Implementasi *Disk Encryption* Menggunakan Algoritma Rijndael

Elfira Yolanda S

Laboratorium Ilmu dan Rekayasa Komputasi
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
e-mail: if13087@students.if.itb.ac.id, chenvael@yahoo.com

Abstrak

Data dapat menjadi salah satu aset penting dalam kelangsungan hidup perusahaan mana pun. Salah satu teknik enkripsi yang ada yaitu *disk encryption*. Sebuah sistem *disk encryption* melibatkan wadah untuk menampung *file-file* yang ingin dienkripsi. Berdasarkan wadah tersebut, ada dua jenis *disk encryption* yaitu *Entire Hard Disk* (*EHD*) *encryption*, dengan enkripsi meliputi keseluruhan *hard disk* termasuk sistem operasinya, dan *Virtual Hard Disk* (*VHD*) *encryption*.

Dalam sebuah sistem VHD encryption, sebuah *virtual disk* dapat dimunculkan dengan memberikan password yang tepat. Dengan fungsionalitas seperti layaknya *disk drive* biasa, perbedaan yang dimiliki *virtual disk* ini adalah bahwa tiap *file* yang dimasukkan ke dalamnya otomatis terenkripsi. Perangkat lunak dalam makalah tugas akhir ini mengimplementasikan *VHD encryption* pada sistem operasi *Windows* menggunakan algoritma Rijndael sebagai algoritma enkripsi.

Kata kunci: *disk encryption*, *Windows driver, virtual disk driver*, Rijndael

1. PENDAHULUAN

Data dapat menjadi salah satu aset penting dalam kelangsungan hidup perusahaan mana pun. Penyimpanan data memerlukan berbagai macam pertimbangan, terutama dari segi keamanannya. Penggunaan *laptop/notebook* saat ini menimbulkan masalah baru berkaitan dengan penyimpanan data tersebut. Mau tidak mau, perusahaan harus mengizinkan data penting mereka "berkeliaran" di luar. Untuk menyelesaikan masalah ini, enkripsi data sebaiknya dilakukan.

Teknik enkripsi yang umum digunakan adalah enkripsi yang dilakukan pada *file* atau *file encryption*. Satu file yang diinginkan dienkripsi pada satu waktu. Teknik ini tidak efektif bila jumlah file yang harus dienkripsi banyak. *Disk encryption* merupakan alternatif enkripsi data selain *file encryption*. Enkripsi ini disebut juga enkripsi volume (*volume encryption*). Penggunaan kata volume dikarenakan enkripsi ini seakan-akan dilakukan pada area tertentu. Area itu menjadi sebuah volume/ kontainer yang dapat digunakan untuk menyimpan *file-file* penting. Berdasarkan besarnya volume tersebut, *disk encryption* dapat digolongkan menjadi dua yakni *entire hard disk encryption* (*EHD encryption*), yang proses enkripsinya meliputi seluruh wilayah hard disk termasuk sistem operasinya, dan *virtual hard disk encryption* (*VHD encryption*). VHD encryption lebih mudah untuk diimplementasikan karena tidak tergantung pada jenis *hardware* tertentu karena enkripsi dikenakan hanya pada sebagian wilayah *hard disk* saja [REF04]. Dengan alasan itulah, tugas akhir ini memilih sistem tersebut.

Mengenai cara kerja *VHD encryption*, pada dasarnya, sistem ini mengizinkan pengguna untuk membuat sebuah *file volume*, yaitu sebuah *file* yang menyimpan konfigurasi dan data *disk* yang terenkripsi. Sebuah *virtual disk* akan dimunculkan jika pengguna memberikan *password* yang tepat untuk membuka *file volume* tadi. *Virtual disk* yang muncul tidak berbeda dengan *disk* biasa, dapat digunakan untuk menyimpan *file*. *File* yang tersimpan di dalamnya pun dapat diperlakukan seperti layaknya bila tersimpan pada *disk* yang lain, yakni dapat di-*copy*, di-*paste*, di-*delete*, dan sebagainya. Satu perbedaan antara *virtual disk* ini dengan *disk* biasa adalah seluruh data yang

disimpan pada *disk* ini secara otomatis dan transparan akan dienkripsi.

Pada saat melakukan enkripsi terhadap data yang disimpan di dalamnya, *disk encryption* bekerja dengan blok bit. Oleh karena itu untuk membangun sistem enkripsi seperti ini, algoritma enkripsi yang dipakai adalah algoritma yang bekerja dengan blok bit juga. Jenis algoritma seperti itu dinamakan *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].

Banyak algoritma kriptografi cipher blok yang sudah pernah dipublikasikan. Salah satu algoritma kriptografi *cipher* blok yang terbaru adalah algoritma Rijndael. Algoritma ini adalah pemenang sayembara terbuka yang diadakan oleh *NIST* (*National Institute of Standards and Technology*) pada tahun 2001. Algoritma Rijndael menjadi standard kriptografi yang dominan paling sedikit selama 10 tahun [MUN04]. Kenyataan bahwa algoritma Rijndael merupakan algoritma kriptografi *cipher* blok yang masih baru dan belum dinyatakan tidak aman adalah dasar pemilihan algoritma ini menjadi algoritma enkripsi pada topik tugas akhir ini. Alasan lain untuk memilih algoritma Rijndael adalah algoritma ini sudah terbukti dapat diimplementasikan untuk aplikasi *disk encryption*.

Pembangunan sebuah perangkat lunak *disk encyrption* sejak awal didasarkan pada pilihan-pilihan komponen yang membentuknya. Dengan jenis *virtual hard disk encryption*, algoritma Rijndael, perangkat lunak *disk encryption* yang dibangun diharapkan dapat memberikan faktor keamanan yang cukup kuat sambil menjaga performansi tetap tinggi dan dapat digunakan secara luas. Komponen lain yang perlu dipertimbangkan dalam implementasi *disk encryption* ini adalah sistem operasi yang digunakan. *Microsoft Windows* yang sudah dikenal luas menjadi pilihan sistem operasi untuk implementasi *disk encryption* ini.

2. 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*). *DE*S 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. plainteks: *array* yang berukuran 16 *byte*, yang berisi data masukan.
2. cipherteks: *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 x 8).

Selama kalkulasi plainteks menjadi cipherteks, status sekarang dari data disimpan di dalam *array of byte* dua dimensi, state, yang berukuran NROWS x NCOLS. Elemen *array* state diacu sebagai $S[r,c]$, dengan $0 \le r < 4$ dan $0 \le c < Nc$ (*Nc* adalah panjang blok dibagi 32). Pada *AES*, $Nc = 128/32 = 4$.

Tiap elemen dari *array* `state` diisi dengan 8 bit teks (1 *byte*) dalam notasi HEX. Contoh pengisian *array* `state` dapat dilihat pada Gambar 1

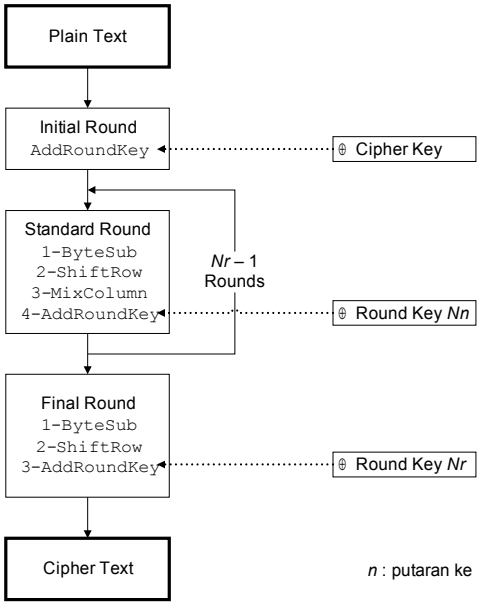
19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

19 = 00011001 (1 byte pertama)
3d = 00111101 (1 byte kedua), dst.

Gambar 1 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.

2.1. Proses Enkripsi Rijndael

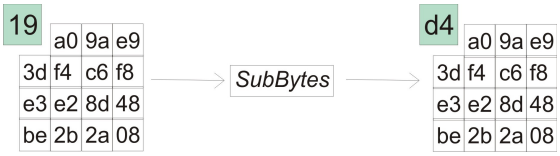


Gambar 2 Skema Enkripsi Rijndael

Skema enkripsi Rijndael dapat dilihat pada Gambar 2.

Pada intinya, tiap blok masukan (array `state`) dikenakan empat fungsi utama berikut:

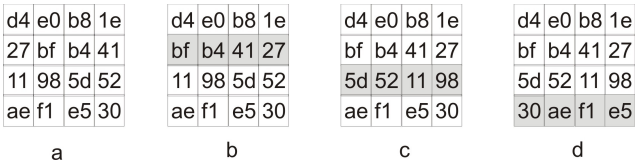
1. *SubBytes* (Gambar 3)
   Melakukan substitusi menggunakan tabel S-Box.
2. *ShiftRows* (Gambar 4)
   Menggeser baris ke-*r* dalam array `state` sebanyak *r byte* ke kiri.
3. *MixColumns* (Gambar 5)
   Mengacak array `state` dengan cara melakukan perkalian matriks yang merupakan transformasi dari perkalian polinom antara tiap kolom dengan polinom 4 suku pada GF(2^8), $a(x)$ mod $(x^4 + 1)$
4. *AddRoundKey* (Gambar 6)
   melakukan *XOR* antara *array* state sekarang dengan *round key*.



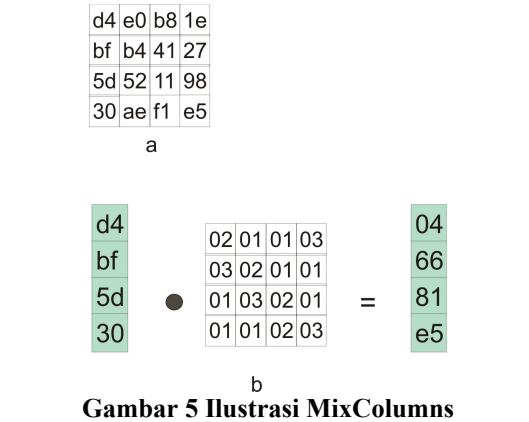
Gambar 3 Ilustrasi Transformasi *SubBytes*

Tabel 1 Tabel S-Box untuk SubBytes

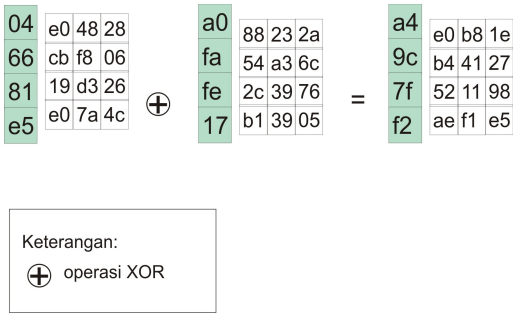
hex	y 0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x 0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	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	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



Gambar 4 Ilustrasi Transformasi *ShiftRows*



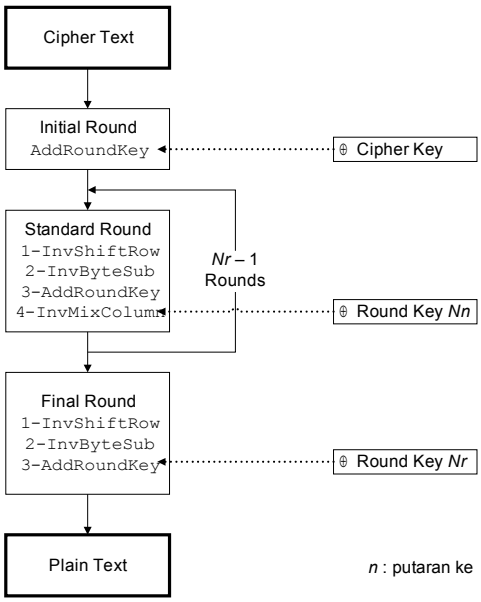
Gambar 5 Ilustrasi MixColumns



Gambar 6 Ilustrasi AddRoundKey

2.2. Proses Dekripsi Rijndael

Proses dekripsi Rijndael dapat dilihat pada Gambar 7.



Gambar 7 Skema Dekripsi Rijndael

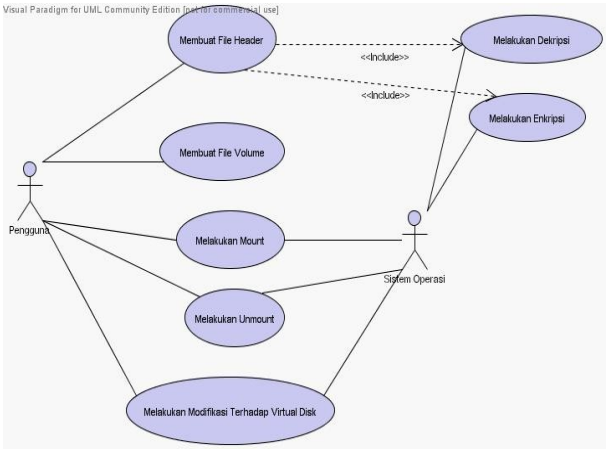
3. ANALISIS

Untuk membuat sebuah sistem *disk encyrption*, ada dua masalah yang harus diselesaikan, yaitu membuat *virtual disk* dan menyelipkan enkripsi di dalamnya.

Sebelum sampai pada jawaban masalah tersebut, Gambar 8 menunjukkan proses-proses yang terdapat dalam sebuah sistem *disk encryption*.

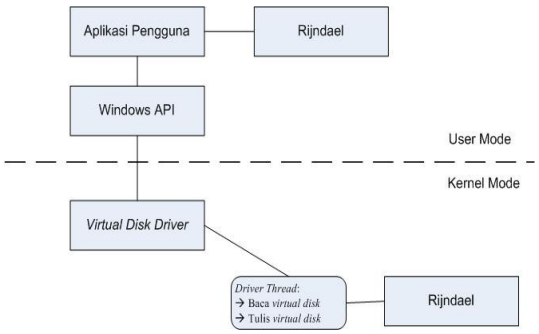
Sistem disk encryption dalam tugas akhir ini menggunakan dua buah file, yaitu *file* volume dan *file header*. File volume merupakan file yang menjadi wadah virtual disk. Tiap *file* yang nantinya akan dimasukkan ke dalam *virtual disk* sebenarnya tertulis dalam *file* ini. Sedangkan *file header* adalah *file* yang menyimpan informasi mengenai *virtual disk*.

Yang perlu diperhatikan adalah ketiga proses yang berkaitan dengan *virtual disk* berhubungan dengan pengguna Sistem Operasi. Hal tersebut disebabkan pemunculan virtual disk membutuhkan komponen *kernel mode* dalam sistem operasi.



Gambar 8 Use Case

Kernel dan *user mode* merupakan dua komponen yang terdapat dalam sistem operasi. Perbedaan keduanya adalah hak akses terhadap *hardware* dan fitur-fitur sistem operasi. Sistem *disk emcryption* membutuhkan *kernel mode* untuk mengakses *hard disk* secara langsung dan merepresentasikannya sebagai *virtual disk*. Gambar 9 memperlihatkan modul-modul yang ada dalam sistem *disk encryption*, baik dalam *user* maupun *kernel mode*.



Gambar 9 Modul-modul *disk encryption*

3.1. Menciptakan *Virtual Disk*

Untuk memunculkan sebuah *virtual disk*, diperlukan peran dari komponen *user mode* dan *kernel mode*. Aplikasi Pengguna yang bergerak dalam user mode dapat menerima informasi seperti nama *file* volume, ukuran *virtual disk*, juga nama *drive* yang diinginkan. Informasi itu diberikan oleh pengguna Kemudian, aplikasi pengguna bertugas memberi tahu *virtual disk driver* untuk membuat *virtual disk* dengan spesifikasi yang telah diberikan dengan memanggil fungsi *Windows API* yang sesuai.

Fungsi *Windows API* yang digunakan adalah DeviceIoControl. Parameter yang digunakan fungsi ini adalah sebagai berikut:

```
BOOL DeviceIOControl(
    HANDLE hDevice,
    DWORD dwIoControlCode,
    LPVOID lpInBuffer,
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped

  );
```

Penanganan *request* yang berasal dari komponen *user mode* melalui *Windows API* merupakan salah satu fungsi utama *driver*. Untuk kepentingan *request* tersebut, *driver* memiliki *control code* yang spesifik untuk melakukan operasi-operasi yang menjadi fungsionalitasnya. *Control code* tersebut akan menjadi masukan DeviceIoControl pada parameter dwIoControlCode. Pada kasus *virtual disk*, Aplikasi Pengguna akan mengirimkan kode untuk menciptakan *virtual disk* kepada *driver* pada saat menu *mount* dipilih.

Selain untuk memicu fungsi driver tertentu, fungsi DeviceIoControl juga dapat dimanfaatkan untuk mengirimkan informasi yang diperlukan untuk melakukan fungsi tersebut maupun menerima hasil fungsi driver. Parameter yang berkaitan adalah *lpInBuffer* dan *nInBufferSize* untuk masukan pada *driver* serta *lpOutBuffer* dan *nOutBufferSize* untuk menampung hasil dari *driver*.

Setelah mendapatkan kode untuk menciptakan *virtual disk* dan informasi yang dibutuhkan, *driver* melakukan pemeriksaan berkaitan dengan *virtual disk* yang akan dibuat, seperti kapasitas *hard disk* yang masih mencukupi atau tidak. Apabila *file* volume sukses dibuat, *driver* akan menciptakan sebuah *device* yang dihubungkan dengan *file* volume tersebut. Untuk memunculkan *virtual disk*, *device* akan diemulasikan sebagai *hard disk* dengan memasukkan parameter tertentu pada fungsi penciptaan *device*. Bersamaan dengan penciptaan *device* tersebut, sebuah *thread* akan diikutsertakan selama *device* tersebut ada. Untuk kasus *virtual disk*, *thread* tersebut akan ada selama *virtual disk* muncul. Sub bab berikut akan membahas lebih lanjut mengenai *thread* dalam *virtual disk driver*.

3.2. *Thread* dalam *Virtual Disk*

Thread adalah satuan unit eksekusi. *Thread* memiliki tingkat prioritas yang mempengaruhi persaingan untuk mendapatkan waktu kendali atas prosesor. Semakin tinggi tingkat prioritasnya, semakin cepat *thread* tersebut dapat menggunakan CPU. Untuk *virtual disk driver*, sebuah *thread* disediakan untuk menangani proses-proses yang perlu dilakukan selama *virtual disk* berjalan. Prioritas *thread* tersebut diatur sedemikian rupa sehingga selama *virtual disk* masih ada, alokasi CPU akan diberikan untuk *thread driver* tersebut untuk memeriksa apakah ada *I/O request* terhadap *virtual disk* atau apakah ada permintaan untuk melakukan *unmount*. Apabila terdapat permintaan untuk melakukan *unmount*, *thread* akan dihentikan. [BAK00]

I/O request yang diterima *virtual disk driver* dapat berupa pembacaan maupun penulisan *virtual disk*. Dengan penggunaan *thread* seperti itu, akses terhadap *hard disk* maupun proses baca

– tulis *virtual disk* dapat dilaksanakan dalam waktu yang lebih singkat.

4. PERANCANGAN

Poin – poin perancangan yang perlu diperhatikan di antaranya perancangan menu interaksi, pembuatan file header dan penyelipan proses enkripsi dan dekripsi pada virtual disk.

4.1. Perancangan Menu Interaksi

Interaksi dengan pengguna untuk perangkat lunak tugas akhir adalah melalui *command line*.

Sintaks yang digunakan:

1. Untuk melakukan *mount*

```
TA –a –t file –m [drive:] –f
[file volume] –h [file header] –s
[ukuran file volume] –u [nomor
unit]
```

keterangan:
[drive:] : nama *drive* untuk *virtual disk*
[file volume] : alamat *file* volume
[file header] : alamat *file* header
[ukuran file volume] : ukuran *file* volume
[nomor unit] : nomor *virtual disk*

Sintaks ini digunakan baik setiap akan melakukan *mount* terhadap *virtual disk*, baik saat pertama kali (dengan membuat *file* volume dan *file header* terlebih dahulu) maupun selanjutnya.

Contoh penggunaan:

```
TA –a –t file –m z: –f
c:\file1.img –h c:\file1.header –
s 10M
```

2. Untuk melakukan *unmount*

```
TA –d – u [nomor unit]
```

keterangan:
[drive:] : nama *drive* untuk *virtual disk*
[nomor unit] : nomor *virtual disk*

Contoh penggunaan:

```
TA –d –u 1
```

4.2. File Header

Bagian ini khusus membahas *file header* yaitu *file* yang menyimpan informasi penting mengenai *virtual disk*. Gambar 10 berikut merupakan skema *file header*:

Gambar 10 Skema *File Header*

File header merupakan *string* yang terdiri dari *salt* dan informasi *virtual disk*. *Salt* merupakan bit-bit acak yang digunakan bersama dengan kunci yang diberikan pengguna untuk menurunkan kunci *header*. Aturan yang digunakan untuk menurunkan kunci *header* mengikuti aturan PKCS #5 yang dikeluarkan oleh laboratorium RSA [RSA99]. Penggunaan bit-bit acak atau *salt* dalam PKCS #5 bertujuan meningkatkan jumlah kemungkinan kunci yang bisa diturunkan sehingga tidak mudah ditebak.

Kunci *header* dibutuhkan untuk melakukan enkripsi informasi *virtual disk* pada saat *file header* pertama kali diciptakan. Selanjutnya, kunci ini digunakan untuk mendekripsi informasi *virtual disk* saat *virtual disk* akan di-*mount*. Apabila kunci *header* yang diturunkan tepat, bagian pertama dari informasi *virtual disk* yang terdekripsi akan memberikan hasil "TRUETRUETRUETRUE". Kemudian informasi selebihnya, yaitu nama *file* volume dan kunci enkripsi *virtual disk*, dapat diekstrak.

4.3. Enkripsi/Dekripsi pada *Virtual Disk*

Modul *Virtual Disk Driver* bertanggung jawab untuk berhubungan dengan *hard disk* untuk membuat *file* volume, merepresentasikan *file* volume tersebut sebagai *virtual disk*, mengolah tiap modifikasi yang dilakukan dan menutup *virtual disk* kembali. Untuk pengerjaan tugas akhir ini, *virtual disk driver* yang digunakan adalah ImDisk (http://www.ltr-data.se/opencode.html#ImDisk) oleh Olof Lagerkvist.

Enkripsi dan dekripsi diikutsertakan dalam proses tulis dan baca terhadap *virtual disk* menggunakan impelementasi algoritma Rijndael berdasarkan AESLib dalam Encrypt It (http://msdn.microsoft.com/msdnmag/issues/03/11/aes/default.aspx).

5. IMPLEMENTASI

5.1. Lingkungan Implementasi

Spesifikasi perangkat keras yang digunakan selama pengerjaan tugas akhir adalah sebagai berikut:

1. Prosesor Intel Celeron 1.73 GHz
2. RAM 512 Mb
3. *Hard Disk* Toshiba 80 GB
4. Perangkat keluaran berupa monitor
5. Perangkat masukan berupa papan kunci dan tetikus

Adapun perangkat lunak yang digunakan:

1. Sistem operasi Microsoft Windows XP *Service Pack* 2
2. *Windows Driver Kit* (WDK) *Windows Server 2008 Pre-RTM with HyperV*
3. VMWare 5
4. Editor Notepad++ v4.0.2

5.2. Batasan Implementasi

Batasan yang didefinisikan untuk implementasi *disk encryption* ini adalah *disk encryption system* ini tidak memberikan perlindungan keamanan terhadap keberadaan *file* volume. Segala bentuk modifikasi, termasuk penghapusan *file* volume maupun pemindahan lokasi *file volume*, tidak ditangani.

6. PENGUJIAN

Pengujian yang dilakukan bertujuan untuk memeriksa keberhasilan fungsi *mount*, *unmount virtual disk* serta keberhasilan melakukan modifikasi terhadap isi *virtual disk*. Selain itu, pengujian juga dilaksanakan untuk memastikan *file* volume pada *disk encryption system* tidak dapat dibaca di luar *virtual disk*.

Berdasarkan pengujian yang dilakukan, perangkat lunak tugas akhir ini dapat menciptakan *virtual disk* yang memiliki fungsionalitas seperti layaknya *disk drive* lainnya. Pada saat *virtual disk* muncul, segala operasi yang dikenakan pada *file* dapat dilakukan dengan baik.

Pada saat *virtual disk* telah selesai digunakan dan telah di-*unmount*, pembacaan terhadap *file* volume tidak menunjukkan isi *file* yang sempat dimasukkan dalam *virtual disk* dalam keadaan terdekripsi. Dengan demikian, data yang tersimpan dalam *virtual disk* dinyatakan aman.

7. KESIMPULAN

Beberapa kesimpulan yang dapat diambil dari implementasi disk encryption menggunakan algoritma Rijndael ini adalah:

1. *Virtual disk* pada sistem operasi Windows dapat dibuat dengan berbagai cara, di antaranya memanfaatkan sebagian memori dan merepresentasikannya sebagai *virtual disk*. Namun, cara tersebut tidak memberikan hasil yang memuaskan karena isi *virtual disk* akan hilang sejalan dengan penutupan *virtual disk*. Cara yang lebih aman adalah menggunakan *file* sebagai kontainer isi *virtual disk*. *File* tersebut, yang diberi nama *file volume*, akan direpresentasikan sebagai *virtual disk*.
2. Proses representasi *file volume* sebagai *virtual disk* membutuhkan sebuah *driver*. *Driver* merupakan wadah koleksi fungsi dan prosedur yang sistem operasi panggil untuk menjalankan berbagai operasi yang berkaitan dengan perangkat keras, dalam hal ini perangkat keras yang dimaksud adalah *hard disk*.
3. Proses enkripsi pada *virtual disk* diikutsertakan pada penanganan proses baca dan tulis *virtual disk*.

REFERENSI

[BAK00] Baker, Art and Jerry Lozano. (2000). *The Windows 2000 Device Driver Book, A Guide for Programmers, Second Edition*. Prentice Hall PTR.

[MUN04] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[REF04] Reflex Magnetic Ltd. (2004). *Entire Hard Disk Encryption vs Virtual Hard Disk Encryption*.

[RSA99] RSA Laboratories. (1999). PKCS #5 v 2.00: *Password Based Cryptography Standard*