = temp XOR w[i-Nk]
8	0914dff4	14dff409	fa9ebf01	01000000	fb9ebf01	603deb10	9ba35411
9	9ba35411					15ca71be	8e6925af
10	8e6925af					2b73aef0	a51a8b5f
11	a51a8b5f					857d7781	2067fcde
12	2067fcde		b785b01d			1f352c07	a8b09c1a
13	a8b09c1a					3b6108d7	93d194cd
14	93d194cd					2d9810a3	be49846e
15	be49846e					0914dff4	b75d5b9a
16	b75d5b9a	5d5b9ab7	4c39b8a9	02000000	4e39b8a9	9ba35411	d59aecb8