Bahan Kuliah ke-13

IF5054 Kriptografi

Advanced Encryption Standard (*AES*)

Disusun oleh:

Ir. Rinaldi Munir, M.T.

Departemen Teknik Informatika
Institut Teknologi Bandung
2004

13. *Advanced Encryption Standard* (*AES*)

13.1 Sejarah *AES*

- DES (*Data Encryption Standard*) mungkin akan berakhir masa penggunaannya sebagai standard enkripsi kriptografi simetri. DES dianggap sudah tidak aman lagi karena dengan perangkat keras khusus kuncinya bisa ditemukan dalam beberapa hari (baca materi kuliah DES).

- *National Institute of Standards and Technology* (*NIST*), sebagai agensi Departemen Perdagangan AS mengusulkan kepada Pemerintah Federal AS untuk sebuah standard kriptografi kriptografi yang baru.

- Untuk menghindari kontoversi mengenai standard yang baru tersebut, sebagaimana pada pembuatan *DES* (*NSA* sering dicurigai mempunyai "pintu belakang" untuk mengungkap cipherteks yang dihasilkan oleh DES tanpa mengetahui kunci), maka *NIST* mengadakan sayembara terbuka untuk membuat standard algoritma kriptografi yang baru sebagai pengganti *DES.* Standard tersebut kelak diberi nama *Advanced Encryption Standard* (*AES*).

- Persyaratan yang diajukan oleh *NIST* tentang algoritma yang baru tersebut adalah:
  1. Algoritma yang ditawarkan termasuk ke dalam kelompok algoritma kriptografi simetri berbasis *cipher* blok.
  2. Seluruh rancangan algoritma harus publik (tidak dirahasiakan)
  3. Panjang kunci fleksibel: 128, 192, dan 256 bit.
  4. Ukuran blok yang dienkripsi adalah 128 bit.
  5. Algoritma dapat diimplementasikan baik sebagai *software* maupun *hardware*.

- NIST menerima 15 proposal algoritma yang masuk. Konferensi umum pun diselenggarakan untuk menilai keamanan algoritma yang diusulkan.

- Pada bulan Agustus 1998, *NIST* memilih 5 finalis yang didasarkan pada aspek keamanan algoritma, kemangkusan (*efficiency*), fleksibilitas, dan kebutughan memori (penting untuk *embedded system*). Finalis tersebut adalah:
  1. *Rijndael* (dari Vincent **Rij**men da**n** Joan **Dae**men – Be**l**gia, 86 suara)
  2. *Serpent* (dari Ross Anderson, Eli Biham, dan Lars Knudsen – Inggris, Israel, dan Norwegia, 59 suara).
  3. *Twofish* (dari tim yang diketuai oleh Bruce Schneier – USA, 31 suara)
  4. *RC6* (dari Laboratorium *RSA* – USA, 23 suara)
  5. *MARS* (dari IBM, 13 suara)

- Pada bulan Oktober 2000, *NIST* mengumumkan untuk memilih Rijndael (dibaca: Rhine-doll), dan pada bulan November 2001, Rijndael ditetapkan sebagai AES, dan diharapkan Rijndael menjadi standard kriptografi yang dominan paling sedikit selama 10 tahun.

13.2 Panjang Kunci dan Ukuran Blok *Rijndael*

- Rijndael mendukung panjang kunci 128 bit sampai 256 bit dengan step 32 bit. Panjang kunci dan ukuran blok dapat dipilih secara independen.

- Setiap blok dienkripsi dalam sejumlah putaran tertentu, sebagaimana halnya pada *DES.*

- Karena *AES* menetapkan panjang kunci adalah 128, 192, dan 256, maka dikenal *AES*-128, *AES*-192, dan *AES*-256.

	Panjang Kunci (*Nk words*)	Ukuran Blok (*Nb words*)	Jumlah Putaran (*Nr*)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Catatan: 1 *word* = 32 bit

- Secara de-fakto, hanya ada dua varian *AES*, yaitu *AES*-128 dan *AES*-256, karena akan sangat jarang pengguna menggunakan kunci yang panjangnya 192 bit.

- Karena *AES* mempunyai panjang kunci paling sedikit 128 bit, maka *AES* tahan terhadap serangan *exhaustive key search* dengan teknologi saat ini. Dengan panjang kunci 128-bit, maka terdapat sebanyak

  $$2^{128} = 3{,}4 \times 10^{38}$$

  kemungkinan kunci.

  Jika digunakan komputer tercepat yang dapat mencoba 1 juta kunci setiap detik, maka akan dibutuhkan waktu $5{,}4 \times 10^{24}$ tahun untuk mencoba seluruh kemungkinan kunci.

  Jika digunakan komputer tercepat yang dapat mencoba 1 juta kunci setiap milidetik, maka akan dibutuhkan waktu $5{,}4 \times 10^{18}$ tahun untuk mencoba seluruh kemungkinan kunci.

13.3 Algoritma *Rijndael*

- Seperti pada *DES*, *Rijndael* menggunakan substitusi dan permutasi, dan sejumlah putaran (*cipher* berulang) – setiap putaran mengunakan kunci internal yang berbeda (kunci setiap putaran disebut *round key*). Tetapi tidak seperti *DES* yang berorientasi bit, *Rijndael* beroperasi dalam orientasi *byte* (untuk memangkuskan implementasi algoritma ke dalam *software* dan *hardware*).

- Garis besar Algoritma *Rijndael* yang beroperasi pada blok 128-bit dengan kunci 128-bit adalah sebagai berikut (di luar proses pembangkitan *round key*):

  1. *AddRoundKey*: melakukan *XOR* antara *state* awal (plainteks) dengan *cipher key*. Tahap ini disebut juga *initial round.*

  2. Putaran sebanyak *Nr* – 1 kali. Proses yang dilakukan pada setiap putaran adalah:

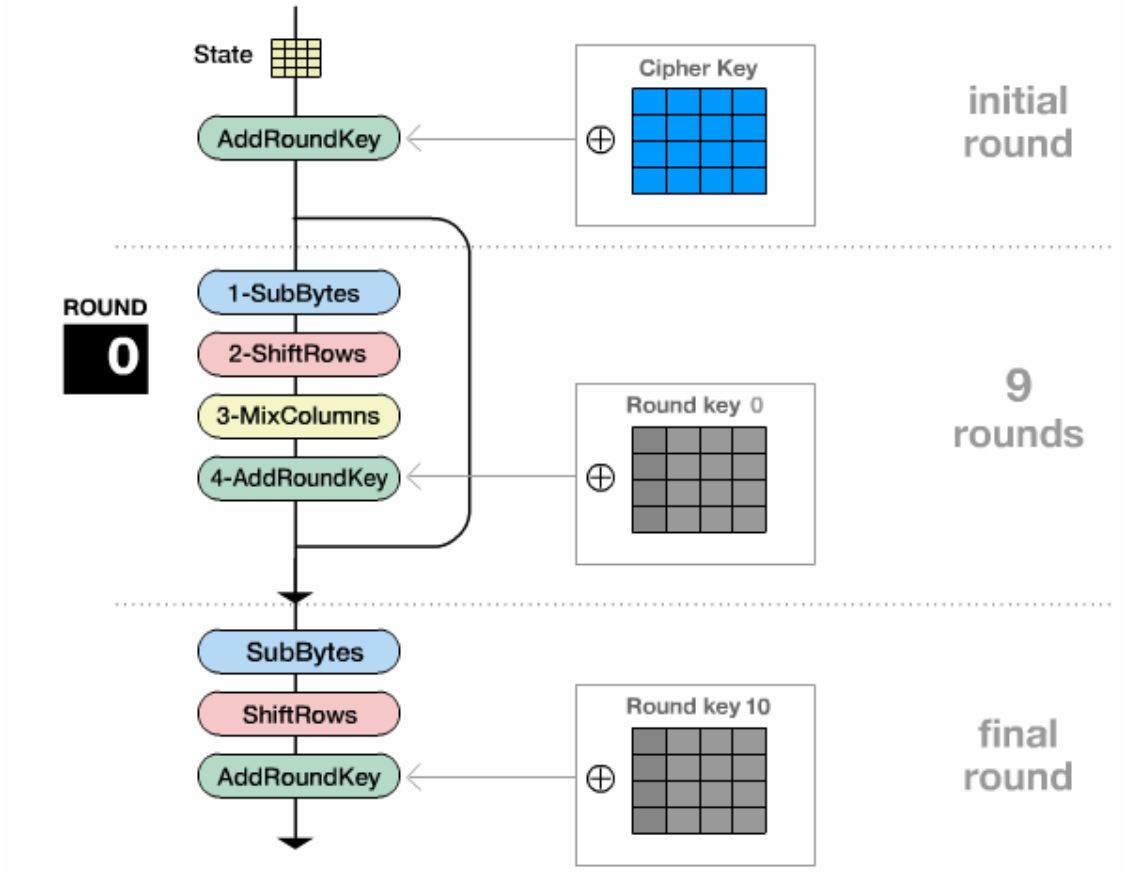
     a. *SubBytes*: substitusi *byte* dengan menggunakan tabel substitusi (*S-box*).

     b. *ShiftRows*: pergeseran baris-baris *array state* secara *wrapping*.

     c. *MixColumns*: mengacak data di masing-masing kolom *array state.*

     d. *AddRoundKey*: melakukan XOR antara *state* sekarang *round key*.

  3. *Final round*: proses untuk putaran terakhir:
     a. SubBytes
     b. ShiftRows
     c. AddRoundKey



Gambar 13.1 Diagram proses enkripsi

Versi 1: (putaran terakhir diperlakukan khusus)

```
#define LENGTH 16           /* Jumlah byte di dalam blok atau kunci */
#define NROWS 4             /* Jumlah baris di dalam state */
#define NCOLS 4             /* Jumlah kolom di dalam state */
#define ROUNDS 10           /* Jumlah putaran */
typedef unsigned char byte;  /* unsigned 8-bit integer */

rijndael (byte plaintext[LENGTH], byte ciphertext[LENGTH],
          byte key[LENGTH])
{
 int r;                     /* pencacah pengulangan */
 byte state[NROWS][NCOLS];  /* state sekarang */
 struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];  /* kunci pada
                                                   setiap putaran */

 KeyExpansion(key, rk);       /* bangkitkan kunci setiap putaran */
 CopyPlaintextToState(state, plaintext);  /* inisialisasi
                                             state sekarang */
 AddRoundKey(state, rk[0]);  /* XOR key ke dalam state */

 for (r = 1; r<= ROUNDS - 1; r++)
 {
  SubBytes(state);        /* substitusi setiap byte dengan S-box */
  ShiftRows(state);       /* rotasikan baris i sejauh i byte */
  MixColumns(state);   /* acak masing-masing kolom */
  AddRoundKey(state, rk[r]);   /* XOR key ke dalam state */
 }
 SubBytes(state);        /* substitusi setiap byte dengan S-box */
 ShiftRows(state);       /* rotasikan baris i sejauh i byte */
 AddRoundKey(state, rk[ROUNDS]);   /* XOR key ke dalam state */

 CopyStateToCiphertext(ciphertext, state);  /* blok cipherteks yang
                                               dihasilkan */

}
```

Versi 2: (proses pada setiap putaran sama)

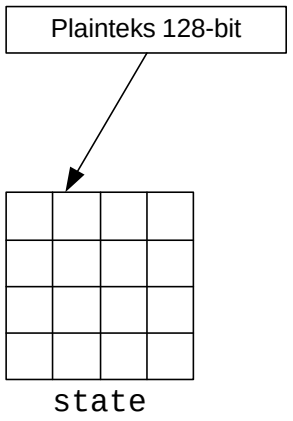
```
#define LENGTH 16            /* Jumlah byte di dalam blok atau kunci */
#define NROWS 4              /* Jumlah baris di dalam state */
#define NCOLS 4              /* Jumlah kolom di dalam state */
#define ROUNDS 10            /* Jumlah putaran */
typedef unsigned char byte;  /* unsigned 8-bit integer */

rijndael (byte plaintext[LENGTH], byte ciphertext[LENGTH],
          byte key[LENGTH])
{
 int r;                     /* pencacah pengulangan */
 byte state[NROWS][NCOLS];  /* state sekarang */
 struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];  /* kunci pada
                                                   setiap putaran */

 KeyExpansion(key, rk);       /* bangkitkan kunci setiap putaran */
 CopyPlaintextToState(state, plaintext);  /* inisialisasi
                                             state sekarang */
 AddRoundKey(state, rk[0]);  /* XOR key ke dalam state */

 for (r = 1; r<= ROUNDS; r++)
 {
  SubBytes(state);        /* substitusi setiap byte dengan S-box */
  ShiftRows(state);       /* rotasikan baris i sejauh i byte */
  if (r < ROUNDS) MixColumns(state);   /* acak masing-masing kolom */
  AddRoundKey(state, rk[r]);   /* XOR key ke dalam state */
 }
 CopyStateToCiphertext(ciphertext, state);  /* blok cipherteks yang
                                               dihasilkan */

}
```

- Algoritma Rijndael mempunyai 3 parameter:
  1. `plaintext` : *array* yang berukuran 16-*byte*, yang berisi data masukan.
  2. `ciphertext` : *array* yang berukuran 16-*byte*, yang berisi hasil enkripsi.
  3. `key` : *array* yang berukuran 16-*byte*, yang berisi kunci ciphering (disebut juga *cipher key*).

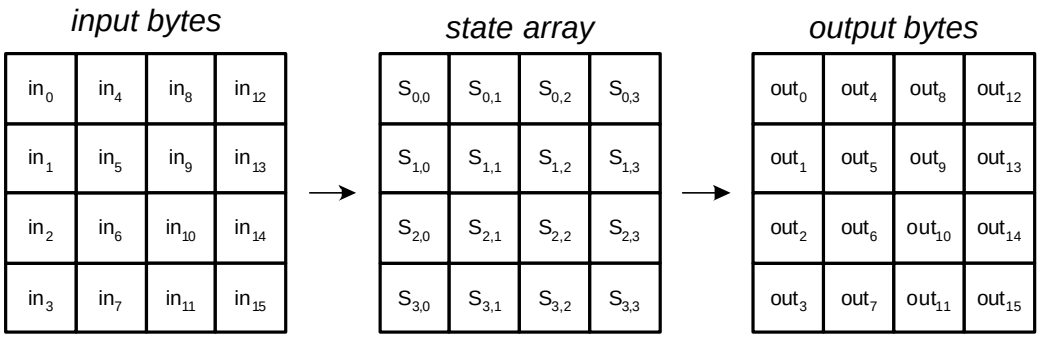
  Dengan 16 *byte*, maka baik blok data dan kunci yang berukuran 128-bit dapat disimpan di dalam ketiga *array* tersebut ($128 = 16 \times 8$).

- Selama kalkulasi plainteks menjadi cipherteks, status sekarang dari data disimpan di dalam *array of bytes* dua dimensi, `state`, yang berukuran `NROWS` × `NCOLS`. Untuk blok data 128-bit, ukuran `state` adalah 4 × 4.

  Elemen *array* `state` diacu sebagai *S*[*r*,*c*], dengan $0 \le r < 4$ dan $0 \le c < Nb$ (*Nb* adalah panjang blok dibagi 32. Pada *AES*-128, Nb = 128/32 = 4).



- Pada awal enkripsi, 16-*byte* data masukan, in_0, in_1, …, in_{15} disalin ke dalam *array* `state` (direalisasikan oleh fungsi `CopyPlaintextToState(state, plaintext)`) seperti diilustrasikan sebagai berikut:



Operasi enkripsi/dekripsi dilakukan terhadap *array S*, dan keluarannya ditampung didalam array *out.*

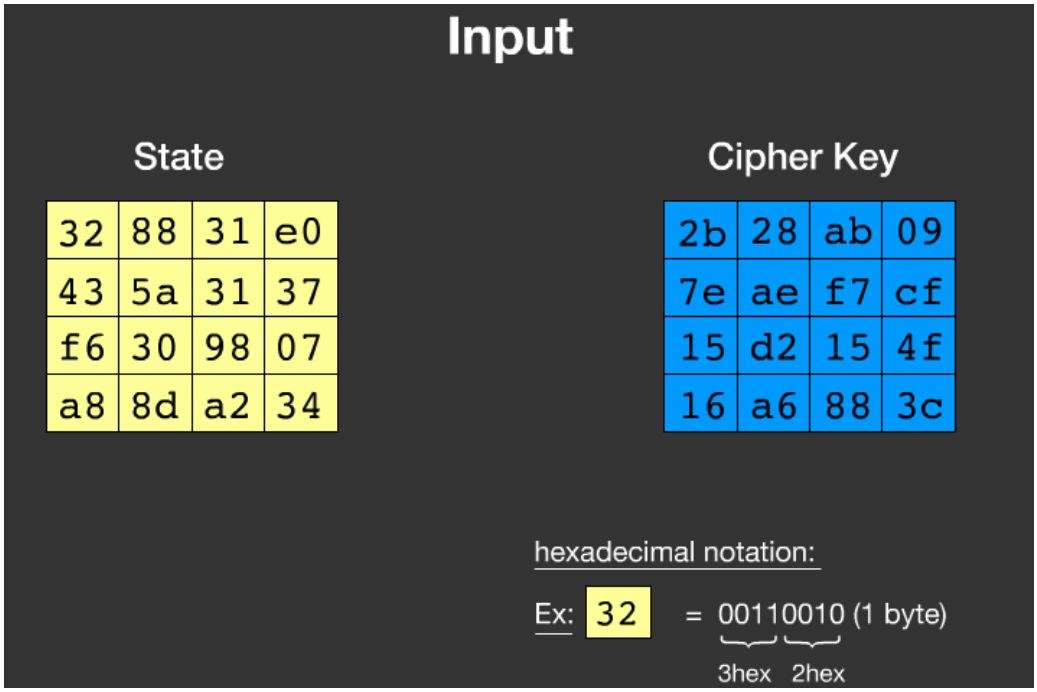
Skema penyalinan array masukan *in* ke *array S*:

$$S[r, c] \leftarrow in[r + 4c] \quad \text{untuk } 0 \le r < 4 \text{ dan } 0 \le c < Nb$$

Skema penyalinan *array S* ke *array* keluaran *out*:

$$out[r+4c] \leftarrow S[r, c] \quad \text{untuk } 0 \le r < 4 \text{ dan } 0 \le c < Nb$$

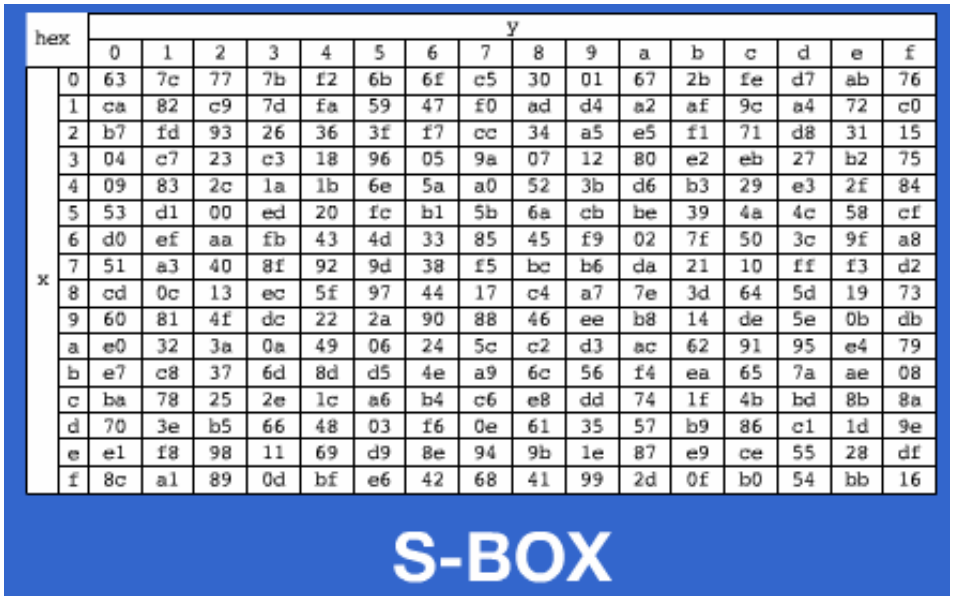
Contoh: (elemen state dan kunci dalam notasi HEX)



13.3.1 Transformasi *SubBytes()*

- Transformasi *SubBytes()* memetakan setiap byte dari *array state* dengan menggunakan tabel substiusi *S-box*. Tidak seperti *DES* yang mempunyai *S-box* berbeda pada setiap putaran, *AES* hanya mempunyai satu buah *S-box*.

  Tabel *S-box* yang digunakan adalah:



hex	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
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

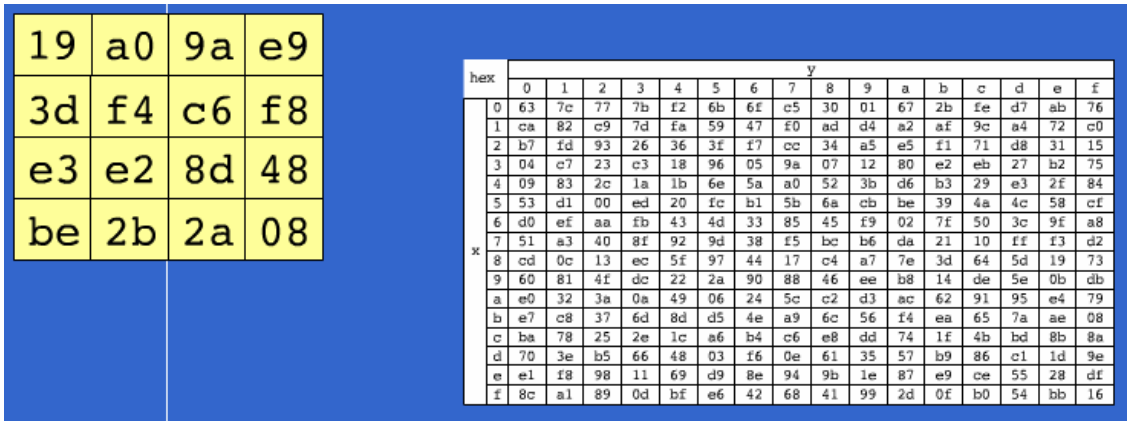
x

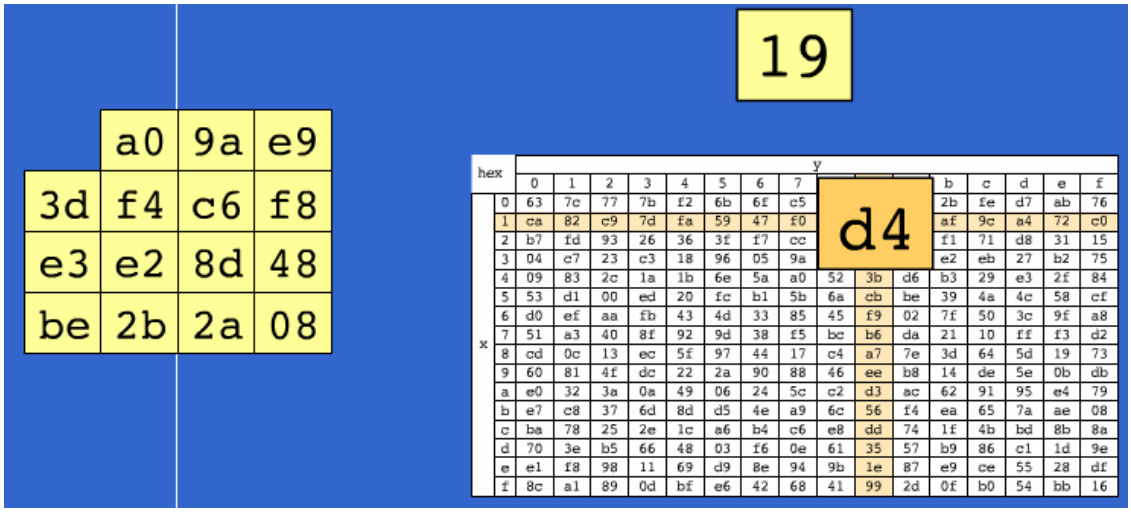
S-BOX

- Cara pensubstitusian adalah sebagai berikut: untuk setiap byte pada *array state*, misalkan *S*[*r*, *c*] = *xy*, yang dalam hal ini *xy* adalah digit heksadesimal dari nilai *S*[*r*, *c*], maka nilai substitusinya, dinyatakan dengan *S*'[*r*, *c*], adalah elemen di dalam *S-box* yang merupakan perpotongan baris *x* dengan kolom *y*.

  Misalnya *S*[0, 0] = 19, maka S'[0, 0] = d4

Contoh:



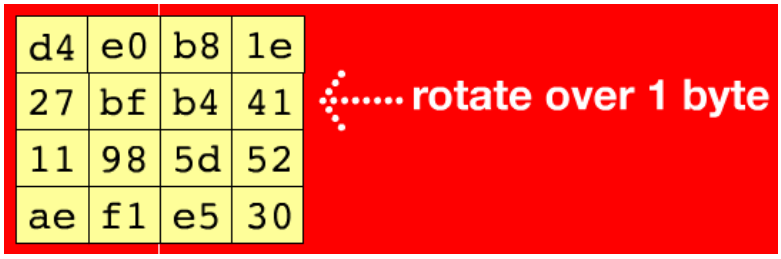


13.3.2 Transformasi *ShiftRows()*

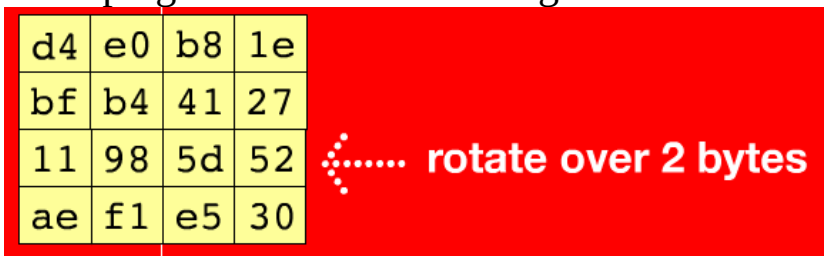
- Transformasi *ShiftRows()* melakukan pergeseran secara *wrapping* (siklik) pada 3 baris terakhir dari array state. Jumlah pergeseran bergantung pada nilai baris (r). Baris *r* = 1 digeser sejauh 1 *byte*, baris r = 2 digeser sejauh 2 *byte*, dan baris *r* = 3 digeser sejauh 3 *byte*. Baris *r* = 0 tidak digeser.

Contoh:

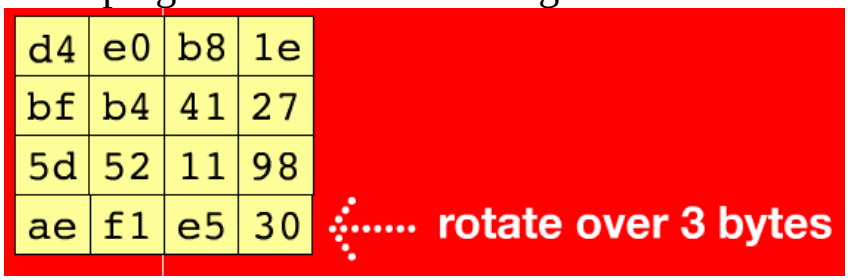
Geser baris ke-1:



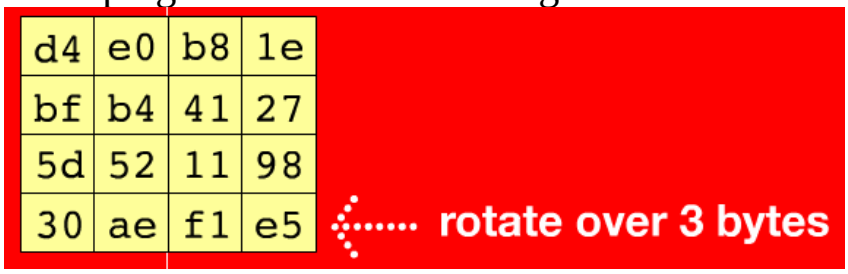
Hasil pergeseran baris ke-1 dan geser baris ke-2:



Hasil pergeseran baris ke-2 dan geser baris ke-3:



Hasil pergeseran baris ke-2 dan geser baris ke-3:



13.3.3 Transformasi *MixColumns()*

- Transformasi *MixColumns()* mengalikan setiap kolom dari *array state* dengan polinom $a(x)$ mod $(x^4 + 1)$. Setiap kolom diperlakukan sebagai polinom 4-suku pada GF(2^8).

  $a(x)$ yang ditetapkan adalah:

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

  Transformasi ini dinyatakan sebagai perkalian matriks:

  $$s'(x) = a(x) \otimes s(x)$$

  $$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

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

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

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

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

Contoh:

Hasil transformasi *ShiftRows()* sebelumnya:

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Operasi *MixColumns()* terhadap kolom pertama:

$$\begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} \bullet \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 02 & 03 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

Hasil transformasi *MixColumns()* seluruhnya:

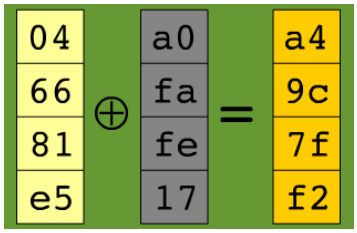
04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

13.3.4 Transformasi *AddRoundKey()*

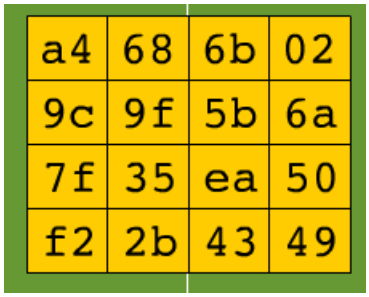
- Transformasi ini melakukan operasi XOR terhadap sebuah *round key* dengan *array state*, dan hasilnya disimpan di *array state*.

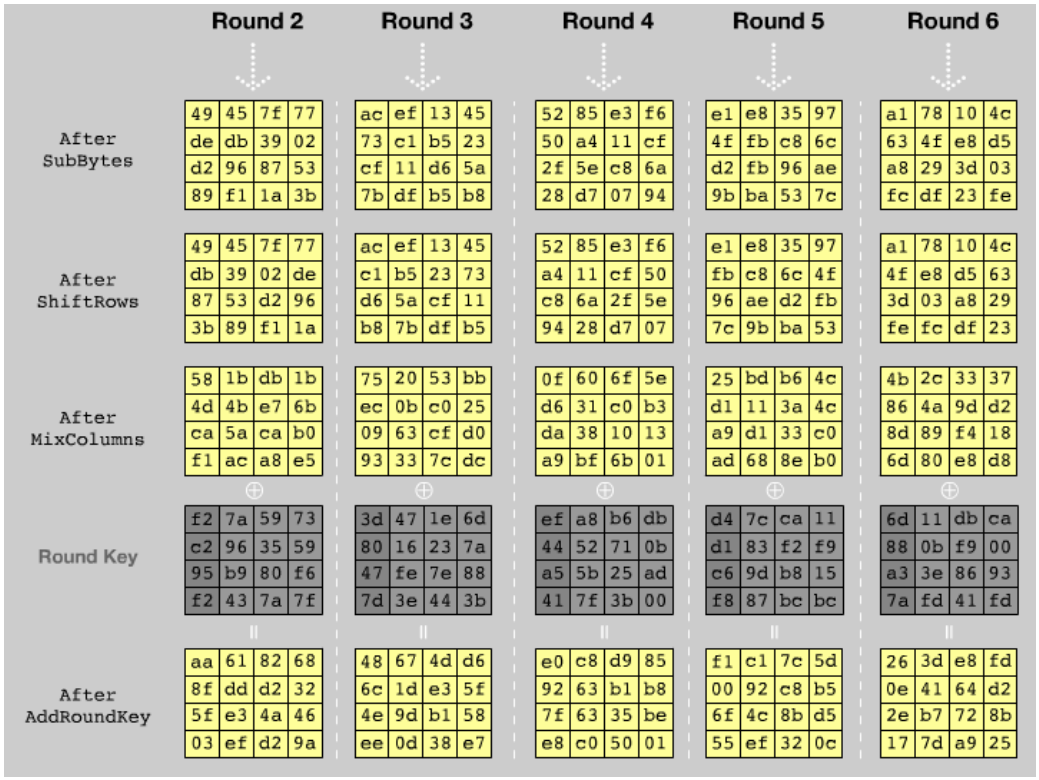
  Contoh:

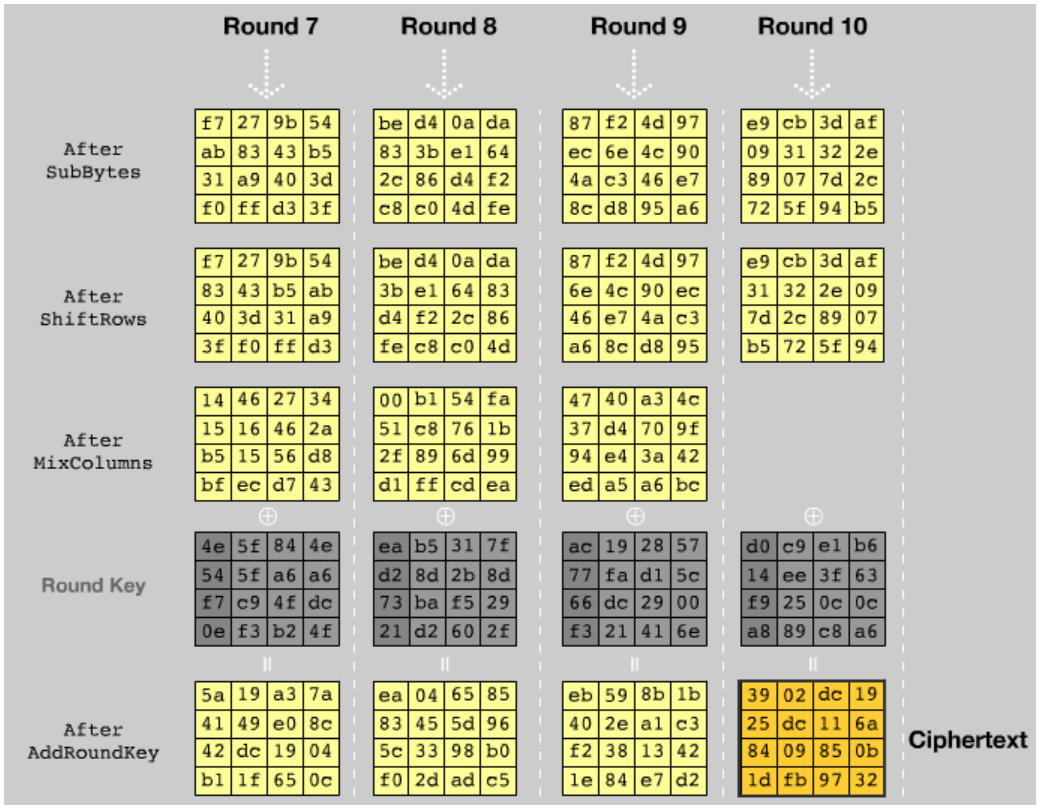
  XOR-kan kolom pertama *state* dengan kolom pertama *round key*:



  Hasil *AddRoundKey()* terhadap seluruh kolom:



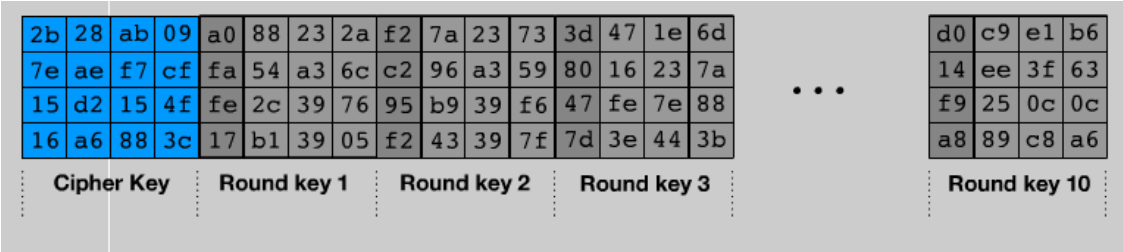

Round 2
Round 3
Round 4
Round 5
Round 6
After SubBytes
49 45 7f 77 | de db 39 02 | d2 96 87 53 | 89 f1 1a 3b
ac ef 13 45 | 73 c1 b5 23 | cf 11 d6 5a | 7b df b5 b8
52 85 e3 f6 | 50 a4 11 cf | 2f 5e c8 6a | 28 d7 07 94
e1 e8 35 97 | 4f fb c8 6c | d2 fb 96 ae | 9b ba 53 7c
a1 78 10 4c | 63 4f e8 d5 | a8 29 3d 03 | fc df 23 fe
After ShiftRows
49 45 7f 77 | db 39 02 de | 87 53 d2 96 | 3b 89 f1 1a
ac ef 13 45 | c1 b5 23 73 | d6 5a cf 11 | b8 7b df b5
52 85 e3 f6 | a4 11 cf 50 | c8 6a 2f 5e | 94 28 d7 07
e1 e8 35 97 | fb c8 6c 4f | 96 ae d2 fb | 7c 9b ba 53
a1 78 10 4c | 4f e8 d5 63 | 3d 03 a8 29 | fe fc df 23
After MixColumns
58 1b db 1b | 4d 4b e7 6b | ca 5a ca b0 | f1 ac a8 e5
75 20 53 bb | ec 0b c0 25 | 09 63 cf d0 | 93 33 7c dc
0f 60 6f 5e | d6 31 c0 b3 | da 38 10 13 | a9 bf 6b 01
25 bd b6 4c | d1 11 3a 4c | a9 d1 33 c0 | ad 68 8e b0
4b 2c 33 37 | 86 4a 9d d2 | 8d 89 f4 18 | 6d 80 e8 d8
Round Key
f2 7a 59 73 | c2 96 35 59 | 95 b9 80 f6 | f2 43 7a 7f
3d 47 1e 6d | 80 16 23 7a | 47 fe 7e 88 | 7d 3e 44 3b
ef a8 b6 db | 44 52 71 0b | a5 5b 25 ad | 41 7f 3b 00
d4 7c ca 11 | d1 83 f2 f9 | c6 9d b8 15 | f8 87 bc bc
6d 11 db ca | 88 0b f9 00 | a3 3e 86 93 | 7a fd 41 fd
After AddRoundKey
aa 61 82 68 | 8f dd d2 32 | 5f e3 4a 46 | 03 ef d2 9a
48 67 4d d6 | 6c 1d e3 5f | 4e 9d b1 58 | ee 0d 38 e7
e0 c8 d9 85 | 92 63 b1 b8 | 7f 63 35 be | e8 c0 50 01
f1 c1 7c 5d | 00 92 c8 b5 | 6f 4c 8b d5 | 55 ef 32 0c
26 3d e8 fd | 0e 41 64 d2 | 2e b7 72 8b | 17 7d a9 25


Round 7
Round 8
Round 9
Round 10
After SubBytes
f7 27 9b 54 | ab 83 43 b5 | 31 a9 40 3d | f0 ff d3 3f
be d4 0a da | 83 3b e1 64 | 2c 86 d4 f2 | c8 c0 4d fe
87 f2 4d 97 | ec 6e 4c 90 | 4a c3 46 e7 | 8c d8 95 a6
e9 cb 3d af | 09 31 32 2e | 89 07 7d 2c | 72 5f 94 b5
After ShiftRows
f7 27 9b 54 | 83 43 b5 ab | 40 3d 31 a9 | 3f f0 ff d3
be d4 0a da | 3b e1 64 83 | d4 f2 2c 86 | fe c8 c0 4d
87 f2 4d 97 | 6e 4c 90 ec | 46 e7 4a c3 | a6 8c d8 95
e9 cb 3d af | 31 32 2e 09 | 7d 2c 89 07 | b5 72 5f 94
After MixColumns
14 46 27 34 | 15 16 46 2a | b5 15 56 d8 | bf ec d7 43
00 b1 54 fa | 51 c8 76 1b | 2f 89 6d 99 | d1 ff cd ea
47 40 a3 4c | 37 d4 70 9f | 94 e4 3a 42 | ed a5 a6 bc
Round Key
4e 5f 84 4e | 54 5f a6 a6 | f7 c9 4f dc | 0e f3 b2 4f
ea b5 31 7f | d2 8d 2b 8d | 73 ba f5 29 | 21 d2 60 2f
ac 19 28 57 | 77 fa d1 5c | 66 dc 29 00 | f3 21 41 6e
d0 c9 e1 b6 | 14 ee 3f 63 | f9 25 0c 0c | a8 89 c8 a6
After AddRoundKey
5a 19 a3 7a | 41 49 e0 8c | 42 dc 19 04 | b1 1f 65 0c
ea 04 65 85 | 83 45 5d 96 | 5c 33 98 b0 | f0 2d ad c5
eb 59 8b 1b | 40 2e a1 c3 | f2 38 13 42 | 1e 84 e7 d2
39 02 dc 19 | 25 dc 11 6a | 84 09 85 0b | 1d fb 97 32
Ciphertext

13.3.5 Ekspansi Kunci

- Algoritma *AES* mengambil *cipher key*, *K*, yang diberikan oleh pengguna, dan memanggil fungsi *KeyExpansion()* untuk membangkitkan sejumlah *round key* (banyaknya *round key* bergantung pada jumlah putaran).

Contoh:



Proses pembangkitan kunci cukup kompleks, dan tidak dibahas di sini.

URL yang terkait dengan AES:

1. AES Homepage, http://www.nist.gov/CryptoToolkit
2. J. Daemen, V. Rijmen, AES Proposal: Rijndael, http://www.esat.kuleuven.ac.be/~rizmen/

Beberapa algoritma kriptografi simetri:

Cipher	Pembuat	Panjang Kunci	Keterangan
Blowfish	Bruce Schneier	1 – 448 bit	*Old and slow*
DES	IBM	56 bit	*Too weak to use now*
IDEA	Massey dan Xuejia	128 bit	*Good, but patented*
RC4	Ronald Rivest	1 – 2048 bit	*Caution: some keys are weak*
RC5	Ronald Rivest	128 – 256 bit	*Good, but patented*
Rijndael	Daemen dan Rijmen	128 – 256 bit	*Best choice*
Serpent	Anderson, Biham, Knudsen	128 – 256 bit	*Very strong*
Triple DES	IBM	168 bit	*Second best choice*
Twofish	Bruce Schneier	128 – 256 bit	*Very strong; widely used*