Pembangunan Perangkat Lunak untuk Enkripsi Folder dengan Algoritma Serpent

Bernardino Madaharsa Dito Adiwidya
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10
Bandung 40132, Indonesia
if17089@students.if.itb.ac.id

Dr.Ir. Rinaldi Munir, M.T.
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10
Bandung 40132, Indonesia
rinaldi@informatika.org

ABSTRAK

Pada saat ini, penyimpanan data dalam media digital sudah banyak digunakan. Dari data dapat diperoleh informasi yang berguna. Tidak semua pihak berhak untuk mengetahui suatu informasi karena bersifat rahasia. Oleh karena itu, penyimpanan data harus diperlakukan dengan khusus dengan menggunakan kriptografi. Salah satu cara untuk melakukan pengamanan data adalah dengan mengenkripsi folder.

Algoritma Serpent merupakan salah satu algoritma enkripsi cipherblock. Algoritma ini merupakan runner-up dalam kompetisi Advanced Encryption Standard (AES). Jumlah putaran proses berulang yang digunakan dalam algoritma ini paling banyak dari antara algoritma-algoritma kandidat AES yang lain, yaitu 32. Proses yang dilakukan sejumlah 32 kali tersebut terdiri atas operasi pencampuran kunci, proses nilai S-boxes, dan transformasi linear.

Implementasi dalam penelitian ini berupa aplikasi pengenkripsi folder berbasis desktop bernama "DACrypt" yang dibuat dengan kakas Microsoft Visual Studio 2008 dengan bahasa C# dan berjalan di atas sistem operasi Windows. Sebenarnya saat ini telah ada aplikasi pengenkripsi folder sejenis, akan tetapi aplikasi-aplikasi tersebut memiliki kelemahan seperti harus melakukan dekripsi keseluruhan jika ingin menambah, mengambil, atau menghapus subfolder atau file yang ada dalam folder terenkripsi. Selain itu, pada beberapa aplikasi, folder masih dapat diakses pada Windows Explorer. Oleh karena itu, dalam implementasi ini seluruh isi folder disatukan dalam satu file dan digunakan struktur header khusus untuk mempermudah pengaksesan subfolder atau file tersebut.

Aplikasi ini mampu untuk menampilkan daftar isi file atau folder yang dikandung beserta tree navigasi folder seperti pada Windows Explorer. Algoritma Serpent dan pengetahuan tentang struktur tree dimanfaatkan untuk aplikasi ini. Aplikasi yang dibuat ini dapat berjalan sesuai dengan fungsionalitas utamanya, yaitu dapat menampilkan isi folder terenkripsi ke GUI, mengenkripsi dan mendekripsi keseluruhan atau beberapa file atau folder, dan menambahkan serta menghapus file atau folder dalam folder terenkripsi.

Kata kunci

DACrypt, enkripsi, file, folder, kriptografi, Serpent, struktur header.

1. PENDAHULUAN

Pada saat ini, penyimpanan data menjadi sesuatu yang penting dalam berbagai kepentingan, baik bagi individu maupun suatu organisasi. Dalam alat-alat seperti *personal computer*, *laptop*, *netbook*, dan *smartphone*, data yang tersimpan berbentuk *file*. Data tersebut dapat diolah menjadi suatu informasi, yang belum tentu dapat diketahui semua pihak sehingga diperlukan suatu cara penyembunyian tertentu sehingga hanya dapat digunakan oleh pihak yang berhak saja dengan memanfaatkan kriptografi. Kriptografi merupakan ilmu sekaligus seni untuk menjaga keamanan pesan [4].

Pada umumnya, *file-file* yang dimiliki oleh pengguna disimpan dalam suatu wadah yang disebut dengan *folder*. Suatu *folder* dapat menampung banyak *file* dan *folder* sehingga membentuk struktur pohon. *Folder* tersebut dapat dienkripsi dengan menggunakan aplikasi tertentu. Contoh aplikasi pengenkripsi *folder* tersebut antara lain Folder Protector [3], Secure Folder [7], Folder Lock [6], dan AxCrypt [2]. Aplikasi-aplikasi tersebut berjalan di atas sistem operasi Windows.

Pada seluruh aplikasi tersebut, untuk mendekripsi suatu *file* atau *folder* yang berada dalam *folder* terenkripsi, perlu dilakukan pendekripsian pada seluruh isi dalam *folder* yang terenkripsi. Seringkali pengguna hanya ingin mendekripsi sebuah *file* saja dalam *folder* terenkripsi yang berukuran besar. Setelah melalui proses dekripsi dengan menggunakan aplikasi tertentu, pengguna tersebut tidak hanya memperoleh *file* yang terdekripsi tetapi juga *file-file* lain yang tidak diperlukannya. Setelah didekripsi, seluruh isi *folder* tersebut harus dikembalikan dengan mengenkripsi kembali secara keseluruhan. Begitu pula jika ingin melakukan penambahan dan penghapusan *file* atau *folder* dalam *folder* terenkripsi.

Setiap aplikasi yang telah ada belum tentu menggunakan algoritma yang sama. Usaha untuk melakukan serangan kriptanalisis selalu ada. Untuk mencegah serangan untuk kriptanalisis tertentu, diperlukan suatu algoritma yang dianggap masih aman dalam serangan kriptanalisis saat ini, yakni algoritma yang belum dilaporkan telah berhasil dipecahkan secara utuh.

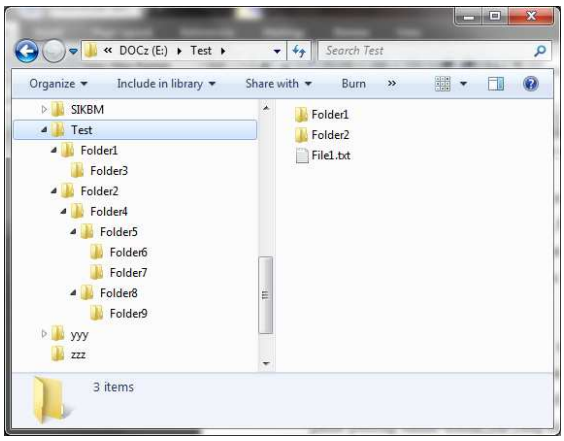
Algoritma yang digunakan untuk membuat aplikasi pengenkripsi *folder* ini tentu saja dipilih berdasarkan belum adanya laporan serangan kriptanalisis yang berhasil memecahkan algoritma tersebut secara utuh. Acuan pemilihan algoritma adalah berdasarkan kandidat kompetisi AES yang lolos putaran kedua. Algoritma yang dipilih adalah algoritma Serpent karena diklaim lebih aman [1] karena menggunakan 32 putaran meskipun akibatnya prosesnya menjadi lebih lambat [8].

2. TEORI YANG DIGUNAKAN

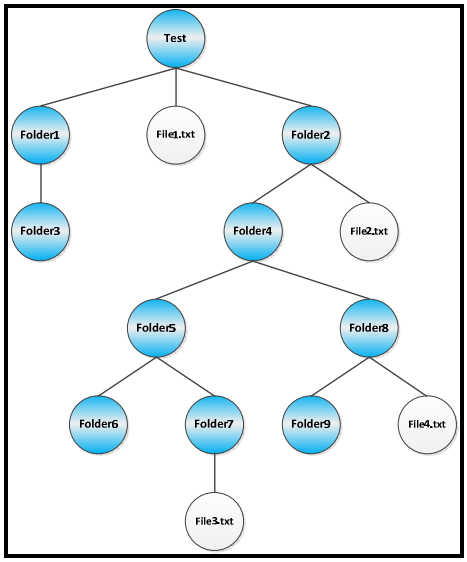
2.1 Folder

Sebuah *folder*, yang nama lainnya adalah direktori, merupakan suatu kontainer yang dapat digunakan untuk menyimpan *file*[5].

Selain *file*, suatu *folder* juga dapat menampung *folder* lain yang disebut dengan *subfolder*. Akibat adanya *folder* dengan *subfolder*-nya ini, dapat terbentuk suatu *tree* dengan seluruh *parent*-nya merupakan *folder*.



Gambar 1 Tampilan Folder pada Windows Explorer



Gambar 2 Contoh Struktur Tree pada Folder "Test"

Pada sistem operasi Linux dan turunan Unix lainnya, segala sesuatu dalam sistem dianggap sebagai *file*, termasuk direktori. Direktori menjadi suatu *file* khusus yang mengandung daftar dari nama-nama *file* beserta *node* masing-masing. Direktori memiliki peran penting dalam sistem *file* yang hierarkis pada sistem operasi komputer modern dengan mengizinkan pengelompokan direktori dan *file* untuk mengatur sistem *file* dalam hierarki yang modular [9].

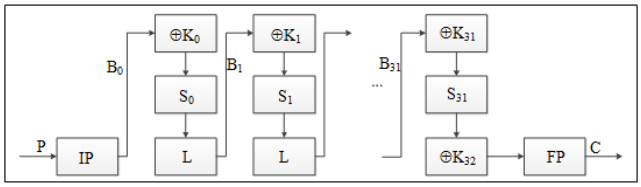
2.2 Serpent

Algoritma Serpent merupakan algoritma blok *cipher* yang didesain oleh Ross Anderson, Eli Biham, dan Lars Knudsen. Algoritma ini merupakan *runner-up* dalam kompetisi Advanced Encryption Standard (AES) yang dimenangkan oleh algoritma Rijndael. Para perancangnya mengklaim meskipun Rijndael lebih cepat karena memiliki putaran lebih sedikit, Serpent lebih aman. Meskipun lebih lambat daripada algoritma AES (Rijndael), algoritma ini masih lebih cepat daripada algoritma DES (Data Encryption Algorithm) yang telah banyak digunakan sebelumnya, yakni berkecepatan rata-rata 45Mbit/s dibandingkan 15Mbit/s pada Pentium 200MHz [1].

Serpent merupakan operasi SP-*network* (*subtitution permutation network*) 32 putaran pada 4 *word* berukuran 32 bit (blok berukuran 128 bit). Pada komputasi internal, semua nilai direpresentasikan *little-endian* dengan *word* pertama adalah *least significant word* dan *word* terakhir adalah *most significant word* dengan bit 0 merupakan *least significant bit* dari *word* pertama. Panjang kunci yang dapat digunakan berukuran 128, 192, atau 256 bit. Jika panjang kunci kurang dari 256 bit, kunci tersebut ditambahkan satu bit '1' yang diikuti bit '0' hingga panjangnya 256 bit.

Dalam enkripsi Serpent terdapat tiga tahap [1]:

1. Permutasi awal (Initial Permutation / IP)
2. Proses 32 putaran, yang masing-masing terdiri atas operasi pencampuran kunci (K_i), proses nilai S-boxes (S_i), dan transformasi linear (L). Pada putaran terakhir, transformasi linear diganti dengan operasi pencampuran kunci
3. Permutasi akhir (Final Permutation / FP)



Gambar 3 Skema Algoritma Serpent

3. ANALISIS

Ada dua hal yang penting yang perlu terdapat dalam aplikasi pengenkripsi *folder*:

1. Keamanan

   Enkripsi tidak hanya dilakukan pada isi *file* saja, melainkan juga struktur *folder*. Jika seseorang yang tidak berhak mengetahui isi suatu *folder* yang terenkripsi dapat menemukan struktur *folder* tersebut, ia bisa saja melakukan analisis terhadap *file* terenkripsi yang ia temukan. Jika ia dapat menemukan cara untuk mendekripsi satu *file* tersebut, ia dapat lebih mudah menemukan cara untuk mendekripsi keseluruhan *file* dalam *folder* tersebut.

2. Kecepatan

   Pengguna tentu menginginkan respon yang cepat dari suatu aplikasi yang dijalankan. Ia tidak ingin menunggu proses lain dari aplikasi tersebut yang tidak diperlukannya. Misalnya jika ia ingin mendekripsi satu *file* saja, ia tidak perlu mendekripsi semua *file* yang terdapat pada *folder* yang terenkripsi karena ia tidak memerlukan *file-file* lain dan tidak ingin menunggu aplikasi memproses pendekripsian keseluruhan isi *folder*.

Penanganan untuk kedua masalah tersebut yaitu:

1. Keamanan

   Untuk melakukan penyembunyian struktur *folder*, seluruh *file* dalam *folder* yang dienkripsi disatukan dalam satu *file*

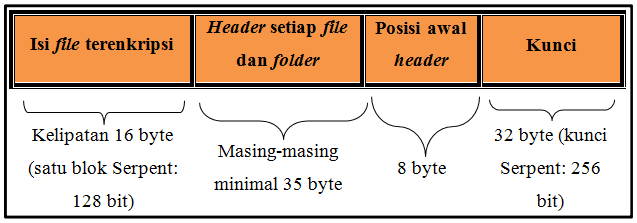
khusus. *File* khusus tersebut mengandung kumpulan *file* yang terenkripsi beserta pengaturan *file* tersebut dalam *folder* yang dienkripsi.

2. Kecepatan

   Untuk mendapatkan *file* atau *folder* yang berada dalam *folder* yang dienkripsi, aplikasi cukup mengakses posisi tertentu pada *file* khusus tersebut. Seperti yang telah disebutkan di atas, *file* khusus tersebut mengandung pengaturan *file-file* yang dienkripsi. Pengaturan tersebut berupa struktur *tree* yang merepresentasikan *folder*.

Solusi yang dibuat merupakan aplikasi pengenkripsi *folder* yang berbasis *desktop* dan berjalan di atas sistem operasi Windows. Aplikasi yang dibuat memiliki beberapa hal utama yang harus dapat dilakukan. Sebagai pengenkripsi *folder*, aplikasi harus mampu melakukan enkripsi sebuah *folder*, baik kosong maupun telah ada isinya, yang dapat diakses dari Windows Explorer. Jika dienkripsi, *folder* tersebut dihapus dan terbentuklah sebuah *file* baru yang merupakan *folder* terenkripsi yang memiliki nama yang sama. Daftar isi *folder* yang telah terenkripsi tersebut dapat dimuat dalam antarmuka aplikasi dengan bentuk menyerupai Windows Explorer. Tentu saja aplikasi tersebut harus mampu mendekripsi *folder* yang telah terenkripsi menjadi *folder* yang dapat diakses dari Windows Explorer. Selain itu, aplikasi harus dapat menambahkan, menghapus, dan mendekripsi sebagian *file* atau *folder* pada *folder* terenkripsi. Ketiga hal tersebut dilakukan aplikasi tanpa harus mendekripsi keseluruhan *folder* terlebih dahulu. Kemampuan ini dilakukan dengan mengimplementasi suatu struktur header khusus yang mewakili masing-masing *file* dan *folder* yang dikandung suatu *folder* terenkripsi.

Aplikasi ini melakukan enkripsi suatu *folder* yang terletak dalam Windows Explorer. Dari *folder* ini dihasilkan suatu *file* baru berekstensi *.dac yang merupakan hasil enkripsi dari seluruh isi *folder* tersebut. Secara otomatis, *folder* tersebut dihapus dari posisi direktori asalnya. Isi *file* hasil enkripsi *folder* tersebut dapat dilihat dengan menggunakan aplikasi ini dengan membaca kumpulan *header* untuk setiap *file* dan *folder* yang terletak dalam *folder* terenkripsi. Struktur *file* terenkripsi dapat ditunjukkan pada Gambar *4*.



Gambar 4 Struktur File *.dac

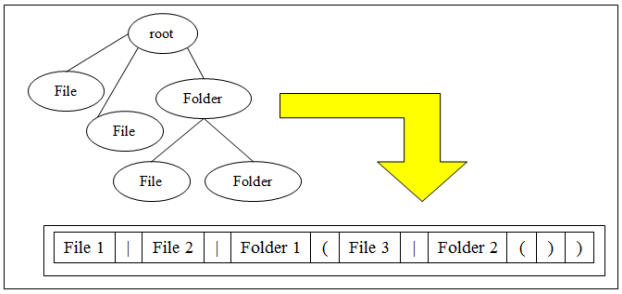
Setiap *header* memiliki beberapa informasi penting mengenai *folder* atau *file* yang terdapat pada *folder* terenkripsi. Selain itu, terdapat pembatas yang menandakan akhir informasi suatu *file* atau *folder*. Pembatas tersebut juga membatasi direktori yang dikandung suatu *folder*. Struktur dari *header* dapat ditunjukkan pada Tabel *1*.

Pembatas '|' menunjukkan akhir dari suatu *file*, sedangkan pembatas '(' dan ')' menunjukkan akhir dari suatu *folder* sekaligus melingkupi *header* untuk setiap *file* dan *folder* yang dikandungnya. Salah satucontoh bentuk *tree* dari *folder* terenkripsi beserta struktur *header*-nya dapat dilihat pada Gambar *5*.

Seperti yang telah dijelaskan sebelumnya, aplikasi ini dapat melakukan enkripsi *folder* dan melakukan dekripsinya. Selain itu, dimungkinkan pula operasi untuk menambahkan *file* atau *folder* atau mendekripsinya secara langsung tanpa harus mendekripsi keseluruhan lalu mengenkripsi ulang keseluruhan *folder*. Untuk melakukannya, *file* berekstensi *.dac yang merupakan *folder* terenkripsi harus dibuka terlebih dahulu dengan menggunakan aplikasi ini. Aplikasi ini akan menampilkan struktur *folder* di dalam *file* *.dac tersebut berdasarkan struktur *header* yang telah dijelaskan di atas. Berikut ini operasi-operasi pokok yang dilakukan oleh aplikasi ini beserta penjelasannya.

Tabel 1 Struktur Setiap Header

Informasi	Ukuran (byte)	Keterangan
Merupakan *file*	1	Bernilai 1 jika merupakan *headerfile*, 0 jika merupakan *header folder*
Ukuran nama	1	Panjang nama *file*/*folder* termasuk ekstensinya
Waktu pembuatan	8	Waktu pembuatan *file*/*folder*
Waktu modifikasi	8	Waktu terakhir modifikasi *file*/*folder*
Offset posisi *file*	8	Posisi *file* yang terenkripsi dalam *folder* terenkripsi
Ukuran	8	Ukuran *file*
Nama	*n*	Nama *file*/*folder* dengan panjang *n* karakter
Pembatas	1 atau 2	Berisi karakter '|' untuk *file*, sedangkan pada *folder* berisi karakter '(' dan ')'



Gambar 5 Contoh Struktur Header Folder Terenkripsi

1. Mengenkripsi *folder*

   Setelah mengetahui *folder* yang akan dienkripsi dan kunci dari pengguna, aplikasi ini melakukan penelusuran setiap *file* dan *folder* yang dikandungnya untuk setiap kedalaman *subfolder*. Pertama-tama, terlebih dahulu dibuat *file* berekstensi *.dac dengan nama sesuai dengan nama *folder* yang dipilih. Jika ketika dilakukan penelusuran ditemukan *file*, *file* tersebut dienkripsi dengan *hash* kunci tadi dan langsung ditambahkan ke dalam *file* *.dac. Informasi dari *file* tersebut dikumpulkan dan disusun dalam memori seperti struktur *header* yang telah dijelaskan sebelumnya. Begitu pula jika ditemukan *folder*. Akan tetapi pada *folder* tidak dilakukan pengenkripsian dan penambahan langsung ke dalam *file* *.dac karena tidak memiliki bentuk seperti *file* yang dapat dienkripsi. Jika penelusuran selesai, *header*

setiap *file* dan *folder* yang telah tersusun tersebut ditambahkan ke dalam *file* *.dac. informasi posisi awal dari kumpulan *header* tersebut ditambahkan pada bagian awal *file* *.dac. Kemudian, hasil *hash* dari kunci ditambahkan di bagian akhir *file* *.dac. Akhirnya, *folder* asli dihapus dan muncul *file* dengan nama yang sama dan berekstensi *.dac.

2. Menampilkan isi *folder* terenkripsi

   Pertama-tama, aplikasi ini mengambil 256 bit terakhir dari *file* *.dac yang merupakan hasil *hash* dari kunci yang sebenarnya dan mencocokkannya dengan *hash* kunci yang dimasukkan pengguna. Kemudian, aplikasi membaca informasi posisi awal kumpulan *header*. Lalu, aplikasi mengambil kumpulan *header* yang berada dari posisi tersebut sampai posisi akhir. Berdasarkan kumpulan *header* tersebut, struktur *tree* dari keseluruhan isi *folder* yang terenkripsi dimuat dalam UI aplikasi. Seperti tampilan *explorer* pada umumnya, aplikasi akan menampilkan isi dari *nodefolder* pada *tree* tersebut yang dipilih oleh pengguna.

3. Mendekripsi *folder*

   Pertama-tama aplikasi akan mencocokkan nilai *hash* kunci dari pengguna dengan *hash* kunci yang terdapat pada bagian akhir *file* *.dac. Aplikasi kemudian mengambil kumpulan *header*, kemudian melakukan dekripsi setiap *file* berdasarkan posisi direktori pada *header*. Setelah melakukan pendekripsian, informasi pada *header* disalin ke dalam setiap *folder* dan *file* yang baru didekripsi tersebut. *File* *.dac tersebut dihapus.

4. Menambahkan *file* atau *folder*

   Untuk melakukan operasi ini, isi *folder* terdekripsi harus sudah ditampilkan pada aplikasi. Setelah *file* atau *folder* yang ingin ditambahkan telah ditentukan, aplikasi mengambil informasi dari setiap *file* atau *folder* tersebut kemudian menambahkannya ke dalam *header* pada posisi direktori yang terbuka pada aplikasi tersebut. Jika yang akan ditambahkan berupa *file*, hasil enkripsinya ditambahkan pada bagian akhir *file* *.dac.

5. Menghapus *file* atau *folder*

   Untuk melakukan operasi ini, isi *folder* terdekripsi juga harus sudah ditampilkan pada aplikasi. Ketika menghapus *file* yang terpilih, aplikasi terlebih dahulu mengambil informasi posisi *file* pada *file* *.dac dan ukuran *file* tersebut, lalu menghapusnya dari *file* *.dac. Header *file* tersebut kemudian dihapus. Ketika menghapus *folder* yang dipilih, aplikasi akan menelusuri seluruh isi *folder* tersebut dan menghapusnya.

6. Mendekripsi *file* atau *folder*

   Untuk melakukan operasi ini, isi *folder* terdekripsi juga harus sudah ditampilkan pada aplikasi. Pertama-tama aplikasi akan meminta konfirmasi kunci dan mencocokkan *hash*-nya. Kemudian aplikasi akan mengambil informasi *header* yang dimiliki oleh *file* atau *folder* yang dipilih. Jika yang akan didekripsi adalah *file*, cipherteks dari *file* tersebut didekripsi ke direktori tujuan. Jika yang akan didekripsi adalah *folder*, aplikasi akan menelusuri seluruh isi *folder* tersebut kemudian mendekripsi *file* yang terkandung di dalamnya dan menyalinnya ke direktori tujuan sesuai posisi direktori masing-masing.

4. PERANCANGAN

Perangkat lunak yang dibuat adalah aplikasi berbasis *desktop* yang dapat mengenkripsi suatu *folder* yang diinginkan beserta seluruh isinya yang berupa *file* dan *folder*. Aplikasi tersebut memungkinkan pengguna untuk mendekripsi *file* atau *folder* tertentu tanpa harus mendekripsi keseluruhan *folder* yang dienkripsi. Untuk mendukung hal tersebut, perangkat lunak ini berupa *graphical user interface* (GUI) yang berisi tampilan *folder* explorer dari *folder* yang terenkripsi tersebut. Perangkat lunak ini bernama DACrypt. Ekstensi *file* hasil pengenkripsian *folder* bernama *.dac. Kebutuhan perangkat lunak ini dibagi dalam dua bagian yaitu kebutuhan fungsional dan nonfungsional.

4.1 Kebutuhan Fungsional

1. Sistem mampu mengenkripsi suatu *folder* beserta isinya yang berupa *file* dan *folder*.
2. Sistem mampu mendekripsi *file* yang merupakan *folder* terenkripsi menjadi *folder* awal.
3. Sistem mampu menampilkan isi *file* yang merupakan *folder* terenkripsi dalam bentuk *explorer*.
4. Sistem mampu menambahkan *file* atau *folder* ke dalam *folder* terenkripsi tanpa harus mendekripsi seluruh isi *folder* terenkripsi terlebih dahulu.
5. Sistem mampu menghapus *file* atau *folder* dalam *folder* terenkripsi tanpa harus mendekripsi seluruh isi *folder* terenkripsi terlebih dahulu.
6. Sistem mampu mendekripsi masing-masing *file* atau *folder* secara terpisah sesuai keinginan pengguna

4.2 Kebutuhan Nonfungsional

1. Sistem mampu menampilkan antarmuka menyerupai Windows Explorer karena pengguna sudah sangat familiar dengan Windows Explorer.
2. Sistem mampu melakukan pengaksesan langsung dari atau ke Windows Explorer untuk melakukan penambahan atau pendekripsian suatu *file* atau *folder* tertentu.

5. IMPLEMENTASI

5.1 Lingkungan Implementasi

Ada dua macam lingkungan implementasi, yaitu pada perangkat keras dan perangkat lunak. Perangkat keras yang digunakan untuk melakukan implementasi adalah sebuah komputer jinjing dengan spesifikasi:

1. Prosesor AMD Turion X2 Dual-Core Mobile RM-75 @2.20Ghz (2 CPU)
2. Memory DDR2 3072MB
3. Hardisk 250GB

Selain itu, perangkat lunak yang digunakan untuk implementasi ini yaitu:

1. Sistem operasi Windows 7 Ultimate 32-bit (6.1, Build 7600)
2. Bahasa pemrograman C#
3. .NET Framework 3.5 SP1
4. Perkakas yang digunakan adalah Microsoft Visual Studio 2008

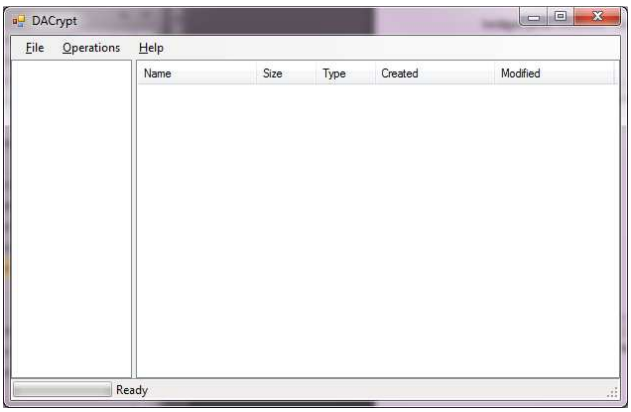
Perangkat lunak ini dibangun sebagai aplikasi *desktop*.

5.2 Batasan Implementasi

Untuk melakukan penyimpanan kunci, digunakan algoritma SHA-256 yang terdapat pada *library* "System.Security.Cryptography" yang sudah terdapat dalam kakas Microsoft Visual Studio 2008. Algoritma ini dapat diakses langsung dari Microsoft Visual Studio. Enkripsi yang dilakukan oleh perangkat lunak ini hanya dapat dilakukan terhadap satu *folder* saja. Perangkat lunak ini tidak melakukan kompresi sehingga ukuran hasil enkripsi tidak lebih kecil dari ukuran sebelum enkripsi. Pengaksesan perangkat lunak melalui *context menu* (menu klik kanan) dari Windows Explorer hanya dilakukan untuk membuka *file* *.dac saja.

5.3 Antarmuka

Antarmuka dalam implementasi ini memanfaatkan Windows Form yang terdapat pada Microsoft Visual Studio. Implementasinya dapat dilihat pada Gambar *6*.



Gambar 6 Implementasi Antarmuka

6. PENGUJIAN

Tujuan pengujian ini antara lain:

1. Memastikan fungsionalitas perangkat lunak dapat berjalan dengan baik sesuai dengan spesifikasi perangkat lunak.
2. Mengetahui keamanan *folder* yang telah dienkripsi terhadap modifikasi isi *folder* terenkripsi tersebut.
3. Mengetahui kinerja perangkat lunak berupa waktu terhadap ukuran *folder* yang akan dienkripsi

Pada hasil pengujian enkripsi, ukuran hasil enkripsi (*file* *.dac) dengan ukuran *folder* semula selalu lebih besar. Hal ini disebabkan oleh adanya bagian *header* pada *file* *.dac yang menyimpan informasi yang dimiliki oleh *file* atau *folder* yang dikandungnya, *byte* penunjuk posisi awal *header*, dan *key* untuk mengakses hasil enkripsi tersebut. Pada *folder* terenkripsi yang kosong tidak terdapat bagian *header* sehingga hanya memiliki ukuran 40 byte yang terdiri atas penunjuk posisi awal *header* (8 byte) dan *key* (32 byte). Setiap *file* yang dikandung oleh *folder* terenkripsi tersebut memiliki ukuran cipherteks yang lebih besar karena di-*padding* dengan beberapa *byte* 0 dan jumlah *byte* 0 tersebut hingga berukuran kelipatan 16 byte. Ukuran setiap *header* untuk *file* adalah 35 byte ditambah dengan jumlah karakter nama *file*, sedangkan untuk *folder* adalah 36 byte ditambah dengan jumlah karakter nama *folder*.

Pada seluruh pengujian dekripsi, setiap proses dekripsi berhasil mengembalikan *folder* dengan isi seperti semula. Hal ini dapat ditunjukkan dengan ukuran *folder* hasilnya sama dengan ukuran *folder* sebelum dienkripsi pada pengujian enkripsi.

Berdasarkan hasil pengujian enkripsi dan dekripsi, diperoleh waktu dekripsi yang lebih cepat daripada proses enkripsi. Hal ini disebabkan oleh penghitungan enkripsi yang melibatkan pengalokasian memori untuk *header* dan penulisan pada *hardisk* untuk cipherteks sedangkan penghitungan dekripsi tidak memperhitungkan kedua hal tersebut. Waktu proses enkripsi dan dekripsi untuk jumlah *folder* dan *file* yang dikandung lebih banyak tentu saja memerlukan waktu lebih lama daripada yang dikandung lebih sedikit.

Pada pengujian penambahan *file* atau *folder*, ukuran *file* *.dac menjadi lebih besar akibat adanya penambahan cipherteks *file* sebesar kelipatan 16 byte dan *header* untuk setiap *file* dan *folder* yang ditambahkan. Begitu pula pada pengujian penghapusan *file* atau *folder* yang mengurangi ukuran *file* *.dac.

Pada pengujian dekripsi *file* atau *folder* yang dikandung *folder* terenkripsi, dapat diperoleh hasil dekripsi yang sesuai dengan *file* atau *folder* semula. Ukuran *file* hasil dekripsi tentu saja lebih kecil daripada ukuran yang tertera pada GUI karena ada penghapusan penambahan byte 0 yang sebelumnya digunakan untuk proses enkripsi.

Pada pengujian pembukaan *folder* terenkripsi, jika *password* yang dimasukkan oleh pengguna tidak cocok dengan *key* yang terdapat pada *file* *.dac, *file* tidak dapat dibuka. *Password* cocok dengan *key* jika hasil *hashpassword* dengan SHA-256 sama dengan *key*. Modifikasi pada *key* yang terletak pada 32 byte akhir *file* *.dac menyebabkan *file* tersebut tidak dapat dibuka, kecuali jika kebetulan *password* yang dimasukkan cocok dengan *key* dan tidak ada perubahan ukuran *file* *.dac. Jika ada penghapusan beberapa *byte* pada *file* *.dac, dapat terjadi gagalnya pembukaan atau pemuatan *folder* terdekripsi pada GUI. Jika terjadi pengubahan isi pada bagian cipherteks, *file* yang terkandung dalam *folder* terenkripsi masih dapat diakses melalui GUI. Akan tetapi, ketika didekripsi, hasilnya akan berbeda dengan yang seharusnya. Selain itu, proses dekripsi terhadap *file* tersebut juga dapat gagal akibat kesalahan penghapusan jumlah bit 0 yang telah di-*padding* pada *file* tersebut sebelum dienkripsi.

7. KESIMPULAN

1. Dalam perangkat lunak diperlukan suatu struktur *header* khusus yang selalu diakses ketika akan menambahkan (mengenkripsi) suatu *file* atau *folder* sehingga tidak perlu melakukan dekripsi terhadap keseluruhan isi *folder*.
2. Selain itu, struktur *header* tersebut diakses untuk mengambil (mendekripsi) atau menghapus suatu *file* atau *folder* sehingga tidak perlu melakukan dekripsi terhadap keseluruhan isi *folder*.
3. Perangkat lunak yang dibuat dapat melakukan enkripsi *folder* dengan menggabungkan seluruh isi *folder* dalam satu *file* sehingga tidak memungkinkan pengaksesan isi dan struktur *folder* dari Windows Explorer.
4. Aplikasi dibuat dengan memanfaatkan algoritma Serpent dengan antarmuka perangkat lunak mirip dengan Windows Explorer dalam menampilkan struktur *folder* berupa *tree* dan menampilkan isi *folder* yang diakses.
5. Perubahan pada *folder* terenkripsi berupa pengubahan atau penghapusan sebagian bit isinya dapat mengakibatkan *folder* tersebut tidak dapat diakses.
6. Jumlah *file* atau *folder* yang dikandung dalam suatu *folder*

yang akan dienkripsi mempengaruhi waktu enkripsi itu sendiri. Begitu pula untuk proses dekripsi.

7. Hasil enkripsi *folder* dengan aplikasi yang dibuat selalu memiliki ukuran yang lebih besar daripada ukuran semula. Hal ini disebabkan oleh kebutuhan enkripsi algoritma Serpent yang melakukan enkripsi setiap blok berukuran 128 bit (16 byte). Ukuran yang lebih besar tersebut juga diakibatkan oleh adanya kebutuhan aplikasi berupa informasi struktur *header*, posisi awal *header*, dan kunci.

8. Jika ada penyerang yang dapat mengetahui struktur *file**.dac dan algoritma yang digunakan, faktor keamanan tinggal bergantung pada algoritma Serpent.

8. REFERENSI

[1] Anderson, Ross, Eli Biham, dan Lars Knudsen. (1998). Serpent: A Proposal for the Advanced Encryption Standard.

[2] Axantum Software AB. (2010). AxCrypt File Encryption for Windows. http://www.axantum.com/AxCrypt. Waktu akses: 12 Oktober 2010 06:24

[3] KaKaSoft. (2006). Folder Protector. http://www.kakasoft.com/lock/index.htm. Waktu akses 12 Oktober 2010 06:05

[4] Munir, Rinaldi. (2005). Diktat Kuliah IF5054 Kriptografi. Bandung: Departemen Teknik Informatika Institut Teknologi Bandung.

[5] Microsoft Corporation. (2010). Working With Files and Folders. http://windows.microsoft.com/en-US/windows7/Working-with-files-and-folders. Waktu akses: 3 November 2010 21:48

[6] NewSoftwares.net. (1998). Folder Lock® - The Fastest Encryption on Earth. http://www.newsoftwares.net/folderlock. Waktu akses: 12 Oktober 2010 06:23

[7] Ninan, Subin. (2010). Secure Folder. http://subininan.blogspot.com/2010/09/secure-folder_21.html. Waktu akses: 12 Oktober 2010 06:12

[8] Scneier, Bruce dan Doug Whiting. (2000). A Performance Comparison of the Five AES Finalist.

[9] The Linux Information Project. (2006). Directory Definition. http://www.linfo.org/directory.html. Waktu akses: 3 November 2010 22:21C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.