

DAFTAR ISI

| | halaman |
|--|---------|
| Daftar Isi | i |
| Daftar Gambar | ii |
| Abstrak | iii |
| BAB I PENDAHULUAN | 1 |
| BAB II DASAR TEORI | 3 |
| 2.1 Network File Sistem (NFS) | 3 |
| 2.2 Key Management | 4 |
| 2.3 <i>Self-Certifying File System (SFS)</i> | 5 |
| BAB III PERANCANGAN SELF-CERTIFYING FILE SYSTEM | 9 |
| 3.1 Sasaran/Tujuan SFS | 9 |
| 3.2 <i>Self-Certifying pathnames.</i> | 9 |
| 3.3 Direktori / SFS | 11 |
| 3.4 Server key management | 12 |
| 3.5 <i>User Authentication</i> | 14 |
| 3.5.1 <i>sfsagent</i> and <i>authserv</i> | 15 |
| 3.5.2 User key management | 16 |
| 3.6 Revocation | 17 |
| BAB IV HASIL PERANCANGAN DAN IMPLEMENTASI | 19 |
| 4.1 Implementasi | 19 |
| 4.2 Cryptographic protocols | 20 |
| 4.3 Key negotiation | 20 |
| 4.4 User authentication | 21 |
| 4.4 Cryptography | 23 |
| BAB IV KESIMPULAN | 25 |

DAFTAR PUSTAKA

26

DAFTAR GAMBAR

| | halaman |
|--|---------|
| Gambar 2.1 Arsitektur Network File System | 3 |
| Gambar 2.2 Proses Remote Mount | 4 |
| Gambar 2.3 Sistem file terdistribusi | 6 |
| Gambar 2.4 Arsitektur SFS | 8 |
| Gambar 3.1 Self-Certifying Pathnames | 10 |
| Gambar 4.1 Bagian-bagian Sistem SFS | 19 |
| Gambar 4.2. SFS key negotiation protocol | 21 |
| Gambar 4.3 <i>SFS user authentication protocol</i> | 23 |

Abstrak

Penggunaan *Network File System* (NFS) tidak akan menjamin pengamanan file dengan berkembangnya internet. Dengan adanya *Key management* maka akan mampu mengamankan sistem secara *online* untuk skala global seperti halnya internet. Dengan berbagai jenis file yang terdapat pada internet, ada suatu mekanisme sistem file untuk mengatur *keys* yang akan mendukung penggunaan berbagai macam tipe file. Dengan memisahkan *key management* dari sistem file *security*, maka *file system* yang bersifat global dan dapat diakses secara bersamaan dapat diatur secara individu tanpa harus dilakukan oleh administrator. Dengan adanya *Self-certifying File System* (SFS) maka akan menjamin keamanan file *system* yang dapat menghindari *internal key management*. Sementara sistem file lain memerlukan *key management* untuk memetakan nama file ke *encryption keys*, SFS akan memberi nama file secara efektif yang berisi *public keys*, dengan membuat *self-certifying pathnames*. *Key management* dalam SFS terjadi diluar file system, sehingga user akan memilih prosedur yang digunakan untuk menghasilkan nama file.

Self-certifying pathnames akan membebaskan SFS *client* dari masalah administrasi. Administrator akan membiarkan para *user* untuk membuktikan keaslian (*users authenticate*) server melalui beberapa teknik yang berbeda. *File namespace* akan menggandakan *key certification namespace* sehingga user akan mendapatkan banyak bentuk *key management*. Akhirnya dengan digunakannya *self-certifying pathnames*, user dapat mem-*bootstrap* suatu mekanisme lain selain *key management*. Dengan adanya SFS, maka akan lebih praktis dan serbaguna daripada file system yang dibangun dengan *key management*.

BAB I

PENDAHULUAN

Network file system akan menjamin file sistem berdasarkan pada sistem jaringan global dan dirancang untuk para pengguna internet dalam melakukan aplikasinya. Saat ini sangat sedikit orang mengerti penggunaan *network file system* untuk mengamankan file-nya dalam pengiriman data melalui internet, bukti-bukti telah banyak bahwa para *attackers* dapat dengan mudah merusak lalu lintas jaringan global (internet), sehingga perlu adanya suatu sistem yang dapat mencegah hal-hal tersebut.

Dengan digunakannya sistem *key management* dan *self-certifying file system* (SFS), maka penanganan keamanan *file system* dalam suatu sistem jaringan global (internet) dapat lebih terjamin. Sistem SFS menggunakan suatu pendekatan baru untuk sistem *file security* yaitu dengan memindahkan *key management* dari sistem file seluruhnya. SFS ini akan memperkenalkan *self-certifying pathnames* yaitu nama file yang secara efektif berisi kunci publik *server remote* yang sesuai. Hal ini disebabkan karena *self-certifying pathnames* telah menetapkan suatu kunci publik, dan SFS tidak memerlukan *key management* yang terpisah untuk mengkomunikasikan *file server* secara aman.

SFS selanjutnya akan melakukan *user authentication* dari *file system* melalui suatu arsitektur modular. Program eksternal digunakan untuk membuktikan keaslian para pemakai dengan suatu protocol yang tidak dapat di tembus ke dalam perangkat lunak *file system*-nya. Program ini akan berkomunikasi dengan perangkat lunak file sistem melalui suatu alat penghubung (*interface*) yaitu *Remote Procedure Call* (RPC). Sehingga para pemrogram dapat dengan mudah menggantinya tanpa menyentuh inti *file system*.

Dalam tulisan ini akan diuraikan beberapa teknik *key management* yang dibangun dalam SFS. Ada dua hal penting yang diperlukan dalam hal ini yaitu *certification authorities* and *password authentication*. Tanpa pendekatan *key management*, SFS juga menyediakan suatu infrastruktur *key management* yang

sangat besar. Sedangkan *Symbolic links* akan menempatkan *human-readable names* ke dalam *self-certifying pathnames*. Sehingga *namespace* global SFS berfungsi sebagai *key certification namespace*.

SFS adalah suatu perangkat lunak *freeware* dan dapat dijalankan dalam sistem operasi UNIX dan menggunakan NFS versi 3. Selain itu dapat juga berjalan dalam sistem operasi OpenBSD, FreeBSD, MacOSX, OSF/1, Solaris, and sebagian distribusi Linux.

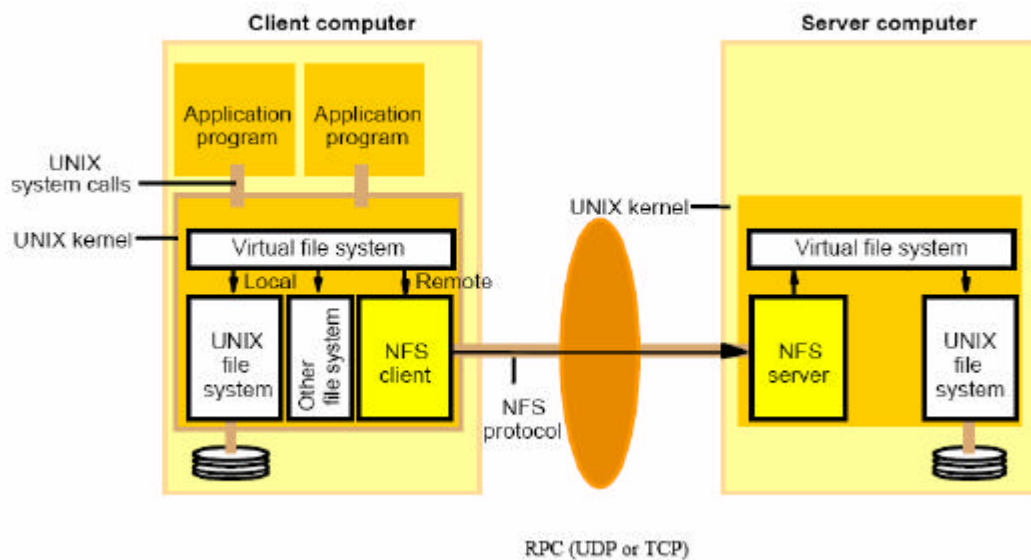
BAB II

DASAR TEORI

2.1 Network File System (NFS)

Dalam lingkungan kerja yang berbasis jaringan atau *local area network*, pemakaian file secara bersama-sama sangat diperlukan. Hal ini dilakukan untuk membuat file baru, mengunci file, dan mengatur kepemilikan file dengan tepat. Beberapa perusahaan telah melakukan beberapa riset untuk mengembangkan *network file system*, diantaranya Apollo Domain mengembangkan *Andrew File system* (AFS), AT&T mengembangkan *Remote File System* dan Sun Microsystem's mengembangkan *Network File System* (NFS). Masing-masing vendor mempunyai kelemahan dan kelebihan.

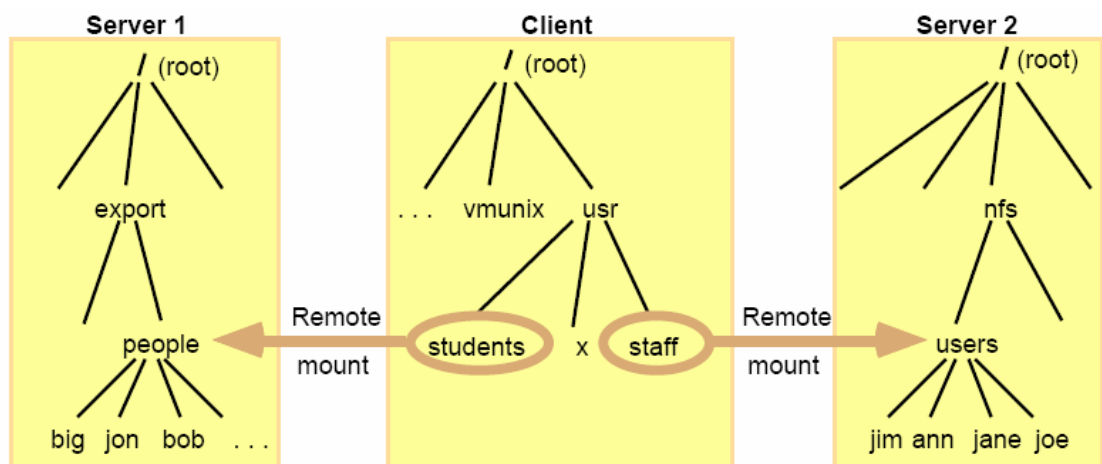
Dari semua file system yang ada Network File System (NFS) digunakan hampir secara luas. NFS tersedia hampir di semua versi UNIX, seperti Apple Machintosh System, MS-DOS, Windows, OS/2, dan VMS. NFS terus berkembang sampai dengan versi 3 yang lebih luas jangkauan keamanannya. Gambar 1 menggambarkan arsitektur network file system.



Gambar 2.1 Arsitektur Network File System

NFS *protocol* dirancang dan diimplementasikan dengan menyediakan pengaksesan menggunakan *remote* untuk file-file yang digunakan secara bersama-sama. Hal ini digunakan oleh komputer yang berbasis *client-server*, dimana *client* memasukan file dari komputer lain dan server *local* akan mengirim file sistem ke komputer tersebut.

NFS *protocol* menggunakan *Sun's Remote Procedure Call (RPC)* protokol yang membolehkan program-program berjalan di suatu komputer dengan memanggil *subroutine* yang akan dieksekusi di komputer lain, yang merepresentasikan eksternal data, sehingga adanya pertukaran informasi dalam komputer yang berbeda. Hal ini dapat berjalan di atas *TCP stream* atau *UDP datagram*. Gambar 2.2 menggambarkan proses *remote mount* yang terjadi dalam NFS. Pada server1 melakukan proses *file system mounted* di dalam direktori */usr/student* dalam *client sub-tree* pada lokasi */expert/people*, sedangkan server 2 melakukan proses *file system mounted* di dalam direktori */usr/staff* dalam *client sub-tree* pada lokasi */nfs/users*.



Gambar 2.2 Proses Remote Mount

2.2 Key Management

Adanya internet tidak bisa terlepas dari pemakaian data secara bersamaan, hal ini membutuhkan suatu sistem yang dapat menjamin pemakaian data-data

tersebut. Menurut Garfinkel, keamanan dalam berkomunikasi atau bertransaksi antara satu komputer dengan komputer lain melalui internet harus adanya *integrity, authenticity, confidentiality* dan *non-repudiability*^[1]. Tujuan ini dapat dicapai dengan adanya teknik kriptografi.

Key management adalah sekumpulan proses dan mekanisme yang mendukung penentuan *key* dan pemeliharaan *keying relationships* secara terus menerus secara berkelanjutan termasuk penggantian *key-key* yang lama dengan *key* baru apabila diperlukan^[3]. *Keying relationships* adalah suatu keadaan di mana dalam berkomunikasi adanya pemakaian data secara bersama-sama dengan menggunakan fasilitas dan teknik kriptografi. Data ini meliputi *public key* maupun *private key*, nilai-nilai inisialisasi, dan parameter tambahan lainnya yang tidak rahasia. Aktifitas yang dilakukan meliputi penanganan *key* kriptografi dan parameter lainnya yang berhubungan dengan keamanan (seperti *password*), selama adanya proses yang berhubungan dengan *key*, termasuk media penyimpanan, masukan dan keluaran, serta adanya perusakan dari pihak-pihak yang tidak berkepentingan.

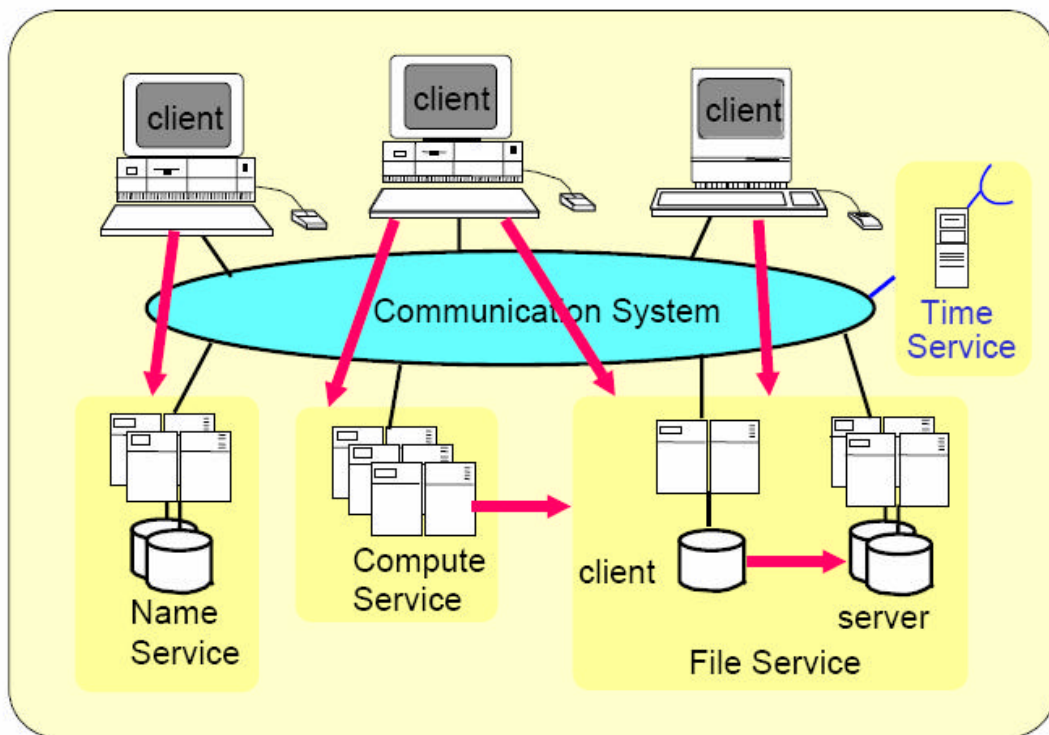
Konsep dasar *sistem key management* harus bisa menghasilkan, menyimpan dan menukar *key* seperti menverifikasi autentifikasi *key*. Suatu hal yang harus diperhatikan adalah dapat mengidentifikasi berbagai kelompok orang dalam proses komunikasi. Biasanya untuk menjamin keamanan berkomunikasi menggunakan *symmetric keys* untuk enkripsi murni (untuk menjaga integritas dan kerahasiaan) dan *asymmetric keys* untuk menukar *symmetric keys* dan *digital signatures*. Komponen utama dalam *sistem key management* adalah suatu *key database*, yaitu tempat penyimpanan dari *key-key* yang dapat menjawab semua kebutuhan. Jenis *key* yang berbeda memerlukan penyimpanan yang berbeda pula.

2.3 Self-Certifying File System (SFS)

Self-Certifying File System (SFS) adalah suatu pengamanan *file system* yang terdistribusi dan berhubungan dengan kegunaan, hal ini dapat meningkatkan keamanan jaringan dan mempermudah sistem administrator dalam suatu sistem jaringan. Dengan adanya SFS, siapapun dapat mengakses *file system* dengan aman

tanpa memerlukan *virtual private networks (VPNs)* atau *public key infrastructure (PKI)*. Administrasi SFS sangat disederhanakan oleh komputer *client* yang tidak mempunyai bentuk. Sebuah SFS *client* hanya dengan menjalankan sebuah program *sfsd* pada saat mem-*boot*, kemudian *users* dapat mengakses server manapun tanpa minta bantuan administrator.

Dalam beberapa *file sytem* terdistribusi biasanya diperlukan suatu daftar *client* untuk me-*remote file system*. Dalam SFS, sebenarnya sebuah *server* menentukan *file system client mount pathnames*.



Gambar 2.3 Sistem file terdistribusi

Secara kriptography, SFS menjamin semua jaringan komunikasi *client-server* tanpa enkripsi dan kode pesan autentifikasi. Untuk melindungi *server* dari seseorang yang melakukan kejahatan, masing-masing *server* SFS mempunyai suatu kunci publik (*public key*). Karena sebuah file server semuanya ada di bawah suatu *self-certifying pathname* yang diperoleh dari kunci publiknya. *Self-Certifying pathname* berisi informasi untuk SFS *client* yang menghubungkan

suatu *server* dan menjamin saluran komunikasi secara kriptography. Ada beberapa cara/teknik untuk para pengguna (*users*) untuk mendapatkan *self-certifying pathname server* dengan aman.

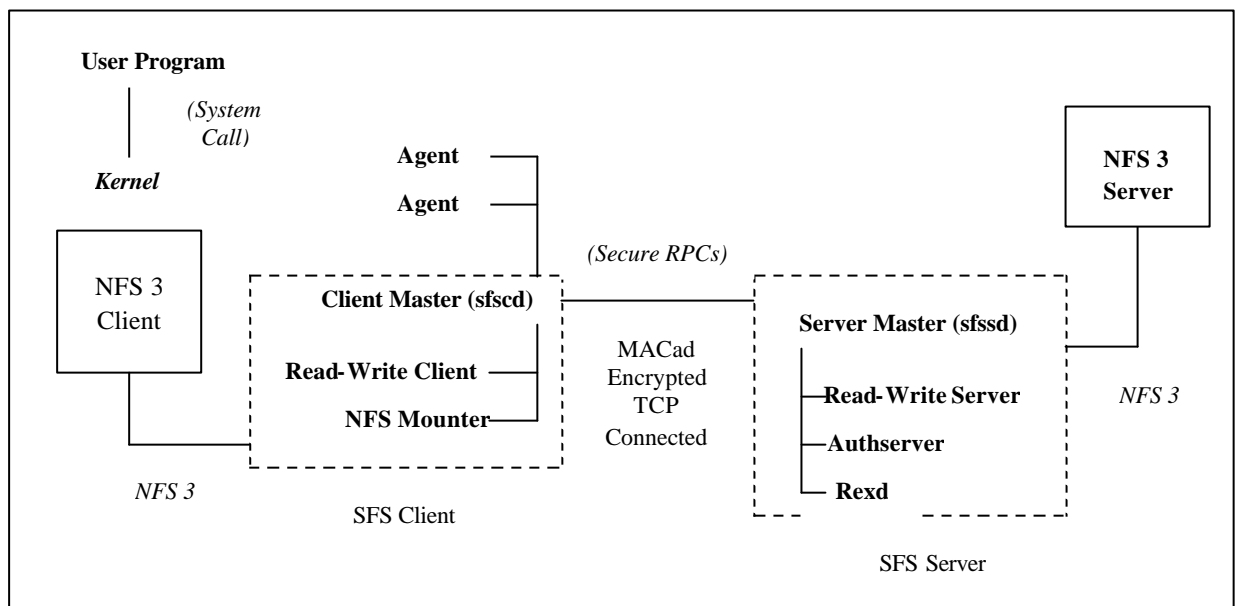
Untuk melindungi lalu lintas jaringan, SFS melakukan *user* autentifikasi untuk memetakan *remote users* dalam *file system* lokal. Mekanisme SFS's *user* autentifikasi didasarkan pada kriptografi kunci *public*. Dari sisi *client*, suatu agen tidak mempunyai hak istimewa untuk memproses seorang *user* yang mempunyai kunci *private* dan dapat dengan jelas membuktikan keaslian dari *users* untuk *me-remote server* apabila dibutuhkan. Dari sisi *server*, SFS menyimpan *database users* beserta kunci publiknya. *User* dapat dengan aman mendaftarkan kunci publik ke administrator. Sebaliknya administrator dapat dengan aman mengirim *database* kunci publik untuk satu sama lain. Suatu *file server* dapat mendatangkan dan bahkan menggabungkan *user account* dari beberapa dunia administrasi yang berbeda.

SFS adalah suatu *software freeware* dan dapat dijalankan dalam sistem operasi UNIX dan menggunakan NFS versi 3. Selain itu dapat juga berjalan dalam sistem operasi OpenBSD, FreeBSD, MacOSX, OSF/1, Solaris, and sebagian distribusi Linux.

Dari segi administratif, SFS berisi dua buah program yang harus dijalankan pada saat *mem-boot*. SFS *client* harus menjalankan SFS *client daemon* (*sfscd*), yang akan membuat suatu direktori */sfs* dan mengimplementasikan sebuah *auto-mounting* dari SFS *servers remote*. SFS *servers* harus menjalankan SFS *server daemon* (*sfssd*) yang membuat *file system* lokal ada pada SFS *client* TCP port 4.

Secara internal, SFS tersusun secara modular dan terdiri dari beberapa *daemon*. *sfscd* dan *sfssd* masing-masing berkomunikasi dengan sejumlah cabang *daemon* seperti dapat dilihat dalam Gambar 2.4. Sebagai contoh *sfscd* bertanggung jawab secara otomatis memasang *file system remote* baru. Dalam *server*, *sfssd* menerima SFS koneksi yang datang dan *demultiplexing* permintaan sesuai dengan SFS *server daemons*. Ada tiga dasar yang harus diperhatikan dalam sistem ini, yaitu SFS *file server*, autentifikasi *server* dan

remote login server. Client dan server file system daemons berkomunikasi dengan kernel yang menggunakan network file security (NFS) looback. Komponen-komponen SFS berkomunikasi satu sama lain menggunakan remote procedure call (RPC). Implementasi SFS dapat menjamin kerahasiaan dan integritas RPC transport.



Gambar 2.4 Arsitektur SFS^[2]

BAB III

PERANCANGAN SELF-CERTIFYING FILE SYSTEM

SFS's dirancang dengan sejumlah ide-de kunci. Dengan adanya nama file dalam SFS yang disebut dengan *self-certifying pathnames*, maka akan membolehkan autentifikasi *server* tanpa melakukan *key management*. Dengan melalui suatu implementasi modular, SFS juga membolehkan *user* autentifikasi keluar dari *file system*. SFS sendiri berfungsi sebagai *key management infrastruktur*, maka akan membuat semakin mudah untuk diimplementasikan dan dikombinasikan dengan berbagai macam mekanisme *key management*. SFS akan memisahkan *key management* dari *key distribution*, dan mencegah fleksibilitas dalam *key management* dari key yang telah disepakati.

3.1 Sasaran dan Tujuan SFS

Self-Certifying File System (SFS) adalah *file system* jaringan yang membolehkan file diakses secara bersama-sama dari manapun seseorang berada. SFS dirancang dengan tiga pemikiran sebagai berikut ^[5] :

- a. **Keamanan**, SFS di asumsikan sepenuhnya untuk mengendalikan jaringan.
- b. **A global namespace**, SFS *mounts* akan me-remote semua *file system* di bawah direktori */sfs*. Direktori ini berisi identitas dari setiap *client* di dunia.
- c. **Kontrol desentralisasi**, SFS tidak berdasarkan pada otoritas manapun untuk mengatur *namespace* global. Seseorang dengan komputernya yang terhubung ke internet dapat men-*set-up* SFS *file server* tanpa harus mempunyai segala hal tentang sertifikat. *Server* akan dapat segera diakses dari semua *client* di dunia.

3. 2 Self-Certifying pathnames.

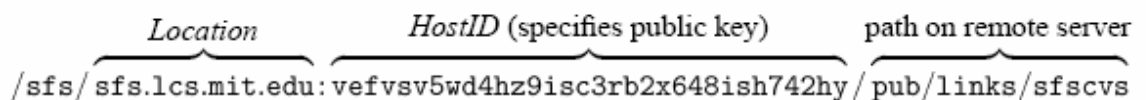
Untuk mencapai sasaran tersebut, SFS secara kriptograpy harus menjamin isi dari file *remote* tanpa didasarkan pada informasi eksternal. SFS tidak

memerlukan suatu otoritas global untuk mengkoordinir sistem keamanan. Para pengguna individu akan menyediakan *client* dengan informasi kemannya, tetapi pendekatan ini membuat sebuah *file cache* akan sulit disembunyikan diantara *users*. Tanpa informasi eksternal, SFS harus memperoleh file data dengan aman dengan hanya suatu nama file. Oleh karena itu SFS memperkenalkan *self-certifying pathnames* yaitu suatu nama file yang tidak terpisahkan dan berisi semua informasi yang diperlukan untuk berkomunikasi secara aman dengan *remote file server*, yaitu suatu alamat jaringan dan sebuah kunci publik.

Setiap file system SFS dapat diakses dibawah suatu *pathname* dengan format */sfs/Location:HostID*. *Location* menyatakan suatu SFS *client* berada, untuk mencari *file system's server*, sedangkan *HostID* menyatakan bagaimana *client* mengamankan saluran dari suatu server. *Location* dapat berupa *DNS hostname* atau *IP address*. Untuk mencapai komunikasi yang aman setiap SFS *server* mempunyai suatu kunci publik. Sedangkan *HostID* adalah suatu kriptography *hash* dari *key* dan lokasi *server*. *HostID client* akan memverifikasi keaslian dari kunci publik. SFS akan menghitung *HostID* dengan SHA-1^[7] yaitu :

$$\text{HostID} = \text{SHA-1} (\text{"HostInfo", Location, PublicKey, "HostInfo", Location, PublicKey})$$

SHA-1 mempunyai keluaran 20 byte, jauh lebih pendek dibanding dengan kunci publik. Meskipun demikian, dua masukan dari SHA-1 akan menghasilkan keluaran yang sama. Perhitungan ini akan membatasi para penyerang (*attacker*) menghasilkan dua kunci publik dengan *HostID* yang sama, secara efektif *HostID* akan menetapkan kunci publik yang unik. Gambar 3.1 menggambarkan format direktori dari *self-certifying pathname*. Semua file remote SFS berada dalam direktori



Gambar 3.1 Self-Certifying Pathnames^[6]

Dalam direktori tersebut, SFS *mounts me-remote file systems* dalam *self-certifying pathnames* dari format *Location:HostID*. SFS akan menyandikan ke 20 *byte HostID* berbasis 32, menggunakan 32 digit dengan huruf kecil. Untuk menghindari kekacauan, dalam huruf kecil, karakter “l” adalah huruf kecil dari L, “1” adalah satu, “0” dan “o”.

SFS *client* tidak perlu mengetahui tentang *file system* sebelum *users* mengaksesnya. Apabila referensi seorang pengguna tidak berada dalam direktori *self-certifying pathname /sfs*, maka seorang *client* dapat mencoba untuk menghubungkan dengan *server* yang disebut dengan *Location*. Jika *server* tetap berjalan, maka SFS dijalankan, dan akan membuktikan kunci *private* sesuai dengan *HostID*, sehingga *client* akan membuat *pathnames* dan *mount* yang disesuaikan dengan *file system remote* yang berada di *server*.

Self-certifying pathname bersama dengan *automatic mounting* akan menjamin semua *user* berhak membuat *file system*. *Location* adalah alamat internet atau nama domain, seorang pengguna dapat menghasilkan sebuah kunci publik, dengan menentukan *HostID* yang sesuai. Kebijakan *key management* dalam SFS diakibatkan oleh *names* dari *file user* dalam memutuskan suatu file diakses. Sehingga seorang *users* dapat mendapatkan kembali *self-certifying pathname* dengan *password*-nya.

3.3 Direktori / SFS

Setiap *user* dalam SFS *client* berjalan dalam suatu program *agent* dengan tidak mempunyai hak istimewa, dan berkomunikasi dengan *file system* yang menggunakan *remote procedure controll* (RPC). *Agent* akan menangani autentifikasi dari *user* ke *server remote*, dan mencegah *user* dari penarikan pengaksesan kembali *HostID*, dan mengendalikan *user* dari direktori */sfs*. *User* dapat mengganti kembali *agent* mereka sesuai dengan kebutuhannya.

SFS *client* akan memetakan setiap pengoperasian *file system agent* didasarkan pada permintaan. *Client* akan menjamin perbedaan direktori */sfs* untuk tiap-tiap *agent*, dan *track-track* dari *self-certifying pathnames* yang telah

disesuaikan dalam direktori */sfs*. Dalam daftar direktori */sfs*, *client* menyembunyikan *pathnames* yang belum diakses di bawah *agent* tertentu.

SFS *agent* mempunyai kemampuan untuk membuat *symbolic links* dalam */sfs* yang sesuai untuk prosesnya sendiri. *Links* ini dapat memetakan ke dalam *self-certifying pathname*. Apabila seorang *user* mengakses sebuah file tidak dalam kondisi *Location:HostID* dalam direktori */sfs*, *software client* akan memberitahu *agent* yang sesuai dengan kejadian. *Agent* dapat membuat *symbolic link* dalam mengatur kembali pengaksesan *user's*. *Symbolic link* adalah suatu cara untuk mempermudah user dalam menggunakan *self-certifying pathnames*.

3.4 Server key management

Adapun teknik-teknik *server key management* yang bermanfaat untuk membangun suatu SFS, diantaranya adalah sebagai berikut.

1. **Manual key distribution.** *Manual key distribution* digunakan untuk mempermudah SFS dalam penggunaan *symbolic link*. *Symbolic link* adalah suatu cara untuk mempermudah user dalam menggunakan *self-certifying pathnames*. Jika administrator di suatu tempat akan meng-*installs* beberapa *server public key* di lokal *hard disk* di setiap *client*, administrator dapat dengan mudah membuat sebuah *symbolic link* untuk *self-certifying pathname* dengan tepat. Sebagai contoh, diberikan server dengan alamat : **sfs.lcs.mit.edu**, maka semua komputer client akan berisi link : **/lcs ? /sfs/sfs.lcs.mit.edu: vefvsy5wdhz9isc3rb2x648ish742hy**. Semua user akan mengacu pada file **/lcs/....** . Daftar *password* dari semua *users* akan berada pada direktori home : **/lcs/users/dm**.
2. **Secure links.** Suatu *symbolic link* dalam satu file system SFS dapat menunjuk ke *self-certifying pathname* yang lain, sebagai pembentuk *secure link*. Dalam contoh sebelumnya, path **/lcs/pub/links/sfscvs** menunjuk ke file **/pub/links/sfscvs** di server **sfs.lcs.mit.edu**. File ini pada gilirannya sebagai *symbolic link* yang menunjuk ke *self-certifying pathname* dari server **sfscvs.lcs.mit.edu**. *User's* akan mengikuti *secure links* dan tidak perlu mengetahui *HostIDs*.

3. **Secure bookmarks.** Ketika *file system* SFS dijalankan, perintah *pwd* dalam UNIX akan memerintahkan kembali *self-certifying pathname* pada direktori yang sedang dipakai. Dari *pathname* ini, seseorang dapat dengan mudah menyadap *Location* dan *HostID* dari *server* yang sedang di akses. Ada 10 baris *shell script* yang disebut dengan *bookmark* untuk membuat suatu *link directory* : ***Location ? /sfs/Location:HostID in a user's ~/sfs-bookmarks.*** Dengan adanya *shells* yang mendukung variabel *cdpath*, *users* dapat menambah direktori ***/sfs-bookmarks*** ke dalam *cdpath*. Dengan hanya mengetik “*cd Location*”, sehingga *user* dapat mengembalikan kembali beberapa *file system*-nya.
4. **Certification Authorities.** SFS *certification authorities* tidak lain hanya *file system* yang melayani *symbolic link*. Sebagai contoh, Jika *verisign* bertindak sebagai SFS *certification authorities*, administrator *client* akan membuat *symbolic link* dari disk lokal ke *file system verisign's* : ***/verisign ? /sfs/sfs.verisign.com:r6ui9gwucpkz85uvb95cq9hdpfbz4pe.*** *File system* ini pada gilirannya berisi *symbolic links file system* SFS lain, sehingga ***/verisign/sfs.mit.edu*** akan menunjuk ke ***/sfs/sfs.mit.edu:bzcc5hder7cuc86kf6qswyx6yuemn69.***
5. **Password Authentication.** SFS membiarkan para *user* mengambil kembali *self-certifying pathname* dengan aman dari *server remote* menggunakan *password*. Sayangnya, *user* selalu memilih *password* yang tidak baik (jelek). Ada dua program yaitu *sfskey* and *authserv*, para *user* dapat dengan aman men-download *self-certifying pathname* menggunakan protokol SRP serta *password*-nya. Dalam menggunakan SRP, pertama kali penggunaan SFS akan menghitung suatu fungsi *password* dan menyimpannya dengan *authserv daemon* yang berjalan pada *file server*-nya. Pada saat *download* dilakukan di *channel file server self-certifying pathname*, *user's agent* akan membuat sebuah *link directory /sfs*.
6. **Certification paths.** Seorang *user* dapat memberikan *agent*-nya daftar direktori yang berisi *symbolic link*, sebagai contoh : ***~/sfs-bookmarks, /verisign, /verisign/yahoo.*** Ketika seorang *user* mengakses suatu *pathname*

yang tidak *self-certifying pathname* dalam */sfs*, *agent* akan memetakan nama yang dari masing-masing direktori dalam *path certification* secara berurutan. Jika *symbolic link* dapat menemukan nama yang sama dalam waktu pengaksesannya, maka akan dibuat *symbolic link* dalam direktori */sfs*.

7. ***Existing public key infrastructures.*** Pembuatan *symbolic link* dalam */sfs*, dapat digunakan untuk mengeksploitasi *public key infrastruktur* (PKI). Seseorang dapat membangun *agent* yang menghasilkan *self-certifying pathname* dari *SSL certificates*, yaitu */sfs/hostname.ssl*.

3.5 User Authentication

Sementara *self-certifying pathnames* memecahkan masalah untuk membuktikan keaslian *file server* ke *user*, maka SFS harus membuktikan keaslian semua *user* ke *server*. Oleh karena itu, SFS akan memisahkan *user autentifikasi* dari *file system*.

Dari sisi *client*, *agent* akan menangani *user autentifikasi*. Ketika seorang *user* pertama kali mengakses *file system* SFS, *client* akan menunda pengaksesan dan memberitahu kejadian tersebut ke *agent*-nya. *Agent* dapat membuktikan keaslian *user* melalui *remote server* sebelum pengaksesan file diselesaikan. Dari sisi *server*, ada sebagian program, untuk membuktikan keaslian *server* atau “*authserver*”, yang akan membuktikan keaslian *user*. *File server* dan *authserver* akan berkomunikasi dengan *remote procedure call* (RPC).

Agent dan *authserver* akan melewati pesan-pesan ke satu sama lain melalui SFS dengan menggunakan sebuah *protocol* yang tak dapat ditembus ke dalam perangkat lunak *file system*. Jika *authserver* menolak permintaan autentifikasi, *agent* dapat mencoba kembali menggunakan *credentials* yang berbeda atau *protocol* yang berbeda.

Apabila *user* tidak mempunyai *account* dalam *file server*, *agent* dapat menggagalkan autentifikasi *user*. *User* akan mengakses *file system* dengan ijin tanpa nama (*anonymous permissions*).

3.5.1 *sfsagent* and *authserv*

Dalam sub-bab ini akan menjelaskan sistem *user* autentifikasi yang dirancang dan dibuat untuk SFS. Sistem ini berisi program *agent* yang disebut *sfsagent*, *authserver*, dan *authserv*.

Salah satu keuntungan dari *self-certifying pathnames* adalah kemudahan dalam menentukan file *server* baru. Jika para *user* terlebih dulu harus menentukan autentifikasinya secara terpisah ke tiap-tiap *server* baru, hal ini akan menyulitkan *server* baru yang baru dibuat. Untuk itu akan dibuat autentifikasi lebih terbuka dan dapat digunakan oleh *user* SFS.

Semua *user* mempunyai satu lebih kunci publik dalam satu sistem. *sfsagent* berjalan dan berhubungan dengan kunci publik. Ketika seorang *client* meminta suatu *agent* untuk autentifikasi *user*-nya, *agent* secara digital memberi tanda permohonan autentifikasi. Permohonan melalui *client* ke *server*, pengesahaannya melalui *authserv*. *authserv* akan menjaga dan memelihara pemetaan database kunci publik ke *user* yang setelah diberi mandat (*user credentials*). Ketika menerima suatu permintaan yang sah (*valid*) dari *file server*, *authserv* akan menjawab dengan sekumpulan mandat dari Unix yaitu *user ID* dan daftar kelompok *IDs*.

sfsagent yang ada sekarang hanya menyimpan kunci privat *user* dalam *memory*. Bagaimanapun, kita harus mempertimbangkan berbagai *agent* yang lebih bagus. *Agent* tidak mempunyai kekuasaan secara langsung dari kunci publik. Untuk melindungi kunci privat dari hal-hal yang membahayakan, sebagai contoh, seserang dapat menjebol *agent* dan *authserver* yang dipercayai menggunakan keamanan proaktif. Seorang penyerang akan membahayakan *agent* dan *authserver* untuk mencuri kunci rahasia yang dipisah. Sebagai alternatif, *agent* mungkin hanya berkomunikasi melalui *port* serial dengan sebuah *personal digital assistance* (PDA) yang mengetahui kuncinya.

Proxy agent dapat meminta pengajuan autentifikasi ke SFS *agent* lain. Diharapkan dapat membuat suatu login yang serupa ke algoritma SSH ^[8] yang berlaku sebagai *proxy SFS agent*. Dengan cara ini, pemakai dapat secara otomatis mengakses filenya ketika melakukan login ke komputer remote. Permohonan

otentifikasi berisi self-certifying pathname server yang diakses oleh user. Selain itu berisi sebuah field yang dipesan untuk pemrosesan. Dengan demikian, SFS agent secara penuh memeriksa setiap pengoperasian kunci privat yang dilakukan.

3.5.2 User key management

authserv menterjemahkan permintaan autentifikasi ke dalam *credentials*. Hal ini dikerjakan melalui konsultasi pemetaan satu atau lebih *database* kunci publik ke para *user*. Karena SFS adalah sesuatu yang mengamankan *file system*, beberapa database dapat berada pada *file server remote* dan diakses melalui SFS-nya. Sebagai contoh, sebuah server dapat mengimport suatu daftar centrally-maintained para *user* SFS, sedangkan pengaturan beberapa *guest account* dalam database lokal. *authserv* secara otomatis menyimpan salinannya dalam database remote lokal.

Masing-masing *database* kunci public *authservers* diatur baik secara *read-only* maupun *writable*. *authserv* menangani sejumlah tugas-tugas yang menangani *user* dalam *database writable*. Hal ini membolehkan untuk menghubungkan di atas jaringan dengan *sfskey* dan perubahan kunci publik. Suatu server dapat me-mount *password guess* yang menyerang kembali kepada seorang *user* jika mengetahui data SRP atau kunci privat yang dienkripsi.

Tidak semua server memerlukan akses ke kunci privat *user* yang telah dienkripsi atau data SRP. *authserver* menjaga dan memelihara setiap versi dari *database writable*, satu publik, dan satu privat. *Database* publik berisi kunci publik dan *credentials*, tetapi tidak ada informasi seorang penyerang dapat memverifikasi *password*-nya. Suatu server dapat mengekspor suatu *database* publik ke seluruh dunia menggunakan *file system* SFS. *authservers* lain dapat membuat *read-only* yang digunakan. Sebagai contoh, pusat server dapat dengan mudah memperbaiki kunci-kunci semua *user* dalam suatu departmen dan mengirim *database* publik-nya ke *file server* secara terpisah tanpa kepercayaan penuh.

3.6 Revocation

Apabila kunci privat *server* disepakati, *selfcertifying pathname* lama dapat menjadi *user* untuk penipu *server* yang dijalankan oleh penyerang. Disini SFS menyediakan dua mekanisme untuk mencegah user dalam mengakses *self-certifying pathnames* yang salah yaitu pembatalan kunci (*key revocation*) dan penolakan HostID (*HostID blocking*). *Key revocation* terjadi hanya mendapat ijin dari pemilik *file server*. Hal ini terjadi secara otomatis menggunakan beberapa *user* yang memungkinkan. Pada sisi lain *HostID blocking* dimulai dari sebuah sumber lain pemilik *file system*. *Agent user* individual memutuskan atau bukannya menghalangi *HostID*.

Dengan digunakannya SFS, maka akan memisahkan *key revocation* dari *key distribution*, sehingga mekanisme *single revocation* dapat menarik kembali suatu *HostID* yang telah dibagi-bagikan melalui jalan yang berbeda. SFS menggambarkan suatu pesan (*message*) yang disebut *key revocation certificate*, dengan format sebagai berikut :

$$\{ \text{“PathRevoke”}, \textit{Location}, \textit{K}, \textit{NULL} \}_{k-1}$$

Revocation certificates sedang di *self-authenticating*. Yang berisi suatu kunci publik, *K*, yang harus di tandatangani dan berkoresponden dengan kunci privat, k_{-1} . “*PathRevoke*” adalah sesuatu yang tetap. *Location* berkoresponden dengan *location* yang di tarik kembali oleh *self-certifying pathname*. *NULL* menandakan *revocation certificate* dari format yang ditunjuk. Suatu *revocation certificate* selalu menolak penunjuk sebelumnya dari *HostID* yang sama.

Apabila perangkat lunak SFS *client* mengetahui *revocation certificate*, maka penghalangan pengaksesan lebih lanjut oleh *user* ke *HostID* ditentukan oleh *certificate's Location* dan *K*. *Client* memperoleh *revocation certificate* dengan dua cara yaitu dari *server* dan *agent*. Ketika SFS pertama kali disambungkan ke suatu *server*, SFS akan memberitahu *Location* dan *HostID* dari *file system* yang mengaksesnya. *Server* dapat menjawab dengan suatu *revocation certificate*. Hal ini tidak dapat dipercaya artinya *revocation certificate* yang terdistribuai, tetapi mungkin dapat membantu dengan cepat penarikan *pathname* kembali. Sebagai

alternatif, ketika seorang *user* pertama kali mengakses suatu *self-certifying pathname*, *client* akan meminta *agent*-nya untuk memeriksa jika *path*-nya ditarik kembali. Pada saat ditunjuk, *agent* dapat menjawab dengan suatu *revocation certificate*.

Revocation certificate boleh jadi digunakan. *Verisign* menentukan untuk menjaga direktori yang disebut */verisign/revocations*. File berada dalam direktori yang bernama *HostID*, dimana masing-masing file berisi sebuah *revocation certificate* untuk berkoresponden dengan *HostID*. Kapan saja seorang *user* mengakses *file system* baru, *agent*-nya memeriksa direktori *revocation* untuk mencari suatu *revocatin ceertificate*. Jika ada, agen akan mengembalikannya ke *software client*.

Sementara *revocation certificate* sedang di autentifikasi, otoritas sertifikasi tidak memerlukan pemeriksaan *user*.

Kadang-kadang suatu *agent* membolehkan suatu *pathname* yang telah lama ditemukan tanda *revocation certificates*. Sebagai contoh, sekalipun pemilik *file system* belum menarik kembali kunci *file syatem*-nya, suatu *agent* dapat menemukan otoritas *revocation* dalam beberapa eksternal *public key infrastructure* (PKI) yang telah ditarik kembali sesuai dengan *certificates*. Untuk menampung situasi seperti itu, *agent* dapat meminta *HostID blocking* dari *client*. Hal ini mencegah pemilik *agent* dalam mengakses *self-certifying pathname* yang dimasalahkan, tetapi tidak mempengaruhi para *user* lainnya.

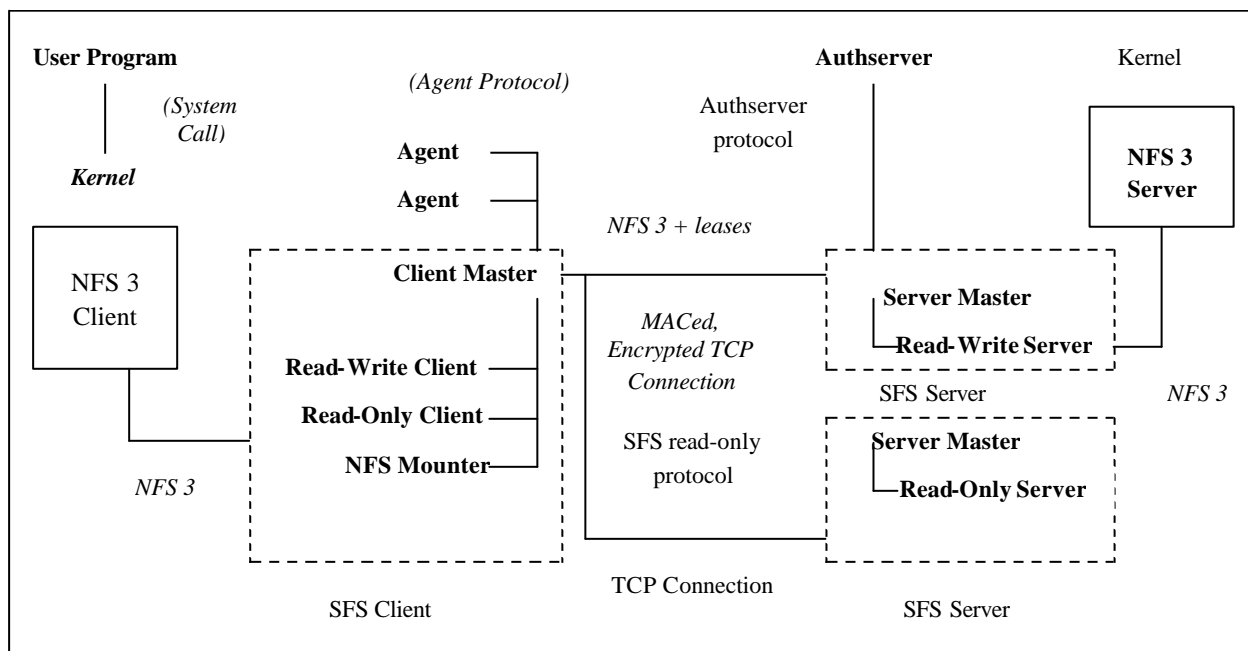
Kedua-duanya menarik kembali dan menghalangi *self-certifying pathname* menjadi *symbolic links* ke file yang tidak ada *:REVOKED:*. Sementara itu, pengaksesan suatu *path* ditarik kembali dan tidak mengakibatkan suatu file ditemukan mengalami kesalahan.

BAB IV

HASIL PERANCANGAN DAN IMPLEMENTASI

4.1 Implementasi

Gambar 4.1 menunjukkan bagian-bagian dari sistem SFS. Di mana, SFS terdiri dari *client* dan *server* yang digabung oleh koneksi TCP. Untuk portabilitas, perangkat lunak client bertindak sebagai NFS versi 3^[4] server. Hal ini untuk berkomunikasi dengan sistem operasi melalui sistem jaringan biasa. Ketika para pemakai mengakses file di bawah SFS, kernel mengirim NFS RPCs ke perangkat lunak *client*. *Client* akan memanipulasi RPCs dan menjamin *channel* ke SFS *server* yang sesuai. Server memodifikasi sedikit permintaan dengan berdasarkan *credential* yang sesuai. Akhirnya, server akan bertindak sebagai suatu NFS *client*, melalui permintaan NFS server dalam komputer yang sama. Sebaliknya, tanggapan akan dilakukan mengikuti *path* yang sama.



Gambar 4.1 Bagian-bagian dalam Sistem SFS

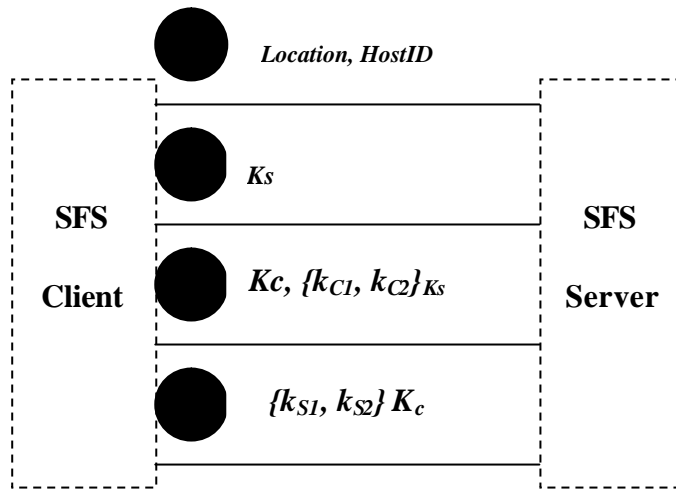
4.2 Cryptographic protocols

Bagian ini akan menjelaskan protokol SFS *client* yang disediakan untuk mengamankan *channel* ke *server* dan *users* membuktikan keasliannya ke server. K_U , K_S , dan K_C menunjukkan kunci-kunci publik yang dimiliki oleh *user*, *client* dan *server*. K^{-1} menunjukkan kunci privat yang berkorespondensi dengan kunci K . *Subscript* K merepresentasikan pesan yang dienkripsi dengan kunci K , sementara *subscript* K^{-1} menandakan suatu pesan yang ditandai oleh K^{-1} .

4.3 Key negotiation

Ketika perangkat lunak SFS mendapatkan *client self-certifying pathname* untuk pertama kali, maka ia harus menetapkan keamanan *channel* ke server yang sesuai. *Client* akan memulai dengan menghubungkan ke komputer yang disebut *Location* dalam *pathname*. Hal ini akan meminta kunci publik K_S , (seperti terlihat dalam gambar 4.2 (langkah ke dua) dan memeriksa kunci sesungguhnya yang memenuhi *HostID pathnames*. Jika kunci memenuhi *pathnames*, *client* akan mengetahui bahwa telah memperoleh kunci publik dengan benar.

Ketika *client* mengetahui kunci *server*, *client* akan merundingkan bersama *session key* dalam penggunaan protokol yang sama. Untuk menjamin kerahasiaan, *client* menggunakan suatu kunci publik yaitu K_C untuk waktu yang pendek. Pada gambar 4.2, langkah 3 akan dikirim dari *client* ke *server* melalui jaringan yang tidak aman. *Client* kemudian mengambil secara acak *key halves* k_{c2} dan k_{c1} dengan cara yang sama *server* mengambil *key halves* K_{S1} dan K_{S2} . Pertukaran dua *key halves* dan enkripsi dapat dilihat dalam gambar 4.2.



Gambar 4.2. SFS key negotiation protocol

Client dan *server* secara serempak akan men-dekripsi tiap-tiap *key halves* satu sama lain, perhitungan dilakukan untuk memperkecil *latency*. Akhirnya, dihitung *keys-one* untuk masing-masing arah mengikuti persamaan dibawah ini :

$$\begin{aligned} k_{CS} &= \text{SHA-1}(\text{"KCS"}, K_s, k_{S1}, K_c, k_{C1}) \\ k_{SC} &= \text{SHA-1}(\text{"KSC"}, K_s, k_{S2}, K_c, k_{C1}) \end{aligned}$$

Client dan *server* menggunakan *session* kunci ini untuk meng-enkrip dan menjamin integritas dari semua komunikasi yang berikutnya di dalam *session*.

Protokol *key negotiation* ini menjamin *client* sehingga dapat mengetahui k_{CS} dan k_{SC} tanpa harus mengetahui K_s^{-1} . Hal ini memberi *channel* dan menjamin *client* ke *server* yang diinginkan. Sementara K_c adalah tanpa nama dan tidak punya hubungan dalam akses kendali atau user autentifikasi. *Client* membuang dan memperbaharui K_c pada waktu tertentu (setiap saat mengalami kegagalan).

4.3 User authentication

SFS agent dan *authserver* akan mengandalkan pada kunci publik untuk *user autentifikasi*. Setiap *user* mempunyai suatu kunci publik dan memberi *agent*-nya untuk mengakses kuncinya. Setiap *authserver* mempunyai suatu pemetaan dari kunci publik ke *credentials*. Ketika seorang *user* mengakses suatu file sistem baru, perangkat lunak *client* akan membangun suatu permintaan dari

agent untuk ditandai. *Client* melalui permintaan yang ditandatangani ke *server*, meminta *authserver* untuk memvalidasi.

SFS menetapkan suatu struktur *Autinfo* untuk mengidentifikasi *session* secara unik yaitu :

$$\begin{aligned} \textit{SessionID} &= \text{SHA-1}(\text{"SessionInfo"}, k_{SC}, k_{CS}) \\ \textit{AuthInfo} &= \{\text{"AuthInfo"}, \text{"FS"}, \textit{Location}, \textit{HostID}, \textit{SessionID}\} \end{aligned}$$

Perangkat lunak *client* juga menyimpan suatu alat penghitung untuk masing-masing *session* yang menugaskan suatu nomor jumlah urutan untuk setiap autentifikasi yang diminta.

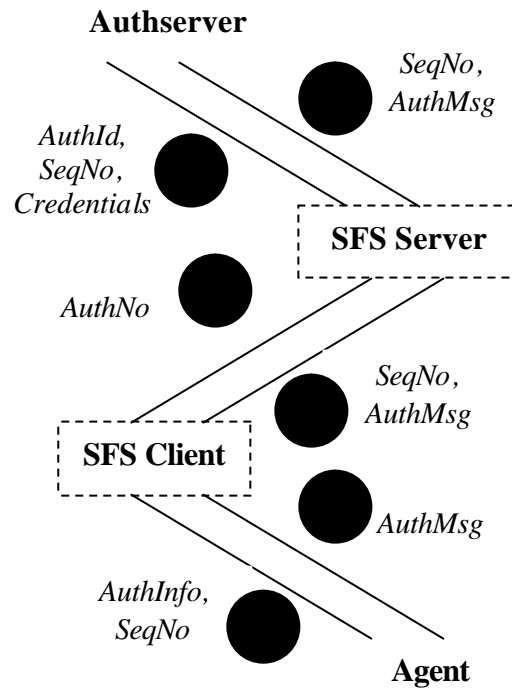
Ketika seorang *user* mengakses suatu *file system* untuk pertama kali, *client* mulai memproses *user* autentifikasi dengan mengirim suatu struktur *AuthInfo* dan nomor jumlah urutan ke *agent user*, seperti terlihat pada gambar 4.3. *Agent* mengembalikan suatu *AuthMsg* dengan struktur *AuthInfo* hashing ke suatu *AuthID* 20-byte yaitu berupa rangkaian jumlah urutan, penandaan hasil, dan penambahan kunci publik user sebagai berikut :

$$\begin{aligned} \textit{AuthID} &= \text{SHA-1}(\textit{AuthInfo}) \\ \textit{SignedAuthReq} &= \{\text{"SignedAuthReq"}, \textit{AuthID}, \textit{SeqNo}\} \\ \textit{AuthMsg} &= \text{KU}, \{\textit{SignedAuthReq}\} K_U^{-1} \end{aligned}$$

Autentifikasi pesan diperlakukan oleh *client* sebagai data yang tidak jelas. Dengan menambahkan nomor jumlah dari salinan yang lain dan mengirimkan data tersebut ke *file server* yang pada gilirannya diteruskan ke *authserver*. *Authserver* akan memverifikasi tanda tangan pada permintaan dan memeriksa bahwa *sequence number* yang ditandatangani sesuai dengan pilihan *client*. Jika permintaannya adalah sah, *authserver* memetakan publik *agent* ke sekumpulan *credentials* lokal. Jika *credentials* dikembalikan ke server bersama dengan *AuthID* dan pesan *sequence number* ditandatangani.

Server memeriksa apakah *AuthID* memenuhi *session* dan *sequence number* tidak berada di dalam *session* 4 yang sama. Jika segala sesuatunya berhasil, *server* menugaskan *sequence number* autentifikasi ke *credentials*, dan mengembalikannya lagi ke *client*. Permintaan label *client* semua *file system* dari *user* dengan nomor autentifikasi. Jika pada sisi lain, autentifikasi gagal dan

agent tidak memilih untuk mencoba lagi, maka permintaan label *client* semua *file system* dari *user* dengan nomor autentifikasi nol, yang dipesan oleh SFS untuk mengakses tanpa nama.



Gambar 4.3 SFS user authentication protocol

Sequence number tidaklah diperlukan untuk keamanan *user* autentifikasi. Seperti, untuk seluruh protokol *user* autentifikasi yang terjadi di atas suatu *secure channel*, maka semua pesan-pesan autentifikasi yang diterima baru saja oleh server harus dihasilkan oleh client. *Sequence number* akan mencegah satu *agent* dari penggunaan permintaan autentifikasi rahasia yang ditandatangani.

4.4 Cryptography

SFS membuat tiga asumsi perhitungan. Yang mengasumsikan ARC4 *stream chiper* yaitu suatu *pseudo-random generator*. Akhirnya, menggunakan asumsi SHA-1 yang bertindak sebagai suatu *random oracle*.

SFS menggunakan suatu *generator random* dalam algoritma dan protokolnya. DSS'S dan *pseudo-random generator* dipilih sebab didasarkan pada

SHA-1. Untuk menempatkan *generator*, SFS secara serempak membaca data dari berbagai program eksternal (ps, netstat) dari */dev/random* (jika tersedia), dari suatu file acak yang di simpan oleh program eksekusi dan dari suatu *nano second* (jika mungkin) pengatur waktu untuk menangkap entropi penjadwalan proses. Semua sumber diatas berjalan melalui algoritma SHA-1 berdasarkan fungsi hash untuk menghasilkan 512 byte. Sebab program eksternal berjalan secara paralel dan SFS membaca secara tak serempak, SFS dapat secara efisien menempatkan generator dari semua sumber setiap kali awal program dieksekusi.

SFS menggunakan Rabin kunci publik *cryptosystem*^[9] untuk enkripsi dan menandatangani. Implementasinya adalah dapat menjamin melawan serangan *chosen-ciphertext* adaptif dan *chosen-message*. Rabin mengasumsikan dengan pengecualian, membuat implementasi SFS's tidak lebih sedikit menjamin dalam model random oracle dari pada sistem kripto yang aberdasarkan RSA. Seperti *low-exponent* RSA, enkripsi dan Rabin verifikasi tandatangan sangat cepat, sebab tidak memerlukan modular exponentiation.

SFS menggunakan kode pesan autentifikasi (MAC) SHA-2 untuk menjamin integritas dari semua lalu lintas *file system* antara *client* dan *read-write server*, dan lalu lintas enkripsi ini dengan ARC4. Kedua enkripsi dan MAC mempunyai implementasi yang sedikit baku. Implementasi ARC4 menggunakan kunci 20 byte dengan sekali penjadwalan kunci ARC4 untuk masing-masing 128 bits data kunci. SFS tetap menyimpan ARC4 stream running untuk jangka waktu suatu session. Penguncian kembali SHA-1 yang didasarkan MAC untuk masing-masing pesan menggunakan 32 bytes data dari ARC4. MAC dihitung dari isi panjangnya *plaintext* untuk tiap-tiap pesan RPC (tidak digunakan untuk kepentingan enkripsi). Semua panjang pesan dan MAC akan dienkripsi.

SFS's stream chipher adalah serupa dengan ARC4 penjadwalan *key*, dan sebagai konsekuensinya mempunyai performasi yang sama. SFS's MAC lebih lambat dibanding alternatif lainnya seperti MD5 HMAC. Keduanya adalah implementasi dari artifak dan dapat ditukar oleh algoritma yang lebih populer tanpa saling mempengaruhi.

BAB IV

KESIMPULAN

Ada beberapa kesimpulan yang dapat diambil dalam tulisan ini adalah sebagai berikut.

1. SFS tidak memerlukan informasi apapun selain dari sebuah *self-certifying pathname* untuk menghubungkan ke *remote file server* secara aman. Sehingga hasil yang dapat diperoleh SFS adalah suatu sistem yang secara aman dan dapat menyediakan sistem file global tanpa adanya suatu *key management*.
2. Adanya beberapa teknik *key management* yang digunakan dalam SFS.
3. Dengan digunakannya *namespace global*, SFS dengan sendirinya akan membuat suatu infrastruktur *key management*. Nama file kunci publik adalah bagian dari *self-certifying pathnames*, dan nama file kunci publik berasal dari *symbolic links*.
4. Dengan adanya perangkat lunak *freeware* (SFS) ini, maka para pengguna (*user*) dapat mengamankan file-nya dalam pertukaran data dalam sistem jaringan global (internet) tanpa perlu adanya administrasi terlebih dahulu.

DAFTAR PUSTAKA

- [1] Garfinkel, Simon, *PGP : Pretty Good Privacy*, O'Reilly & Associates, Inc, New York, (1995).
- [2] Fu, Kaminsky, and Mazières, *Using SF Using SFS for a Secure Network File System*, The Magazine Of Usenix & Sage, volume 27, number 6, December 2002.
- [3] Lehtonen, Sami and Pärssinen, Juha, *A Pattern Language for Key Management*, 2001.
- [4] RFC1813 - NFS Version 3 Protocol Specification, <http://www.ietf.org/rfc/rfc1813.txt>.
- [5] Mazieres, David, *SFS 0.7.1 Manual*, Free Software Foundation, 2002.
- [6] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124–139, Kiawa Island, SC, 1999. ACM.
- [7] FIPS 180-1, *Secure Hash Standard*. U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, VA, 1994.
- [8] Tatu, Ylönen, *SSH – secure login connections over the Internet*. In *Proceedings of the 6th USENIX Security Symposium*, page 37-42, San Jose, CA, July 1996.
- [9] Hugh C. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, November 1980.