

## **BAB III**

# **ANALISIS DAN PERANCANGAN**

Pada bab ini dijelaskan analisis dan perancangan untuk melakukan implementasi *disk encryption* menggunakan algoritma Rijndael.

### **3.1 Analisis**

Fungsionalitas utama dari perangkat lunak tugas akhir ini adalah membuat *file* volume sesuai masukan pengguna dan melakukan *mount* atau *unmount virtual disk* yang terenkripsi dari *file* volume tersebut. Pada saat membuat *file* volume, pengguna memberikan masukan seperti kapasitas yang nantinya akan menjadi kapasitas *virtual disk* dan *password* untuk memunculkan *virtual disk* tersebut.

#### **3.1.1 Analisis Masalah**

Analisis masalah perangkat lunak yang akan dibangun adalah sebagai berikut:

1. Bagaimana membuat *file* volume yang dapat direpresentasikan sebagai *virtual disk* pada sistem operasi Windows.
2. Bagaimana menyelipkan proses enkripsi/dekripsi pada *virtual disk driver* saat melakukan *unmount/mount virtual disk*.

#### **3.1.2 Analisis Kebutuhan**

Analisis kebutuhan perangkat lunak ini adalah sebagai berikut:

1. Mampu membuat *file* volume yang tidak terbaca bila tidak dilakukan proses *mount* dengan *password* yang tepat.
2. Mampu menampilkan *virtual disk* setelah dilakukan *mount* pada *file* volume tertentu dengan *password* yang tepat.
3. Mampu menjalankan segala operasi pembuatan, modifikasi dan penghapusan *file* pada *virtual disk* seperti layaknya operasi yang terjadi pada *physical disk* lainnya.
4. Mampu menutup *virtual disk* (*virtual disk* tidak tampak kembali) setelah dilakukan *unmount* dan isi dari *file* volume tetap tidak terbaca.

### 3.1.3 Tujuan Pengembangan

Perangkat lunak ini dikembangkan dengan tujuan memberikan keamanan data pada media penyimpanan *hard disk* dengan cara menciptakan *virtual disk* yang otomatis terenkripsi. Dengan demikian, pengguna dapat meletakkan data yang ingin dienkripsi dalam *virtual disk* tersebut. *Virtual disk* hanya akan muncul bila password yang diberikan pengguna tepat, sebaliknya data yang ada dalam *virtual disk* tersimpan dalam bentuk *file* yang tidak terbaca.

### 3.1.4 Batasan Rancangan Sistem

Batasan rancangan sistem dalam pengembangan perangkat lunak ini adalah sebagai berikut:

1. Perangkat lunak menciptakan *virtual disk* dengan kapasitas terbesar sama dengan kapasitas *partition disk* yang masih tersedia.
2. Ukuran blok untuk algoritma Rijndael adalah 16 *byte*.

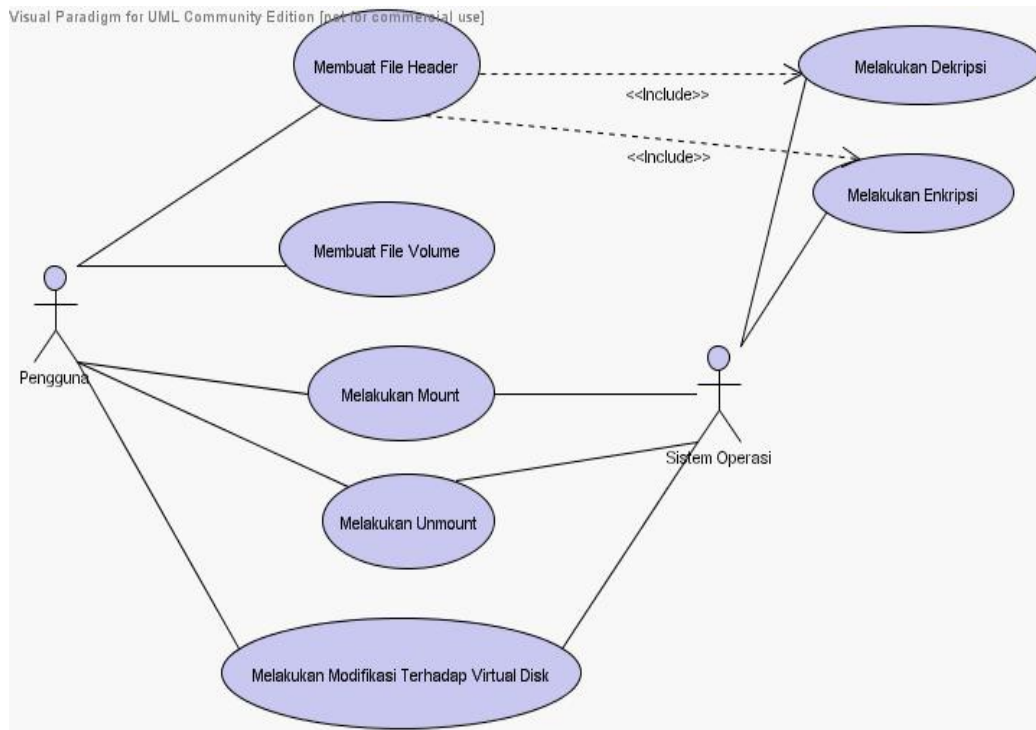
### 3.1.5 Use Case

Perangkat lunak dalam tugas akhir ini memiliki tujuh proses utama yakni membuat *file volume* dan *file header*, melakukan *mount*, melakukan *unmount*, melakukan modifikasi terhadap *virtual disk*, melakukan enkripsi dan dekripsi. Aktor yang terlibat dalam sistem ini yaitu pengguna dan sistem operasi, seperti yang tampak pada Gambar III-1.

Untuk lebih jelasnya, proses yang terjadi dalam perangkat lunak tugas akhir ini adalah sebagai berikut:

1. Membuat *file volume*.

Proses ini adalah proses yang wajib dilakukan oleh pengguna. Proses ini memerlukan masukan berupa kapasitas *virtual disk* yang diinginkan dan password untuk membuka *virtual disk* tersebut serta lokasi tempat *file volume* disimpan. Besar *file volume* yang dibuat akan sama dengan kapasitas *virtual disk*. *File volume* ini tidak dapat dibaca isinya.



**Gambar III-1 Use Case Diagram**

2. Membuat *file header*

Sebuah *file header* akan diciptakan untuk setiap *file volume* untuk menyimpan informasi yang dibutuhkan untuk melakukan *mount virtual disk*.

3. Melakukan *mount*

Proses ini hanya dapat dilakukan bila sudah terdapat *file volume*. Masukan untuk proses ini adalah *file volume* yang ingin dibuka dan *password* untuk membuka *file volume* tersebut. Apabila proses ini berjalan lancar, sebuah *virtual disk* akan muncul. Fungsi *virtual disk* ini tidak berbeda dengan *physical disk* lain.

4. Melakukan *unmount*

Proses ini dilakukan untuk menutup *virtual disk* yang sedang dibuka supaya data yang ada di dalamnya tidak dapat dibaca orang lain.

5. Melakukan modifikasi terhadap *virtual disk*

Proses ini meliputi penanganan setiap perubahan isi *virtual disk* selama dimunculkan.

6. Melakukan enkripsi

Proses ini melibatkan algoritma Rijndael untuk mengenkripsi data yang akan disimpan *hard disk*, tepatnya data yang tercatat pada lokasi *virtual disk*.

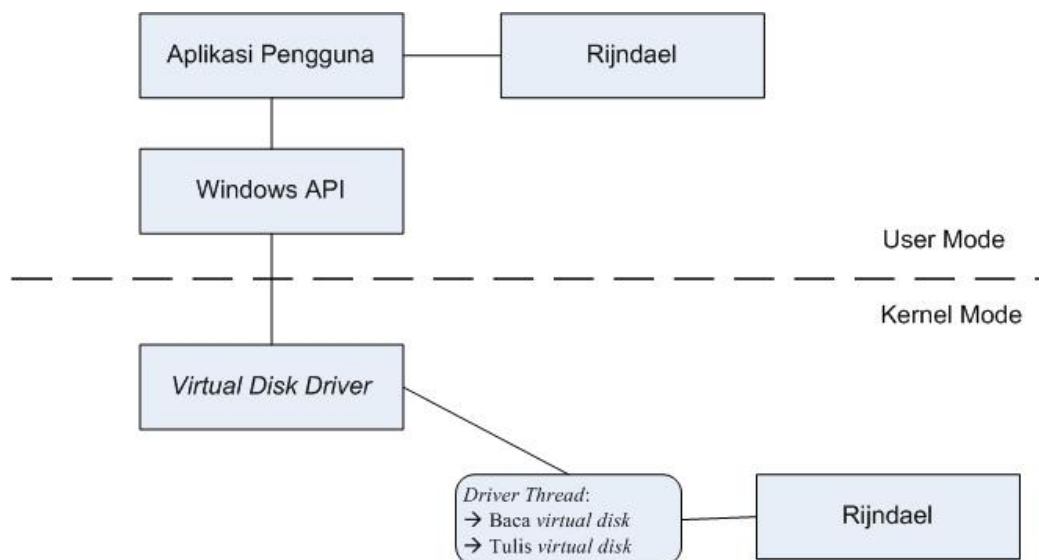
### 7. Melakukan dekripsi

Proses ini dipanggil bersamaan dengan proses melakukan *mount*. Data yang terenkripsi dalam *hard disk* didekripsi terlebih dahulu sebelum ditampilkan bersama dengan *virtual disk*.

#### 3.1.6 Identifikasi Modul

Perangkat lunak tugas akhir ini terdiri atas dua komponen yaitu *user mode* dan *kernel mode*. Modul yang termasuk dalam *user mode* adalah modul aplikasi pengguna dan modul Rijndael. Modul aplikasi pengguna menangani segala proses yang tidak membutuhkan akses ke *kernel*, di antaranya interaksi dengan pengguna serta pembuatan *file header*. Untuk tugas akhir ini, interaksi dilakukan menggunakan *command line*. Modul Rijndael memuat fungsi dan prosedur yang dibutuhkan untuk memberikan faktor keamanan perangkat lunak. Untuk menghubungkan *user mode* dan *kernel mode*, perangkat lunak tugas akhir ini menggunakan Windows API.

Modul yang termasuk dalam *kernel mode* adalah modul *virtual disk driver* dan modul Rijndael. Modul *virtual disk driver* menangani segala sesuatu yang berkaitan dengan *virtual disk*, mulai dari penciptaan dan konfigurasi *file volume*, proses *mount* dan *unmount*. Untuk lebih jelasnya, keterhubungan antar modul dapat dilihat pada Gambar III-2



Gambar III-2 Keterkaitan Antar Modul

Tabel III-1 Modul yang terlibat

Proses	Modul yang terlibat
Membuat <i>file volume</i>	<ul style="list-style-type: none"> <li>- Aplikasi Pengguna</li> <li>- <i>Virtual Disk Driver</i></li> <li>- Windows API</li> <li>- Rijndael</li> </ul>
Membuat <i>file header</i>	<ul style="list-style-type: none"> <li>- Aplikasi Pengguna</li> <li>- Rijndael</li> </ul>
Melakukan <i>mount</i>	<ul style="list-style-type: none"> <li>- Aplikasi Pengguna</li> <li>- <i>Virtual Disk Driver</i></li> <li>- Windows API</li> <li>- Enkripsi</li> </ul>
Melakukan <i>unmount</i>	<ul style="list-style-type: none"> <li>- Aplikasi Pengguna</li> <li>- <i>Virtual Disk Driver</i></li> <li>- Windows API</li> <li>- Rijndael</li> </ul>
Melakukan modifikasi <i>virtual disk</i>	<ul style="list-style-type: none"> <li>- <i>Virtual Disk Driver</i></li> <li>- Rijndael</li> </ul>
Melakukan enkripsi	<ul style="list-style-type: none"> <li>- Rijndael</li> </ul>
Melakukan dekripsi	<ul style="list-style-type: none"> <li>- Rijndael</li> </ul>

### 3.1.7 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,           // handle to device of interest
    DWORD dwIoControlCode,    // control code of operation to
                                // perform
    LPVOID lpInBuffer,       // pointer to buffer to supply input
                                // data
    DWORD nInBufferSize,    // size of input buffer
    LPVOID lpOutBuffer,     // pointer to buffer to receive
                                // output data
    DWORD nOutBufferSize,   // size of output buffer
    LPDWORD lpBytesReturned, // pointer to variable to receive
                                // output byte count
                                // pointer to overlapped structure
    LPOVERLAPPED lpOverlapped // for asynchronous operation
);

```

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*.

### 3.1.8 Thread dalam *Virtual Disk Driver*

*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. [ART00]

*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.

## 3.2 Perancangan

Bagian ini mencakup perincian modul yang telah teridentifikasi pada tahap analisis. Perincian yang dimaksud mencakup tanggung jawab proses yang dimiliki oleh setiap modul termasuk *pseudo code* fungsi/prosedur utama yang berkaitan dengan proses tersebut.

### 3.2.1 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* yang akan di-*mount*

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* yang akan di-*mount*

Contoh penggunaan:

```
TA -d -u 1
```

### 3.2.2 File Header

Sub bab ini khusus membahas *file header* yaitu *file* yang menyimpan informasi penting mengenai *virtual disk*. Gambar III-3 berikut merupakan skema *file header*:



**Gambar III-3 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 (rincian terdapat pada LAMPIRAN A). 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.

### 3.2.3 Perancangan Enkripsi/Dekripsi pada *Virtual Disk Driver*

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*.

### 3.2.4 Perancangan Modul Rijndael

Modul ini bertanggung jawab atas algoritma enkripsi yang dipakai sebagai aspek keamanan dalam tugas akhir ini. Untuk implementasi tugas akhir ini, modul Rijndael yang digunakan berdasarkan implementasi Rijndael dari AESLib dalam Encrypt It (<http://msdn.microsoft.com/msdnmag/issues/03/11/aes/default.aspx>).

Prosedur enkripsi(input/output arrBlok: array of byte; input kunci: byte)

Masukan : blok-blok byte dengan ukuran 1 blok = 16 byte,  
kunci yang merupakan password pengguna

Keluaran: blok-blok byte yang telah terenkripsi

Proses : tiap blok diperlakukan sesuai dengan aturan-aturan enkripsi algoritma Rijndael.

Prosedur dekripsi(input/output arrBlok: array of byte; input kunci: byte)

Masukan : blok-blok byte dengan ukuran 1 blok = 16 byte,  
kunci yang merupakan password pengguna

Keluaran: blok-blok byte yang telah terdekripsi

Proses : tiap blok diperlakukan sesuai dengan aturan-aturan dekripsi algoritma Rijndael.