Desain dan Implementasi Kriptografi di *File System*
Studi Kasus: Penggunaan Criptographic File System pada Unix

Pandu Pradana[1] dan Syaikhuddin[2]

Departemen Teknik Informatika
Institut Teknologi Bandung
Jalan Ganesha 10 Bandung 40132

Email: *if12010@students.if.itb.ac.id*[1], *if12018@students.if.itb.ac.id*[2]

Abstrak

Keamanan secara total dan mutlak sangat sulit untuk dicapai. Salah satu hal untuk mengatasi masalah ini adalah dengan membuat sistem pengamanan berlapis. Dengan kemajuan teknologi, pengamanan sudah dapat dilakukan pada tingkat file system. Ada beberapa pertimbangan mengapa pengamanan dianggap lebih baik dilakukan pada tingkat file system, bukan pada tingkat pengguna atau tingkat yang lebih rendah. Kriptografi pada tingkat file system memiliki tujuan tertentu, sehingga untuk memenuhi tujuan tersebut, dibuatlah suatu arsitektur sistem kriptografi spesifik, dalam hal ini dibahas mengenai *Cryptographic File System* (CFS) pada Unix. Namun sistem ini belum tentu menghasilkan suatu perlindungan yang mutlak bagi sistem. Masih diperlukan pengembangan dan perbaikan lagi dalam penerapannya di masa mendatang. Namun makalah ini memberikan suatu gambaran mengenai ide tentang penerapan teknik kriptografi pada tingkat *file system*.

Kata kunci: kriptografi, file system, CFS

1. Pendahuluan

Keamanan sudah menjadi bagian yang penting dalam teknologi informasi. Perlindungan data merupakan sesuatu yang mahal dan sulit untuk diterapkan secara sempurna/total. Salah satu pendekatan yang dilakukan dalam perlindungan data adalah perlindungan secara berlapis. Kriptografi menyediakan sebuah solusi yang menjanjikan untuk hal demikian. Dengan semakin majunya teknologi, sekarang metode pengamanan sudah dapat diterapkan pada tingkat *file system*. Sudah banyak aplikasi yang dikembangkan sehubungan dengan perlindungan data. Namun, sering kali aplikasi yang dikembangkan sehubungan dengan enkripsi data tidaklah dikembangkan secara terintegrasi dengan sistem, sehingga penanganan keamanan data yang mengandalkan aplikasi yang berjalan di *user level* sangatlah beresiko. Atas dasar inilah lahir sebuah pemikiran bahwa penanganan keamanan harus berjalan terintegrasi pada *file system*. Makalah ini membahas penerapan teknik kriptografi pada *file system*.

1.1 Pendekatan Kriptografi pada F*ile System*

Terdapat beberapa pendekatan yang berhubungan dengan kriptografi pada file system. Pendekatan-pendekatan ini memiliki keuntungan dan kerugiannya masing-masing. Beberapa pendekatan itu adalah:

- *Volume encryptors*
  Volume encryptors menggunakan layer *device driver* untuk melakukan enkripsi dan dekripsi informasi dari dan ke sebuah disk fisik. *Volume encryptors* melakukan

enkripsi secara utuh terhadap *drive* yang ada dan transparan bagi pengguna akhir. Namun, *volume encryptors* tidak menyediakan kendali akses yang cukup baik untuk file atau direktori secara individu.

- *File encryptors*
  File encryptors bekerja pada *application layer* atau *presentation layer* untuk menyediakan sebuah enkripsi file yang bersifat end-to-end. Untuk sejumlah kecil file, tipe enkripsi ini sudah mencukupi, namun belum mencukupi untuk *storage system*.
- *File system encryptors*
  File system encryptors dapat digunakan untuk melakukan enkripsi terhadap file dengan basis file per file atau direktori per direktori.[4)]

Sementara, beberapa hal yang menjadi pertimbangan penerapan enkripsi pada file system adalah

1.2 *User-Level Cryptography*

Pendekatan paling mudah yang dapat digunakan untuk melakukan enkripsi adalah dengan menggunakan *tool*. *Tool* ini bekerja di level pengguna yang tentunya membutuhkan kunci enkripsi dari pengguna. Enkripsi dan dekripsi berada di bawah kendali pengguna. Kebijakan untuk melakukan enkripsi tergantung pada tool tersebut. Ada yang langsung menghapus *plaintext* setelah melakukan enkripsi dan ada yang tidak.

Pendekatan lain adalah dengan mengintegrasikan algoritma enkripsi pada suatu perangkat lunak. Dengan demikian, jika data yang dimanipulasi tersebut bersifat penting, data tersebut bisa langsung dienkripsi dengan perangkat lunak tersebut. Sebagai contoh, teks editor yang di dalamnya terintegrasi *tool* untuk melakukan enkripsi dan dekripsi data. Ketika teks editor tersebut akan menyimpan file teks, secara otomatis pengguna diminta untuk memasukkan kunci karena file tersebut akan dienkripsi. Begitu juga sebaliknya saat membuka file.

Sayangnya, pendekatan tersebut memiliki kelemahan baik dari segi keamanan maupun kenyamanan. Pada pendekatan pertama, pengguna mungkin secara tidak sengaja melakukan kesalahan dalam enkripsi, misalnya lupa untuk menghapus *clear-text* dari file tersebut setelah dienkripsi. Program enkripsi/dekripsi selalu meminta pengguna untuk memasukkan kunci. Hal ini bisa membuat pengguna merasa tidak nyaman.

Pada pendekatan kedua, tiap program harus memiliki fungsionalitas enkripsi *built-in*. Meskipun enkripsi/dekripsi dapat dilakukan secara otomatis, pengguna harus tetap menyediakan kunci. Terkadang kesalahan aplikasi (pada saat implementasi) akan sangat sulit dideteksi. Misalnya pada aplikasi *vi* yang mampu melakukan enkripsi di beberapa versi Unix masih meninggalkan *clear-text* dalam data *temporary*. Selain itu, perubahan pada algoritma enkripsi menyebabkan perubahan pada program-program yang menggunakan algoritma tersebut.

1.3 *System-Level Cryptography*

Salah satu cara yang digunakan untuk mengatasi kelemahan pada *user-level cryptography* adalah dengan menerapkan kriptografi pada level sistem sebagai *service* dasarnya. Dalam mendesain sistem semacam ini, hal yang perlu diperhatikan adalah identifikasi data/file yang memerlukan proteksi dan yang tidak (*clear-text*). Dengan kata lain, harus diketahui komponen sistem mana yang bersifat sensitif.

Untuk file, metode pengamanan yang biasa digunakan adalah pengamanan secara fisik, yaitu memproteksi media penyimpanan yang digunakan untuk menyimpan data sensitif/penting. Hal ini bisa dilakukan dengan menggunakan *disk controller* yang di dalamnya terdapat perangkat enkripsi terintegrasi. Dengan alat tersebut, enkripsi -- terhadap seluruh isi disk atau sebagian-- bisa dilakukan dengan kunci enkripsi tertentu. Akan tetapi, metode ini memiliki kelemahan. Karena kunci enkripsi berbeda antara media yang satu dengan yang lain, akan terjadi kesulitan jika ingin melakukan *resource sharing* antara beberapa pengguna. Permasalahan juga terjadi pada saat melakukan *backup*. Jika data *backup* diambil dari disk tanpa terdekripsi dan *disk controller* tidak tersedia, data tidak dapat digunakan. Selain itu, pendekatan semacam ini tidak memproteksi data yang keluar dari dan masuk ke dalam *disk controller*, sehinga kurang aman jika digunakan untuk memproteksi data di *remote file server*.

Pada sistem terdistribusi, proteksi sering dilakukan pada koneksi jaringan antara *server* dan *client.* Proteksi tersebut dilakukan dengan menggunakan enkripsi *end-to-end* dan autentikasi. Proteksi juga bisa dilakukan dengan perangkat keras atau perangkat lunak khusus. Akan tetapi, tidak semua jaringan mendukung enkripsi/dekripsi, dan meskipun mendukung, pasti ada perbedaan algoritma yang diterapkan.

2. Kriptografi pada *File System*

Berdasarkan deskripsi mengenai kelebihan dan kekurangan dari kriptografi pada tingkat user ataupun sistem yang telah dijelaskan, sebuah implementasi dari kriptografi pada file system ini, yaitu *Cryptographic File System* (selanjutnya disebut CFS) di desain untuk berada di antara tingkat pengguna (*user-level cryptography*) dan tingkat yang rendah (*system-level cryptography*).

2.1 Desain

CFS dibuat untuk memberikan proteksi terhadap *file storage* yang mudah untuk diserang dengan suatu cara yang nyaman untuk digunakan sehari-hari. Implementasi CFS tidak boleh membebani sistem secara keseluruhan. Secara khusus, CFS dibuat dengan kriteria sebagai berikut:

- Manajemen kunci yang rasional. Untuk menggunakan data yang diproteksi, tentunya memerlukan kunci. Oleh karena itu, diperlukan manajemen kunci. Namun, hal ini tidak boleh mengganggu sistem, kunci enkripsitidak perlu dimasukkan lebih dari satu kali per sesi. Sekali kunci dimasukkan dan diautentikasi, *user* tidak perlu memasukkan kunci tersebut pada operasi berikutnya pada sesi yang sama. Di samping itu, sistem harus menyediakan suatu mekanisme untuk melepaskan kunci dari sistem setelah sesi berakhir atau *user* tidak aktif.
- Semantik pengaksesan yang transparan. File yang terenkripsi harus berlaku seperti halnya file biasa, kecuali dalam hal penggunaan kunci. File terenkripsi harus mendukung metode akses (*access methods*) yang sama dengan *storage system* yang mendasarinya. Semua *system call* harus bekerja secara normal.
- Performa yang transparan. Meskipun beberapa algoritma kriptografi memiliki komputasi yang cukup rumit dan lama, penalti pada performa yang berkaitan dengan operasi file terenkripsi tidak boleh terlalu tinggi.

- Proteksi terhadap isi file. Data dalam file harus diproteksi termasuk struktur data yang berkaitan dengan isi file.
- Proteksi terhadap meta-data sensitif. Informasi yang berkaitan dengan file terenkripsi harus diproteksi, misalnya nama file.
- Proteksi terhadap koneksi jaringan. Tidak ada informasi dalam file terenkripsi yang dapat dilihat dalam lalu lintas jaringan.
- Pengelompokan kunci. Pengelompokan terhadap apa yang diproteksi dengan kunci yang sama harus mencerminkan struktur konstruksi sistem yang mendasarinya, misalnya struktur direktori. Harus ada kemudahan dalam melakukan proteksi terhadap file-file yang berkaitan dengan menggunakan kunci yang sama dan kemudahan untuk membuat kunci baru untuk file-file lain.
- Kompatibilitas dengan servis yang ada pada sistem. File dan direktori yang terenkripsi harus disimpan dan dikelola seperti halnya file yang lain. Harus ada mekanisme yang memungkinkan untuk melakukan *backup* data terenkripsi tanpa *tool* khusus atau mengetahui kunci.
- Portabilitas. Sistem enkripsi harus menggunakan *interface* sistem yang ada dan tidak bergantung pada fitur khusus tertentu.
- Skala. Mesin enkripsi tidak memberikan beban tambahan yang berlebihan pada *shared component* sistem. File server secara khusus tidak perlu melakukan pemrosesan tambahan untuk *client* yang memerlukan proteksi dengan kriptografi.
- Akses yang konkuren. Pengaksesan file terenkripsi secara konkuren oleh beberapa *user* harus dimungkinkan.
- Kepercayaan terbatas. Umumnya, *user* harus percaya pada komponen-komponen yang berada di bawah kendalinya.
- Kompatibilitas dengan teknologi yang aka datang. Teknologi yang akan datang memiliki aplikasi yang potensial memerlukan proteksi data. Misalnya, koleksi kunci dikumpulkan pada *smart card* yang hanya dimiliki oleh *authorized user*. Sistem enkripsi harus mendukung hal tersebut tanpa tambahan hardware tertentu selain pembaca *smart card*. [1)]

3. Sistem Enkripsi

CFS menggunakan DES untuk mengenkripsi data. Namun DES saja tidak cukup. Hal ini dikarenakan antara lain oleh panjang kunci yang hanya 56 bit, dan penghitungan DES yang sulit jika diterapkan pada perangkat lunak (*costly*), sehingga bila diterapkan pada suatu aplikasi *file system*, tampaknya akan menimbulkan kelambatan.

Untuk memperbolehkan akses acak terhadap file, CFS melakukan enkripsi terhadap isi file dengan dua cara. Asumsikan kunci pada CFS berupa "*passphrases*". Ketika kunci ini muncul pada waktu attach, kunci ini dibagi menjadi dua bagian kunci DES 56 bit. Kunci pertama digunakan untuk menghitung *pseudo-random bit mask* dengan mode *output feed back* (OFB) pada DES. Mask ini kemudian disimpan untuk dipergunakan dalam proses *attach*. Ketika sebuah blok file hendak ditulis, pertama blok tersebut di-XOR-kan dengan bagian dari mask yang bersesuaian dengan *byte offset* dari modulo file hasil perhitungan *mask length* sebelumnya. Hasilnya kemudian dienkripsi dengan kunci kedua menggunakan mode ECB. Ketika dibaca, cipher dibalik dalam urutan tertentu: pertama didekripsi dengan ECB, kemudian di-XOR-kan dengan *positional mask*. Kombinasi ini menjamin bahwa blok yang identik akan dienkripsi

secara berbeda pada *ciphertext* bergantung kepada posisinya dalam file.

Untuk mematahkan analisis terhadap blok yang sama di posisi yang sama pada file yang berbeda, setiap file yang dienkripsi dengan kunci yang sama dapat menggunakan sebuah *initial vector* (IV) yang unik. IV dapat dibuat secara otomatis, misalnya dari waktu pembuatan file. Namun untuk dapat melakukan dekripsi, IV perlu disimpan juga bersama dalam file. Oleh karena IV harus disimpan bersama dalam file untuk proses dekripsi, maka terdapat beberapa solusi, pertama adalah menempatkan IV di awal file, namun hal ini akan menggeser isi file dari batasan block, dan jika ditambahkan padding bit, maka akan menyita tempat. Pilihan kedua untuk menyimpan IV adalah inode sendiri bersama dengan atribut file lainnya. Namun, karena CFS berada pada *file system* dan menggunakan *system call* untuk operasi I/O, maka CFS tidak memiliki akses langsung ke inode sehingga tidak dapat menambahkan atribut file yang baru.

Oleh karena itu, CFS menyediakan dua mode enkripsi, pada mode standar, tidak terdapat IV yang digunakan sehingga file dapat digunakan untuk analisa blok data yang identik. Pada mode sekuriti tinggi, IV disimpan dalam *group id* (`gid`) dari inode setiap file.

4. Arsitektur Cryptographic File System

Prototipe CFS diterapkan sepenuhnya pada level pengguna dan berkomunikasi dengan kernel UNIX melalui antarmuka NFS (*Network File System*). Setiap mesin *client* menjalankan server NFS yang khusus, yang bernama `cfsd` (CFS Daemon) pada antarmuka *localhost* yang meng-interpretasikan *request* terhadap CFS.

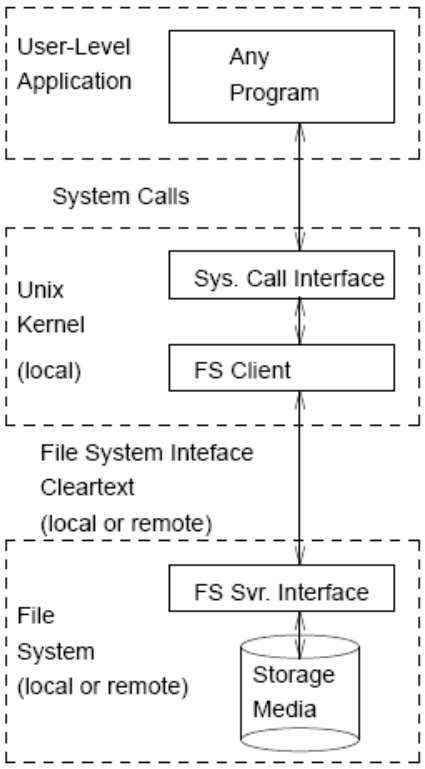
Protokol NFS ditujukan untuk *remote file server*, sehingga diasumsikan bahwa *file system* sangat *loosely coupled* dengan *client*. Kernel pada *client* berkomunikasi dengan *file system* melalui *remote procedure calls* (RPCs). *Server* bersifat *stateless*, yaitu tidak menyimpan *state* dari *system call client*.

NFS pada *client* melakukan *caching* blok file untuk meningkatkan performansi. Semua komunikasi me-rupakan diinisiasi oleh *client*, dan *server* akan memproses masing-masing *system call*, dan menunggu *system call* berikutnya.

`cfsd` diimplementasikan sebagai RPC server untuk sebuah protokol NFS yang lebih luas. Pada awalnya, root dari file system CFS muncul sebagai sebuah direktori kosong. Perintah `cattach` mengirimkan sebuah RPC kepada `cfsd` dengan argumen mengandung *full path name* dari sebuah direktori, nama dari "*attach point*", dan kuncinya. Jika kuncinya benar, maka `cfsd` menghitung *cryptographic mask* dan membuat sebuah entry pada direktori root yang bersesuaian dengan *attach point name*. Entry pada *attach point* ditampilkan sebagai sebuah direktori yang dimiliki oleh pengguna yang melakukan *request attach* dengan mode proteksi 700 (untuk mencegah pengguna lain melihat isinya).

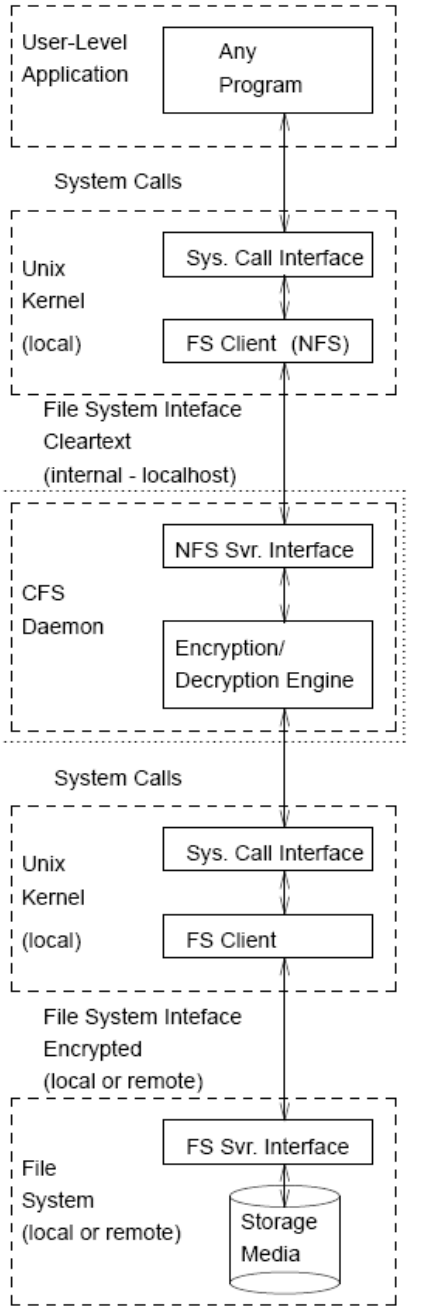
Untuk sebuah file terenkripsi yang diakses melalui *attach point*, `cfsd` membuat sebuah *file handle* yang unik untuk mengidentifikasi file yang ber-sangkutan. Untuk setiap *attach point*, CFS daemon menyimpan sebuah tabel yang berisi *file handler* yang ada bersamaan dengan nama file bersangkutan yang sudah terenkripsi. Ketika ada operasi terhadap file yang bersangkutan *file handle* akan digunakan sebagai indeks dari tabel. `cfsd` menggunakan *system call* standar Unix untuk melakukan pembacaan dan penulisan

terhadap file, yang dienkripsi sebelum penulisan, dan didekripsi setelah dibaca.



Gambar 1 Aliran Data pada Vnode File System Standar

Gambar 1 menjelaskan antarmuka antara aplikasi dan unix *file system* berbasis vnode milik Sun. Setiap tanda panah antar kotak menggambarkan pergerakan data pada kernel, perangkat keras, atau *network boundaries*. Diagram ini menggambarkan data yang ditulis dari aplikasi pertama kali disalin ke kernel kemudian ke *file system* (lokal atau *remote*). Gambar 2 menjelaskan arsitektur dari prototipe CFS pada *user level*. Data disalin beberapa kali dari aplikasi ke kernel, kemudian ke cfs daemon, kemudian kembali ke kernel, dan akhirnya ke *file system*.



Gambar 2 Aliran Data pada Prototipe CFS

5. Kelebihan dan Kekurangan

Kelebihan CFS adalah mampunya mengurangi atau mencegah beberapa ma-cam kesalahan:

- Setelah bekerja dengan *clear-text files*, CFS tidak memerlukan langkah enkripsi ulang yang terpisah, sehingga menghindari masalah ketika mela-kukan enkripsi ulang dengan kunci yang berbeda.
- Kontrol terhadap perubahan lebih kecil kemungkinannya mengalami kesalahan.
- CFS memperbolehkan adanya timeout, sehingga *clear-text file* tidak dibiarkan terlalu lama dapat diakses.

Kekurangan yang harus dipertim-bangkan ketika menggunakan CFS dan perangkat privasi lain adalah:

- Proses *keyboard snooping/keystroke logging* dapat mengekspos kunci rahasia ketika pengguna mengetik kunci untuk enkripsi atau dekripsi
- Pengguna yang memiliki hak akses dapat melihat *clear-text files* yang sedang di-*attach* dengan cara yang beragam.
- *Clear-text files* milik pengguna dapat diekspos pada jaringan dalam berbagai cara.
- Pertimbangkan untuk menyimpan beberapa *file* pribadi dalam beberapa direktori pribadi yang berbeda.
- Ketika menggunakan perangkat *revision control* terhadap *file* pribadi, perlu dipikirkan cara agar *clear-text files* hanya terekspos sementara.[3)]
- Masalah performansi juga salah satu kekurangan dari CSF. Karena berjalan pada *user level*, maka akan terjadi banyak *context switch* dan penyalinan data antara *user space* dan *kernel space*.[2)]

6. Kesimpulan

CFS menyediakan sebuah mekanisme sederhana untuk melindungi data yang ditulis ke disk dan dikirim ke jaringan file server. Antarmuka file system pada client tampaknya merupakan tempat yang tepat untuk melakukan proteksi data, bila dibandingkan dengan enkripsi pada *application layer*. Namun, hanya CSF sendiri saja belum tentu dapat mengamankan data, dukungan untuk *hardware security token* (misalnya dari OpenSSL) juga dapat digunakan untuk mencegah *keyboard sniffing*.[3)]

7. Referensi

[1] Matt Blaze, "*A Cryptographic File System for Unix*", In *Proceedings of the first ACM Conference on Computer and Communications Security*, 1993.

[2] Charles P. Wright et al, "*Cryptographic File Systems Performance: What You Don't Know Can Hurt You*", Stony Brook University, IEEE SISW 2003

[3] Jerry Sweet, "*Using CFS, the Cryptographic Filesystem*", http://www.linuxjournal.com/article/6381.html, diakses tanggal: 1 Januari 2006

[4] Ido Dubrawsky, "*Cryptographic Filesystems*", http://www.linuxexposed.com/Articles/General/Cryptographic-Filesystems.html, diakses tanggal: 1 Januari 2006