

BAB II

DASAR TEORI

Pada bab ini dibahas konsep dan dasar teori yang akan mendasari implementasi *disk encryption*, tepatnya *virtual hard disk encryption* dengan aspek enkripsi algoritma Rijndael. Terakhir, bab ini juga membahas sekilas mengenai *Windows driver* yang berkaitan dengan pembuatan *virtual disk*.

2.1 Disk Encryption

Disk Encryption adalah salah satu teknik enkripsi yang dilakukan pada *data at rest*. Istilah *data at rest* ditujukan pada data yang tersimpan pada media penyimpanan komputer. Dengan kata lain, enkripsi tidak dilakukan pada data yang sedang ditransmisi melalui jaringan. Media penyimpanan yang dimaksud dalam *disk encryption* merupakan peralatan/*device* yang memiliki sektor-sektor dan alamat untuk tiap sektor tersebut. Contohnya adalah *hard disk* atau *flash card*.

2.2 Hard Disk Encryption

Penerapan *disk encryption* pada *hard disk* dinamakan *hard disk encryption*. Satu sektor pada *hard disk* biasanya memiliki ukuran 512 *byte*. Istilah lain yang paling sering digunakan untuk menyebut *hard disk encryption* adalah *volume encryption*. Untuk penulisan dokumen Tugas Akhir ini, istilah *disk encryption* mengacu pada *hard disk encryption*.

Volume Encryption

Berkaitan dengan teknik enkripsi pada *data at rest*, teknik yang umum digunakan adalah enkripsi yang dilakukan pada file atau *file encryption*. Satu file yang diinginkan dienkripsi pada satu waktu. *File encryption* sudah dilakukan bertahun-tahun dan masih memiliki manfaat dari segi keamanan. Berbeda dengan *file encryption*, *disk encryption* dalam implementasinya menggunakan suatu wadah/kontainer/volume. Wadah ini digunakan sedemikian rupa sehingga tiap file yang dimasukkan ke dalam wadah ini akan otomatis terenkripsi. Dengan alasan itu, salah satu istilah lain untuk *disk encryption* adalah *volume encryption*. Wadah/

kontainer/volume ini bila dibuka dengan *password* yang tepat akan diperlakukan sebagai *disk* tersendiri (contohnya H:\ *drive*). Dengan kata lain, dengan password yang tepat, sistem enkripsi ini akan memunculkan sebuah *virtual disk*.

Dibandingkan dengan teknik *file encryption*, *disk encryption* memiliki beberapa keuntungan terutama bila jumlah file penting yang hendak dienkripsi amat banyak. Wadah yang disediakan oleh *disk encryption* memungkinkan file-file penting tersebut terkumpul dengan rapi dalam satu tempat (tidak tersebar). Penanganan kunci pun lebih mudah karena hanya diperlukan satu kunci untuk satu wadah. Dengan demikian, pengguna tidak perlu mengingat kunci mana yang digunakan untuk *file* apa. Selain itu, pengguna tidak perlu turun tangan untuk memicu proses enkripsi/dekripsi berulang kali karena sifatnya yang otomatis melakukan proses tersebut terhadap isi *virtual disk*.

Berdasarkan besarnya kontainer/volume tersebut, *disk encryption* dapat digolongkan menjadi dua yakni *entire hard disk encryption (EHD encryption)* dan *virtual hard disk encryption (VHD encryption)* [REF04].

2.2.1 EHD Encryption dan VHD Encryption

Sesuai namanya, *EHD encryption* berarti enkripsi dilakukan pada seluruh area *hard disk*, termasuk faktor yang menyangkut perangkat keras *hard disk* itu sendiri. Dengan demikian, *EHD encryption* ini hanya dapat diimplementasikan secara khusus terhadap salah satu jenis *hard disk* tertentu. Jenis enkripsi ini tentu memiliki kelebihan karena seluruh area *hard disk* aman. Namun, jika dikaitkan dengan jaringan dan kegiatan *sharing* yang biasa dilakukan di dalamnya, *EHD encryption* menuntut penyesuaian yang tidak mudah.

VHD encryption, di sisi lain, tidak melakukan pengamanan terhadap seluruh area *hard disk*, tetapi hanya sebagian, yaitu sebuah *virtual disk*. Dengan demikian, sepanjang ada kesesuaian sistem enkripsi dengan sistem operasi tempat *VHD encryption* terinstal, semua akan berjalan lancar, tidak perlu memikirkan faktor yang berkaitan dengan perangkat keras *hard disk*.

2.2.2 *Virtual Disk*

Virtual disk berarti tidak ada *disk* yang nyata dalam bentuk fisik. Walaupun demikian, sebuah *virtual disk* dapat digunakan seperti *physical disk* lainnya. *Virtual disk* dapat digunakan untuk menyimpan *file* dan *file* yang terdapat di dalamnya dapat dimodifikasi atau dihapus. *Virtual disk* merupakan *disk* yang muncul sementara, hanya selama dipanggil saja (istilahnya di-*mount*). Bentuk asli dari *virtual disk* dapat berupa *memory*. Hal tersebut berarti pada saat *virtual disk* di-*mount*, sebagian dari *memory* diperlakukan seperti layaknya *physical disk*. Selain untuk mengeksekusi program, data dapat disimpan pada bagian *memory* tersebut. Kondisi ini memungkinkan seluruh proses yang berkaitan dengan *virtual disk* yang tercipta berlangsung lebih cepat. Kelemahan cara ini adalah bila komputer dimatikan, seluruh data juga hilang (karena tersimpan di *memory*).

Cara lain yang akan digunakan dalam tugas akhir ini adalah dengan menggunakan sebuah *file*. *File* ini akan memuat struktur dan isi data yang tersimpan dalam media penyimpanan tertentu. Jenis *file* seperti ini disebut *disk image* [WIK07]. Contoh yang paling populer adalah CD/DVD *image*. Dalam pengerjaan tugas akhir ini, sebuah *disk image* akan berperan sebagai *file* kontainer/volume seperti yang telah dijelaskan pada bagian 2.2 (*Volume Encryption*).

2.2.3 *Disk Encryption dan Cipher Blok*

Disk encryption bekerja dengan *cipher* blok (*block cipher*). Rangkaian bit plainteks/cipherteks yang akan dienkripsi atau didekripsi dibagi menjadi blok-blok bit yang panjangnya sama dan sudah ditentukan sebelumnya[MUN04]. *Cipher* blok termasuk dalam tipe algoritma simetri, tipe algoritma kriptografi modern yang berarti proses enkripsi dan dekripsi memiliki struktur yang serupa. Salah satu *cipher* blok yang dapat digunakan untuk *disk encryption* adalah algoritma Rijndael.

2.3 Algoritma Rijndael [MUN04]

Algoritma Rijndael adalah pemenang sayembara terbuka yang diadakan oleh *NIST* (*National Institute of Standards and Technology*) untuk membuat standard algoritma kriptografi yang baru sebagai pengganti *Data Encryption Standard (DES)*. *DES* sudah

dianggap tidak aman terutama karena panjang kunci yang relatif pendek sehingga mudah dipecahkan menggunakan teknologi saat ini.

Algoritma *Rijndael* menggunakan substitusi, permutasi, dan sejumlah putaran yang dikenakan pada tiap blok yang akan dienkripsi/dekripsi. Untuk setiap putarannya, *Rijndael* menggunakan kunci yang berbeda. Kunci setiap putaran disebut *round key*. Tetapi tidak seperti *DES* yang berorientasi bit, *Rijndael* beroperasi dalam orientasi *byte* sehingga memungkinkan untuk implementasi algoritma yang efisien ke dalam *software* dan *hardware*. Ukuran blok untuk algoritma *Rijndael* adalah 128 bit (16 *byte*).

Algoritma *Rijndael* dapat mendukung panjang kunci 128 bit sampai 256 bit dengan step 32 bit. Panjang kunci berpengaruh pada jumlah putaran yang dikenakan pada tiap blok. Misalnya, untuk ukuran blok dan panjang kunci sebesar 128 bit ditentukan 10 putaran, sedangkan untuk ukuran blok 128 bit dan panjang kunci 256 bit jumlah putaran yang ditentukan adalah 14 putaran.

Algoritma *Rijndael* mempunyai 3 parameter sebagai berikut:

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* (untuk panjang kunci 128 bit), 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 *plaintext* menjadi *ciphertext*, status sekarang dari data disimpan di dalam *array of byte* dua dimensi, *state*, yang berukuran $NROWS \times NCOLS$. Elemen *array state* diacu sebagai $S[r,c]$, dengan $0 \leq r < 4$ dan $0 \leq c < Nc$ (Nc adalah panjang blok dibagi 32). Pada *AES*, $Nc = 128/32 = 4$.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Gambar II-1 Ilustrasi array state

Tiap elemen dari *array state* diisi dengan 8 bit teks (1 *byte*) dalam notasi HEX. Urutan pengisian dimulai dari kolom awal ($c = 0$) sampai kolom terakhir (pada contoh di atas, $c = 3$) dan dari baris awal ($r = 0$) sampai baris akhir ($s = 3$). Pada Gambar II-1, 1 *byte* teks pertama disimpan dalam notasi HEX pada $s_{0,0}$, 1 *byte* teks kedua pada $s_{1,0}$, 1 *byte* teks kelima pada $s_{0,1}$, dan seterusnya.

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

19 = 00011001 (1 *byte* pertama)

3d = 00111101 (1 *byte* kedua), dst.

Gambar II-2 Ilustrasi Pengisian *Array State*

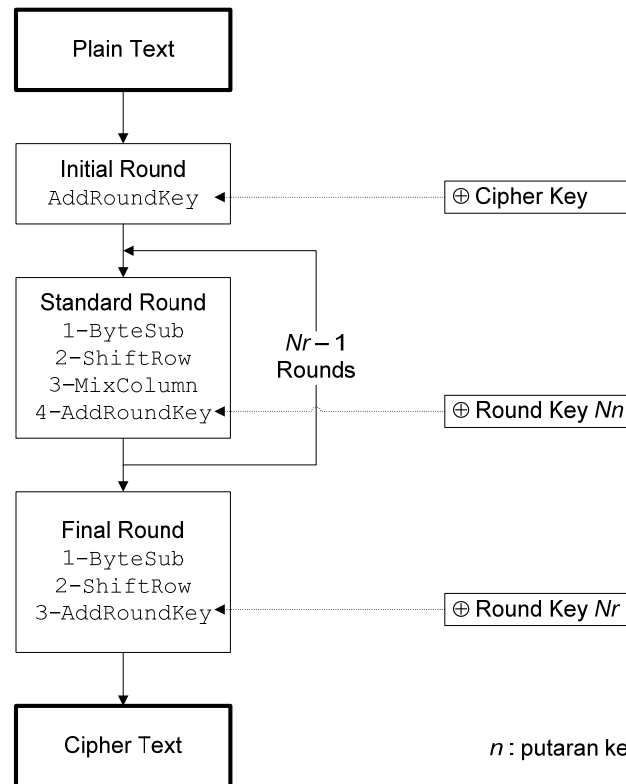
Bagian selanjutnya akan menjelaskan lebih lanjut perlakuan-perlakuan yang dikenakan pada *array state* tersebut dari awal yang berisi salinan plainteks sampai menghasilkan cipherteks.

2.3.1 Proses Enkripsi Algoritma Rijndael

Diagram proses enkripsi Rijndael dapat dilihat pada Gambar II-3.

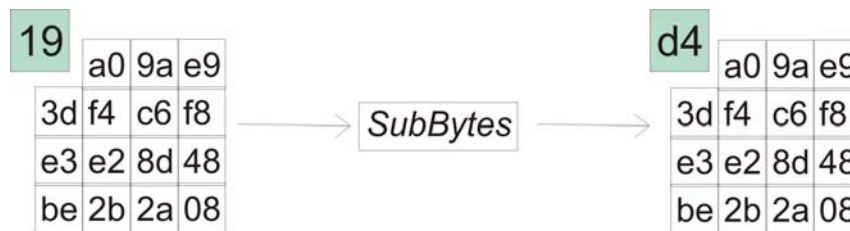
Garis besar algoritma *Rijndael* yang beroperasi blok 128-bit dengan kunci 128-bit adalah sebagai berikut:

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*). Dua digit bilangan HEX yang merupakan representasi 1 *byte* dari tiap teks menjadi koordinat untuk substitusi pada *S-box*. Digit pertama sebagai koordinat x dan digit kedua sebagai koordinat y . Tabel substitusi dapat dilihat pada Tabel II-1.



Gambar II-3 Diagram Proses Enkripsi Rijndael

Ilustrasi proses *SubBytes* dapat dilihat pada Gambar II-4. Elemen pertama dari *array state*, yaitu 19, disubstitusi menggunakan tabel *S-box* (Tabel II-1) dengan cara memasukkan nilai $x = 1$ dan nilai $y = 9$ hingga mendapatkan hasil d4. Proses ini dilakukan untuk tiap elemen pada *array state* hingga mendapatkan hasil seperti pada Gambar II-5



Gambar II-4 Ilustrasi Transformasi *SubBytes()* Rijndael

Tabel II-1 Tabel S-box yang digunakan dalam transformasi *ByteSub()* Rijndael

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	3	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

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

Gambar II-5 Hasil Transformasi *Subbytes()* Rijndael

b. **ShiftRow**: pergeseran baris-baris *array state* secara *wrapping* dengan cara menggeser baris ke-*r* dalam *array state* ke kiri sebanyak *r byte*. Ilustrasi *ShiftRow* dapat dilihat pada Gambar II-6. Baris ke-0 dari blok tidak mengalami pergeseran (Gambar II-6 a). Baris ke-1 bergeser 1 *byte* ke kiri (Gambar II-6 b), baris ke-2 bergeser 2 *byte* ke kiri (Gambar II-6 c), sampai akhirnya didapatkan hasil terakhir setelah menggeser baris ke ke-3 sebanyak 3 *byte* ke kiri (Gambar II-6 d).

a	b	c	d																																																																
<table border="1" style="border-collapse: collapse;"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>27</td><td>bf</td><td>b4</td><td>41</td></tr> <tr><td>11</td><td>98</td><td>5d</td><td>52</td></tr> <tr><td>ae</td><td>f1</td><td>e5</td><td>30</td></tr> </table>	d4	e0	b8	1e	27	bf	b4	41	11	98	5d	52	ae	f1	e5	30	<table border="1" style="border-collapse: collapse;"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>bf</td><td>b4</td><td>41</td><td>27</td></tr> <tr><td>11</td><td>98</td><td>5d</td><td>52</td></tr> <tr><td>ae</td><td>f1</td><td>e5</td><td>30</td></tr> </table>	d4	e0	b8	1e	bf	b4	41	27	11	98	5d	52	ae	f1	e5	30	<table border="1" style="border-collapse: collapse;"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>bf</td><td>b4</td><td>41</td><td>27</td></tr> <tr><td>5d</td><td>52</td><td>11</td><td>98</td></tr> <tr><td>ae</td><td>f1</td><td>e5</td><td>30</td></tr> </table>	d4	e0	b8	1e	bf	b4	41	27	5d	52	11	98	ae	f1	e5	30	<table border="1" style="border-collapse: collapse;"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>bf</td><td>b4</td><td>41</td><td>27</td></tr> <tr><td>5d</td><td>52</td><td>11</td><td>98</td></tr> <tr><td>30</td><td>ae</td><td>f1</td><td>e5</td></tr> </table>	d4	e0	b8	1e	bf	b4	41	27	5d	52	11	98	30	ae	f1	e5
d4	e0	b8	1e																																																																
27	bf	b4	41																																																																
11	98	5d	52																																																																
ae	f1	e5	30																																																																
d4	e0	b8	1e																																																																
bf	b4	41	27																																																																
11	98	5d	52																																																																
ae	f1	e5	30																																																																
d4	e0	b8	1e																																																																
bf	b4	41	27																																																																
5d	52	11	98																																																																
ae	f1	e5	30																																																																
d4	e0	b8	1e																																																																
bf	b4	41	27																																																																
5d	52	11	98																																																																
30	ae	f1	e5																																																																

Gambar II-6 Ilustrasi Transformasi *ShiftRow()* Rijndael

- c. **MixColumn**: mengacak data di masing-masing kolom *array state*. Pengacakan data dilakukan dengan mengalikan setiap kolom dari *array state* dengan polinom $a(x) \bmod (x^4 + 1)$. Setiap kolom diperlakukan sebagai polinom 4-suku pada $GF(2^8)$. Polinom $a(x)$ yang ditetapkan adalah:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (\text{II.1})$$

Transformasi ini dinyatakan sebagai perkalian matriks:

$$s'(x) = a(x) \otimes s(x) \quad (\text{II.2})$$

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

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

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

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

Ilustrasi perkalian matriks dapat dilihat pada Gambar II-7.

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

Gambar II-7 Ilustrasi Perkalian Matriks *MixColumn()* Rijndael

Ilustrasi *MixColumn* dapat dilihat pada Gambar II-8. Hasil proses sebelumnya ditunjukkan pada Gambar II-8 a. Untuk tiap kolom dari hasil tersebut dilakukan perkalian matriks seperti pada Gambar II-8 b. Setelah *MixColumns* dilakukan pada keempat kolom *array state*, didapatkan hasil seperti pada Gambar II-9

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

a

d4	●	02	01	01	03	=	04
bf		03	02	01	01		66
5d		01	03	02	01		81
30		01	01	02	03		e5

b

Gambar II-8 Ilustrasi Transformasi *MixColumn()* Rijndael

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

Gambar II-9 Hasil Transformasi *MixColumn()* Rijndael

- d. **AddRoundKey**: melakukan *XOR* antara *array state* sekarang dengan *round key*. Ilustarsi *AddRoundKey* dapat dilihat pada Gambar II-10. Tiap kolom dari *array state* dikenakan *XOR* dengan kolom *round key* yang sepadan. Proses ini dilakukan terhadap seluruh kolom pada *array state*. Contoh hasil proses *AddRoundKey* yang lengkap dapat dilihat pada Gambar II-11.

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

 \oplus

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

 $=$

a4	e0	b8	1e
9c	b4	41	27
7f	52	11	98
f2	ae	f1	e5

Keterangan:

\oplus operasi XOR

Gambar II-10 Ilustrasi Transformasi *AddRoundKey()* Rijndael

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

Gambar II-11 Hasil Transformasi *AddRoundKey()* Rijndael

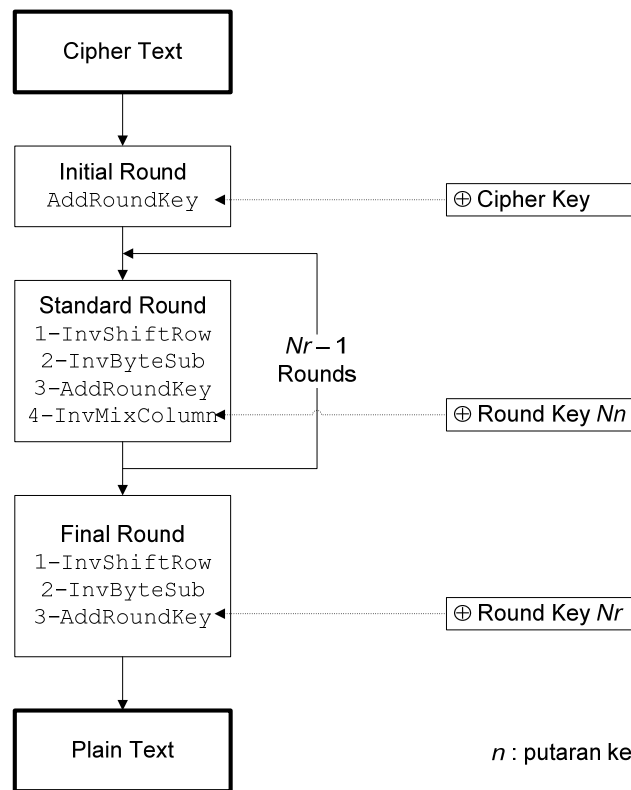
3. *Final round*: proses untuk putaran terakhir:
 - a. *ByteSub*.
 - b. *ShiftRow*.
 - c. *AddRoundKey*.

2.3.2 Proses Dekripsi Algoritma Rijndael

Diagram proses dekripsi Rijndael dapat dilihat pada Gambar II-12.

Secara garis besar, proses dekripsi algoritma Rijndael yang beroperasi blok 128-bit dengan kunci 128-bit adalah sebagai berikut:

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



Gambar II-12 Diagram Proses Dekripsi Rijndael

2. Putaran sebanyak $Nr - 1$ kali. Proses yang dilakukan pada setiap putaran adalah:
 - a. **InvShiftRow**: pergeseran baris-baris *array state* secara *wrapping*.
 - b. **InvByteSub**: substitusi byte dengan menggunakan tabel substitusi kebalikan (*inverse S-box*). Tabel substitusi dapat dilihat pada Tabel II-2.
 - c. **AddRoundKey**: melakukan *XOR* antara *state* sekarang dengan *round key*.
 - d. **InvMixColumn**: mengacak data di masing-masing kolom *array state*.
3. **Final round**: proses untuk putaran terakhir:
 - a. *InvShiftRow*.
 - b. *InvByteSub*.
 - c. *AddRoundKey*.

Tabel II-2 Tabel *S-box* yang digunakan dalam transformasi *InvByteSub()* Rijndael

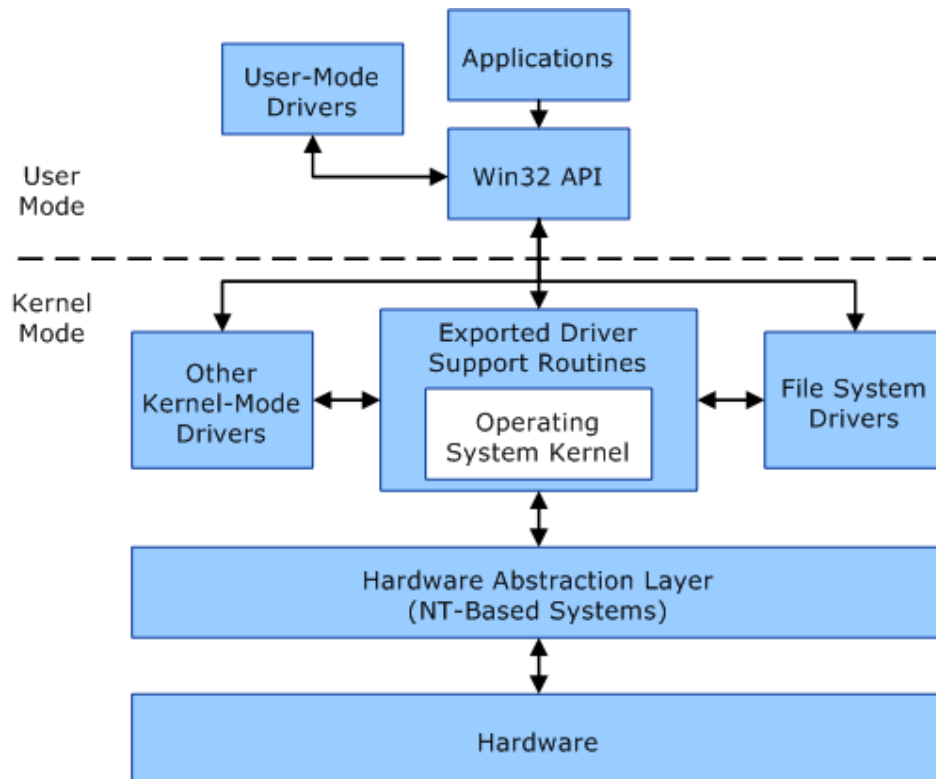
hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	a5	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	a3	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	lc	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

2.4 Device Driver

Seperti yang dipaparkan pada bagian 2.2.2, salah satu cara untuk memunculkan *virtual disk* adalah dengan memanfaatkan sebuah *file* sebagai kontainer (*file volume*). Hal ini dimungkinkan dengan membuat sebuah *driver*. *Driver* merupakan wadah koleksi fungsi dan prosedur yang sistem operasi panggil untuk menjalankan berbagai operasi yang berkaitan dengan perangkat. Dalam kasus ini (*virtual disk driver*) perangkat keras yang dimaksud adalah *hard disk*.

Seperti yang dapat dilihat pada gambar, sistem operasi memiliki komponen *user mode* dan *kernel mode*. *User mode driver* dan aplikasi yang termasuk dalam *user mode* tidak memiliki akses langsung terhadap perangkat keras dan memori. Komponen ini harus memanggil *Application Programming Interface* atau API dari setiap kali mengakses perangkat keras dan memori. Isolasi semacam itu memberikan perlindungan sehingga kerusakan yang ditimbulkan pada komponen *user mode* lebih mudah untuk ditanggulangi dibandingkan dengan pada komponen *kernel mode*.

Komponen *kernel mode* mampu melakukan operasi dan mengakses struktur sistem yang terlarang untuk komponen *user mode*, misalnya operasi yang menyangkut *Input/Output (I/O)*, konfigurasi sistem, *Plug and Play*, pengaturan memori dan fitur-fitur sistem operasi lainnya.



Gambar III-13 Komponen Sistem Operasi