Penggunaan Kriptografi dalam Perancangan Protokol Sinkronisasi *File* yang Aman dan *Reliable*

Achmad Baihaqi - 13508030[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if18030@students.if.itb.ac.id*

***Abstract*—Sinkronisasi *file* adalah hal yang sangat umum dihadapi dalam bekerja dengan lebih dari satu *host*. Sinkronisasi *file* ini berguna dalam menyamakan isi *repository* antar suatu komputer atau *host*. Untuk menyamakan isi *repository* ini digunakan suatu protokol komunikasi. Pada makalah ini penulis akan mencoba merancang dan mengimplementasikan suatu protokol komunikasi untuk sinkronisasi *file* ini yang aman (*secure*). Untuk mengamankan ini maka penulis memakai beberapa algoritma dalam ilmu kriptografi. Algoritma tersebut diantaranya algoritma kriptografi kunci-publik yaitu RSA, algoritma kriptografi simetri yaitu RC4, dan fungsi *message digest* yaitu SHA-1. Dalam perancangannya akan dibutuhkan suatu *server* yang berfungsi untuk mengkoordinasikan beberapa *client* yang ingin mensinkronkan isi *repository*-nya.**

***Index Terms*—Sinkronisasi *File*, Protokol Komunikasi, SHA, RSA, RC4.**

I. PENDAHULUAN

Dalam bekerja dengan lebih dari satu *host*, pertukaran dan sinkronisasi *file* adalah hal yang sangat umum dihadapi. Sinkronisasi *file* ini adalah suatu sistem untuk menyamakan isi suatu *repository* antar beberapa *host* yang tersebar dan dihubungkan oleh suatu perangkat jaringan komputer. Sinkronisasi *file* ini banyak dipakai oleh komputer *server*. Contohnya suatu *server* yang akan banyak diakses oleh user dalam suatu waktu, mungkin *server* tersebut akan diimplementasikan dengan menggunakan banyak *server* agar layanan yang disediakan oleh *server* tersebut tersedia dengan baik dan tidak mudah down. Akan tetapi dengan menggunakan banyak *server* ini memiliki dampak negatif yaitu terjadi perbedaan data antar suatu *server* dengan *server* lain. Hal tersebut biasanya dapat diatasi dengan menjalankan sinkronisasi data secara berkala. Dengan demikian data-data antara satu *server* dengan *server* yang lain akan saling konsisten satu sama lain.

Contoh lain yang menggunakan sinkronisasi *file* adalah kolaborasi pekerjaan dalam suatu kelompok. Dalam suatu kelompok tersebut mempunyai anggota yang masing-masing mempunyai komputer. Masing-masing anggota harus mempunyai data yang sama dengan anggota lain yang disebut *repository* pada komputernya masing-masing. Agar data pada masing-masing *repository* anggota dalam kelompok tersebut konsisten maka diperlukan sinkronisasi *file* yang berkala. Contoh ini merupakan kejadian yang sering dibutuhkan pada kenyataannya.

Beberapa contoh perangkat lunak sinkronisasi *file* yang ada saat ini yaitu Dropbox, Windows Live Synch, rsynch, SVN, dll. Pada perangkat lunak yang ada kebanyakan menggunakan protokol HTTP sebagai media komunikasinya. Sedangkan protokol HTTP adalah protokol yang kurang aman.

Pada makalah ini penulis akan mencoba dalam merancang dan mengimplementasikan protokol komunikasi baru untuk sinkronisasi *file* ini. Nantinya protokol sinkronisasi tersebut dirancang agar proses transaksi yang dilakukan tersebut aman. Artinya semua data penting yang akan dilewatkan dalam jaringan akan dienkripsi dengan menggunakan algoritma enkripsi *stream cipher*. Sedangkan untuk pertukaran kunci dari algoritma tersebut akan menggunakan algoritma kunci-publik.

Sedangkan untuk mengkonsistenkan isi *repository* antar suatu *host*, akan dibantu oleh fungsi dalam ilmu kriptografi berupa fungsi *message digest*.

II. DASAR TEORI

A. Algoritma RSA

Algoritma RSA adalah salah satu algoritma kunci-public yang terkenal dan paling banyak aplikasinya. Kemanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor prima. Algoritma ini ditemukan oleh Rivest, Shamir, dan Adleman. Algoritma ini menggunakan sepasang kunci yaitu kunci publik dan kunci privat. Kunci publik ini digunakan untuk mengenkripsi pesan sedangkan kunci privat digunakan untuk mendekripsi pesan yang telah dienkripsi dengan kunci publik tadi. Formulasi enkripsi dan dekripsi hampir sama yaitu dengan memangkatkan pesan dengan kunci kemudian dimodulus dengan n. n ini didapatkan pada saat pembangkitan kunci publik dan kunci privat.[3]

Pada protokol ini algoritma RSA digunakan sebagai

metode pertukran kunci untuk proses enkripsi dan dekripsi pada algoritma *stream cipher*. Kunci yang akan dipakai dalam algoritma ini sepanjang 512 bit.

B. Algoritma RC4

Algoritma RC4 termasuk ke dalam cipher aliran (*stream cipher*). Algoritma ini dibuat oleh Ron Rivest. Dalam prosesnya algoritma RC4 membangkitkan aliran kunci (*keystream*) yang kemudian di-XOR-kan dengan plainteks. RC4 memproses data dalam ukuran *byte*, bukan dalam *bit*.[4]

Sampai saat ini tidak ada yang dapat memecahkan RC4 sehingga dikatakan sangat kuat. Hasil dari proses algoritma ini ukuran dari *cipherteks* akan sama dengan plainteks.

Pada protokol ini algoritma RC4 digunakan untuk mengenkripsi dan mendekripsi pesan yang akan dikirimkan antar *server* dan *client*. Pesan tersebut bisa berupa pesan *request*, pesan *response*, atau isi dari suatu *file*. Algoritma RC4 ini digunakan karena algoritma ini merupakan algoritma *stream cipher*, sedangkan pada protokol ini dibutuhkan pengiriman pesan secara aliran (*stream*). Alasan lain yaitu karena format protokol ini menggunakan *bytes stream*, maka dibutuhkan pengiriman secara per-*byte* dan algoritma ini bisa melakukan hal tersebut. Nantinya panjang kunci yang akan dibangkitkan sebanyak 128 bit.

C. SHA-1

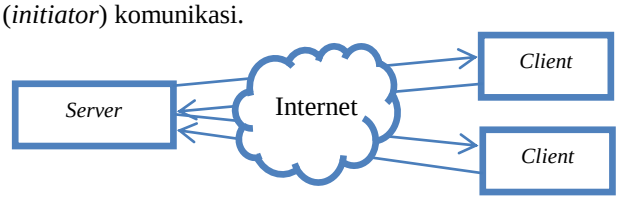
SHA adalah fungsi *hash* satu-arah yang dibuat oleh NIST dan digunakan bersama DSS (*Digital Signature Standard*). Algoritma SHA menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit. SHA mengacu pada keluarga fungsi *hash* satu-arah. Beberapa varian dari SHA adalah SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512. Pada fungsi SHA-1 menggunakan putaran sebanyak 80 kali.

Pada protokol ini fungsi SHA yang digunakan adalah SHA-1. Nantinya SHA akan digunakan sebagai referensi apakah suatu *file* dengan nama *file* yang sama telah dimodifikasi atau tidak.

III. PERANCANGAN

A. Arsitektur Komunikasi

Komunikasi protokol ini dirancang dengan arsitektur *client-server*. *Client* ini bertindak sebagai penginisiasi (*initiator*) komunikasi.



Gambar 1 Arsitektur protokol.

Masing-masing *client* dan *server* mengirimkan suatu pesan (*message*) yang telah didefinisikan. Protokol ini diimplementasikan pada lapisan (*layer*) aplikasi dalam arsitektur model referensi TCP/IP maupun model referensi OSI dan protokol ini akan diimplementasikan diatas koneksi TCP/IP. Koneksi TCP/IP ini digunakan agar antara *server* dan *client* tidak perlu menangani pesan yang hilang dalam perjalanan.

Protokol ini bersifat *statefull* artinya antara *server* dan *client* menjaga alur interaksi[8], jadi koneksi yang telah dibangun antara *client* dan *server* akan terus berjalan sampai salah satu dari *client* atau *server* menutup koneksinya. Semua pesan yang dikirim akan menggunakan koneksi yang sama dengan koneksi awal.

B. Perancangan Protokol DTP

Protokol yang akan dirancang dinamakan Protokol DTP (*Drogbox Transfer Protocol*). Protokol ini menggunakan format pesan yang disebut format *length-prefixed message* yang didefinisikan sebagai berikut:

<length prefix><message ID><payload>

Length prefix ditulis dalam format 4 *byte* big-endian. *Length prefix* dihitung dari ukuran *message ID* + ukuran *payload*. *Message ID* dituliskan dalam format 1 *byte* big-endian. *Payload* setiap *message* dapat bervariasi. *Message-message* yang ada diantaranya:

a. Handshaking
- *length prefix*: 23
- *message ID*: 0
- *payload*: *text* yang di-*encode* dengan UTF berisi *identifier* dari protokol yaitu *text* "didierdrogboxmelet"
- arah komunikasi: *client* → *server* dan *server* → *client*
- *Handshaking* ini digunakan untuk mengidentifikasi apakah *server* atau *client* yang sedang berkomunikasi adalah *server* atau *client* yang menyediakan protokol DTP ini. Apabila ternyata bukan merupakan *server* atau *client* yang benar, maka segera koneksi akan ditutup. Apabila suatu *server* menerima *message* ini, maka *server* tersebut harus langsung membalasnya dengan *message handshaking* yang sama.

b. Request Public Key
- *length prefix*: 1
- *message ID*: 1
- *payload*: kosong (tidak ada *payload*)
- arah komunikasi: *client* → *server*
- *Request public key* digunakan oleh *client* setelah *handshaking* untuk meminta kunci publik dari *server*. *Request* ini harus dikirim sebelum autentikasi dilakukan.

c. Response Public Key

- *length prefix*: 1 + X
- *message ID*: 2
- *payload*: <*public key*>
- arah komunikasi: *server* → *client*
- *Response public key* dikirimkan oleh *server* setelah *client* mengirimkan pesan. Panjang X ini adalah panjang *public key*. Pada protokol ini panjang kunci publik yang digunakan adalah 512 bit sehingga nilai X adalah 64 *byte*.

d. Send Authentication

- *length prefix*: 1 + X + Y
- *message ID*: 3
- *payload*:
  <*username*: UTF><*encrypted password*>
- arah komunikasi: *client* → *server*
- *Send Authentication* digunakan untuk mengirimkan *username* dan *password* untuk memulai sinkronisasi. *Username* ini dikirimkan dalam format yang ter-*encode* dengan UTF sepanjang X *byte*. Dan *password* ini adalah *password* yang sudah terenkripsi dengan *public key* dari *server* sepanjang Y *byte* menggunakan algoritma RSA.

e. Response Authentication

- *length prefix*: 2
- *message ID*: 4
- *payload*:
  <*status code*: 1 *byte*>
- arah komunikasi: *server* → *client*
- *Response authentication* dikirim oleh *server* setelah *client* mengirimkan *message* authentication. Isi dari pesan ini berupa *status code* berformat 1 *byte* yang bisa bernilai sebagai berikut:
  - 1: *authentication* berhasil
  - 2: *authentication* gagal

  Apabila *authentication* berhasil maka *client* bisa memulai sinkronisasi.

f. Send Logout

- *length prefix*: 1
- *message ID*: 5
- *payload*: kosong (tidak ada *payload*)
- arah komunikasi: *client* → *server*
- *Send logout* dikirim oleh *client* untuk mengakhiri koneksinya. Setelah dilakukan logout ini maka proses sinkronisasi akan dihentikan.

g. Send Key

- *length prefix*: 1+ X
- *message ID*: 6
- *payload*: <kunci RC4>
- arah komunikasi: *client* → *server*
- *Send key* dikirim oleh *client* setelah *response authentication* berhasil diterima dari *server*. Awalnya *client* harus membangkitkan kunci untuk algoritma RC4. Kunci yang dipakai sepanjang 128 bit. Kemudian kunci tersebut dienkripsi terlebih dahulu dengan kunci publik dari *server* dengan algoritma RSA. Hasilnya sebanyak X *byte* akan dikirimkan sebagai *payload*.

h. Request List File dan Folder

- *length prefix*: 1
- *message ID*: 7
- *payload*: kosong (tidak ada *payload*)
- arah komunikasi: *client* → *server*
- *Request list file* dan *folder* dikirim oleh *client* untuk mendapatkan response berupa seluruh *file* dan *folder* yang terdapat pada *repository server*.

i. Response List File dan Folder

- *length prefix*: 1 + X
- *message ID*: 8
- *payload*:
  <*path*: UTF><*time modified*: 8*byte*><tipe: 1 *byte*>[<SHA *file*: 20 *byte*>]
  <*path*: UTF><*time modified*: 8*byte*><tipe: 1 *byte*>[<SHA *file*: 20 *byte*>]
  :
  :
  <diakhiri oleh *text* UTF berupa "/">
- *Response list file* dan *folder* dikirim oleh *server* setelah menerima *request list file* dan *folder* dari *client*. *Response* ini berupa daftar seluruh *file* dan *folder* yang terdapat pada *repository server*. Format masing-masing *file* atau *folder* yaitu terdiri dari:
  - *path* yaitu *path* atau lokasi *file* atau *folder* relatif terhadap *folder repository*, berformat UTF
  - *time modified* yaitu waktu terakhir perubahan pada *file* atau *folder*, bertipe data *long* sepanjang 8 *byte*. Waktu ini dihitung berdasarkan *epoch time* (1 Januari 1970 00:00).
  - tipe yaitu berisi apakah *file* atau *folder*. Apabila berupa *file* maka bernilai 0, sedangkan apabila berupa *folder* maka bernilai 1.
  - SHA yaitu nilai SHA-1 dari *file* sepanjang 20 *byte*. SHA ini dikirimkan hanya apabila bertipe *file*. Sedangkan apabila bertipe *folder*, tidak perlu dikirimkan.

  Akhir dari response ini berupa *text* UTF berisi *text* "/" (*Slash*). Akhir ini untuk menandakan bahwa daftar seluruh *file* dan *folder* telah dikirim.

j. Request File

- *length prefix*: 1 + X
- *message ID*: 9
- *payload*: <*path file*: UTF>
- arah komunikasi: *client* → *server*
- *Request file* dikirim oleh *client* untuk mengunduh suatu *file* dari *server*. Nantinya *server* akan membalas dengan *message send file*. *Payload*-

nya berupa *path file* yang ingin diunduh dengan *path* relatif terhadap direktori *repository*. *Path file* ini dikirim dalam bentuk ter-*encode* dengan UTF sepanjang X *byte*.

k. *Send File*
- *length prefix*: 1 + X + 8 + Y
- *message ID*: 10
- *payload*:
  <*path file*:UTF><*time modified*: 8 *byte*><*content file*>
- arah komunikasi: *server* → *client* dan *client* → *server*
- *Send file* dikirim oleh *client* apabila akan mengunggah *file* ke *server*, sedangkan *send file* dikirim oleh *server* sesudah *server* menerima *message request file*. *Payload*-nya berupa path *file* yang dikirim berformat UTF sepanjang X *byte* diikuti waktu terakhir modifikasi *file* berformat *long* sepanjang 8 *byte*, kemudian diikuti isi *file* sepanjang ukuran *file* yaitu Y *byte*.

l. *Request For Delete File*
- *length prefix*: 1 + X
- *message ID*: 11
- *payload*: <*path file*: UTF>
- arah komunikasi: *client* → *server*
- *Request for delete file* dikirim oleh *client* untuk meminta *server* menghapus suatu *file* dari *repository* di *server*. *Payload*-nya sama seperti *message request file*.

m. *Request List Modification File*
- *length prefix*: 1 + X
- *message ID*: 12
- *payload*: <*path file*: UTF>
- arah komunikasi: *client* → *server*
- *Request list modification file* dikirim oleh *client* untuk meminta daftar waktu modifikasi *file* yang diminta. Nantinya balasan dari *message* ini digunakan untuk mengganti versi suatu *file* ke versi *file* pada waktu tertentu. *Payload*-nya sama seperti *message* request *file*.

n. *Response List Modification File*
- *length prefix*: 1 + X
- *message ID*: 13
- *payload*:
  <*time modified*: 8 *byte*>
  :
  :
  <*time* 0>
- arah komunikasi: *server* → *client*
- *Response list modification file* dikirim oleh *server* setelah menerima *message request list modification file* dari *client*. *Payload*-nya berupa daftar waktu modifikasi yang pernah dilakukan pada *file* yang diminta. Waktu modifikasi berformat long sepanjang 8 *byte*. *Payload* ini diakhiri dengan waktu dengan nilai 0 yang bertipe sama seperti diatasnya yaitu *long*.

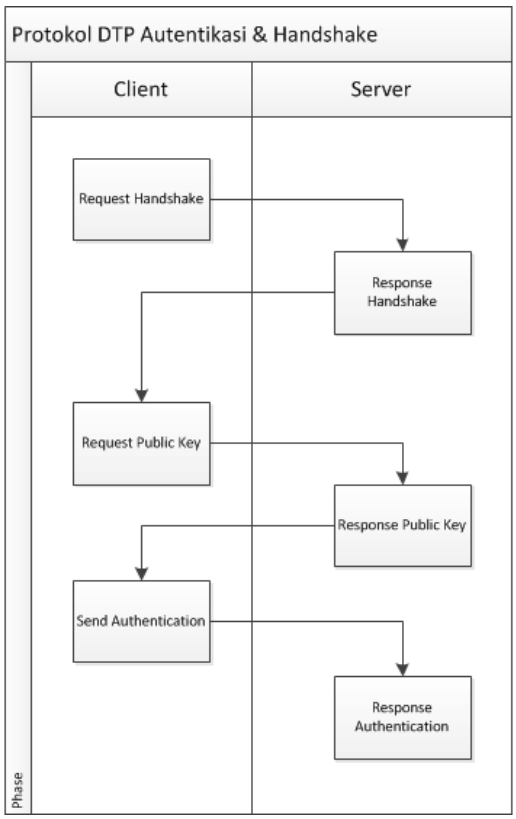
o. *Request For Revert*
- *length prefix*: 1 + X + 8
- *message ID*: 14
- *payload*:
  <*path file*: UTF><*time modified for revert*: 8 *byte*>
- arah komunikasi: *client* → *server*
- *Request for revert* dikirimkan oleh *client* untuk meminta *server* mengganti *file* yang diminta dengan versi *file* pada waktu tertentu yang dicantumkan pada *payload*. *Payload*-nya berupa path *file* yang berformat UTF, diikuti versi waktu modifikasi yang diminta berformat *long* sepanjang 8 *byte*. Setelah *message* ini dikirim maka *server* akan membalas dengan *message send file*, yaitu mengirimkan isi *file* sesuai dengan versi yang diminta.

p. *Send Directory*
- *length prefix*: 1 + X + 8
- *message ID*: 15
- *payload*:
  <*path file*: UTF><*time modified*: 8 *byte*>
- arah komunikasi: *client* → *server*
- *Send directory* dikirim oleh *client* untuk meminta *server* membuat direktori baru sesuai dengan path yang diminta. *Payload*-nya berupa path *file* berformat UTF diikuti waktu modifikasi direktori berformat *long* sepanjang 8 *byte*.

q. *Error Message*
- *length prefix*: 1
- *message ID*: 99
- *payload*: kosong (tidak ada *payload*)
- arah komunikasi: *server* → *client*
- *Error message* dikirim oleh *server* apabila terdapat kesalahan pada *message* yang dikirim oleh *client*. Kesalahan tersebut bisa berupa *message ID* salah atau *path file* yang diminta tidak ditemukan.

C. Interaksi dan Diagram Alir

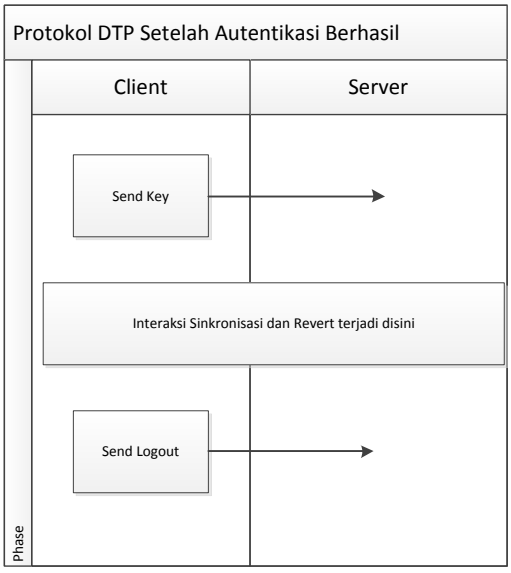
a. Interaksi dan Diagram Alir pada Autentikasi dan *Handshake*



Gambar 2 Interaksi dan Diagram Alir Autentikasi dan *Handshake*

Interaksi ini dilakukan pada saat pertama kali *client* terkoneksi dengan *server*. Interaksi ini diakhiri setelah *client* menerima *response* autentikasi.

b. Interaksi Setelah Autentikasi Berhasil

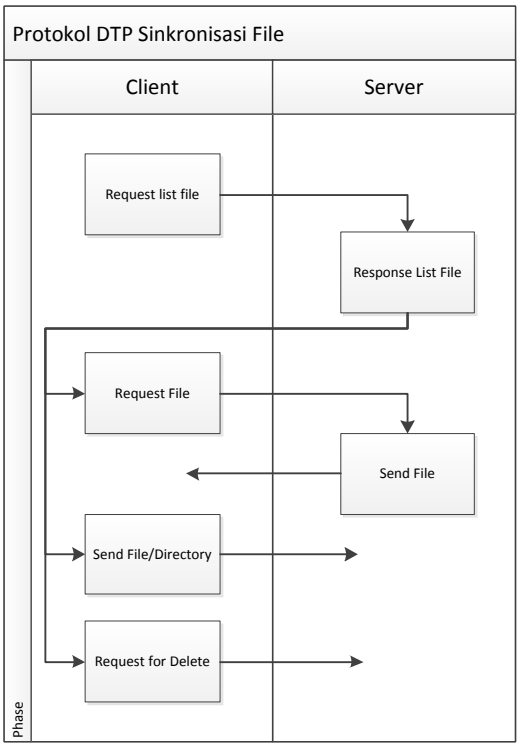


Gambar 3 Interaksi dan Diagram Alir Setelah Autentikasi Berhasil

Interaksi ini dilakukan setelah *client* menerima *message response* berhasil autentikasi. Pada interaksi ini awalnya *client* harus mengirimkan kunci RC4 ke *server*. Apabila kunci ini selesai dikirim maka komunikasi yang terjadi setelahnya adalah komunikasi yang aman sehingga semua pesan yang dikirim baik oleh *client* maupun oleh *server* harus terenkripsi.

Interaksi ini diakhiri dengan *message send logout* yang dikirimkan oleh *client*. Selanjutnya koneksi antara *client* dan *server* akan ditutup.

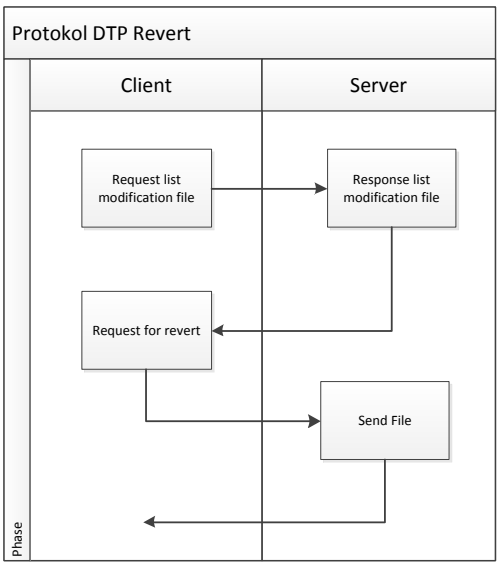
c. Interaksi Sinkronisasi



Gambar 4 Interaksi dan Diagram Alir Sinkronisasi

Interaksi ini dilakukan untuk mensinkronisasikan seluruh isi *repository*. Interaksi ini dilakukan secara periodik selama koneksi masih berlangsung dan *message send logout* belum dikirim oleh *client*. Interaksi ini diawali dengan *client* yang mengirim *request list file*. Seluruh komunikasi dalam interaksi ini adalah komunikasi yang aman.

d. Interaksi Untuk *Revert File*



Gambar 5 Interaksi dan Diagram Alir untuk *Revert File*

Interaksi ini dilakukan apabila *client* ingin

mengganti isi *file* dengan versi isi *file* pada waktu tertentu. Komunikasi dalam interaksi ini juga dilakukan secara aman.

D. Finite State Machine Server dan Client

a. *Finite State Machine* di *Server*

```
START
Accept Client
Read Handshake
Send Handshake
Read Request Public Key
Generate Public Key
Send Public Key
While(not haslogin) do
  haslogin = Read Authentication
  If (haslogin) then
     Send Success Response Authentication
  Else
     Send Failed Response Authentication
While(not logout)
  Read message
  If (message ID == Message Logout) then
     logout = true
  Else if(message ID == Message Request List File)
  then
     Send List File
  Else if(message ID == Message Request File) then
     Send File to client
  Else if(message ID == Message Send File) then
     Read File from client
     Move current repository file to trash
     Save file which read from client to repository
  Else if(message ID == Message Request For Delete)
  then
     Delete File
  Else if(message ID == Message Request List
  Modification) then
     Send List Modification
  Else if(message ID == Message Request For Revert)
  then
     Revert File
     Send Reverted File
  Else if(message ID == Message Send Key) then
     Save key
     Create Secure Stream Communication
  Else if(message ID == Message Send Directory)
  then
     Create Directory
  Else
     Send Error Message
END
```

b. *Finite State Machine* di *Client*

```
START
Connect to server
Send Handshake
Read Handshake
Send Request Public Key
Read Response Public Key
Send Authentication
responselogin = Read Response Authentication
If (responselogin == Fail Response) then
   Close connection
   Exit
Generate RC4 Key
Send Key
Create Secure Stream Communication
While (not logout) do
   Save current list files in client
   Send request list
   Read response list
   Synchronize repository by compare list files in
   client with list files in server
   Delay (Period)
END
```

E. Algoritma Sinkronisasi File

Algoritma dalam mensinkronkan antara *file* di *server* dengan *file* di *client* dilakukan pada *client*. Pada awalnya *client* menyimpan *state* berupa seluruh daftar *file* dan *folder* yang berada pada *repository client*. Kemudian *client* mengirimkan *message request list file* dan *folder* ke *server*, dan *client* akan menerima *response list file* dan *folder* yang berada pada *repository server*. Setelah itu masing-masing dari *file* dan *folder* yang berada pada *client* maupun yang berada pada *server* dibandingkan. Algoritma pembandingannya sebagai berikut:

Membandingkan semua *file* yang terdapat pada *client*:

```
Foreach file and folder in client do
  If (server has this file) then
     If (isfile and SHA in client != SHA in server) then
        If (time modified in server > in client) then
           Request File //Download File
           Read File
        Else
           Send File //Upload File
  Else //server hasn't this file
     If (time modified in client <= last time synchronize)
     then
        If (isfile) then
           If (old client repository has file with same SHA)
           then
              If (old path != current path) then
                 If (current client repository has old path)
                 then
                    //conclusion: duplicated file
                    Send File //Upload File
                 Else
                    //conclusion: renamed file
                    Send File //Upload File
                    Request For Delete (old path)
              Else //this is old file which has deleted
                 Delete File
           Else //this is new file from this client
              Send File
        Else //is directory
           If (old client repository has directory with same
           time modified) then
              If (old path == current path) then
                 If (current client repository has old path)
                 then
                    //conclusion: duplicated directory
                    Send Directory //Upload Directory
                 Else
                    //conclusion: renamed directory
                    Send Directory //Upload Directory
                    Request For Delete (old path)
              Else //this is old directory which has deleted
                 Delete Directory
           Else //this is new directory from this client
              Send Directory
     Else //last modified in client > last time synchronize
        //this is new directory from this client
        If (isfile) then
           Send File
        Else
           Send Directory
```

Membandingkan semua *file* yang terdapat pada *server* dan tidak terdapat pada *client*:

```
Foreach file and folder in server do
  If (client hasn't this file) then
     If (time modified in server > last time synchronize)
     then
        If (isfile) then
           Request File //Download file
           Read File
        Else
           Create Directory
     Else
        If (isfile) then
           If (old server repository has file with same
           SHA) then
              If (old path != current path) then
                 If (current server repository has old
                 path) then
                    //conclusion: duplicated file
                    Request File //Download File
                    Read File
                 Else
                    //conclusion: renamed file
                    Request File //Download File
                    Read File
                    Delete (old path)
              Else //this is old file which has deleted
                 Request For Delete
           Else //this is new file from this server
              Request File //Download File
              Read File
        Else //is directory
           If (old server repository has directory with
           same time modified) then
              If (old path == current path) then
                 If (current server repository has old
                 path) then
                    //conclusion: duplicated directory
                    Create Directory
                 Else
                    //conclusion: renamed directory
                    Create Directory
                    Delete (old path)
              Else //this is old directory which has
              deleted
                 Request For Delete
           Else //this is new directory from this server
              Create Directory
```

Dalam algoritma diatas peran SHA sangat penting diantaranya:

1. Mendeteksi perubahan isi *file* pada *file* dengan *path* yang sama.
2. Mendeteksi apakah terjadi penggantian nama (*renaming*) pada suatu *file*. Penggantian nama dideteksi dengan cara membandingkan nilai SHA *file* dengan semua nilai SHA pada daftar *file* di *repository* yang lama. Apabila terdapat SHA yang sama dan *path*-nya berbeda dan *path* yang lama tidak ada pada daftar *file* pada current *repository* maka dapat dipastikan *file* tersebut telah diubah namanya.
3. Mendeteksi apakah terjadi penduplikasian *file*. Pendeteksian duplikasi *file* hampir sama caranya dengan mendeteksi penggantian nama, hanya kondisinya yaitu apabila terdapat SHA yang sama dan *path*-nya berbeda dan *path* yang lama masih ada pada daftar *file* pada current *repository*.

IV. IMPLEMENTASI

Implementasi protokol ini dengan membuat tiga program yaitu program *server*, *web* untuk registrasi, dan program *client*. Lingkungan implementasi program *server* dan program *client* yaitu dengan menggunakan bahasa pemrograman Java dengan *platform* J2SE. Sedangkan lingungan implementasi *web* dengan bahasa pemrograman PHP dan menggunakan *web server* Apache.

A. Program Server dan Web pada Server

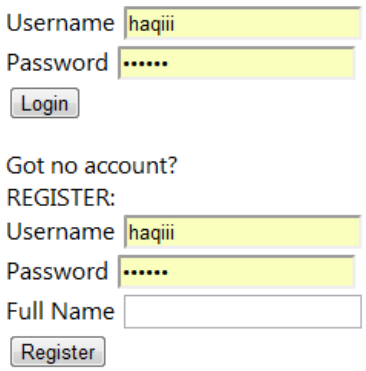
Progam *server* berupa program *console*. *Server* mampu menangani lebih dari satu *client* dalam waktu bersamaan. Pada implementasi ini *server* menggunakan port 2000. *Server* mempunyai *database* untuk menyimpan data-data autentikasi dan data-data *file* yang telah terhapus/terganti.

Model *database*-nya sebagai berikut:

a. Tabel user
   - id
   - fullname
   - username
   - password

b. Tabel deleted_file
   - id
   - path_asli
   - time_deleted
   - name_in_repositori

Sedangkan *web* berfungsi sebagai tempat registrasi akun untuk sinkronisasi. Pada *web* ini user juga bisa melihat *file*-*file* yang terdapat pada *repository*-nya.

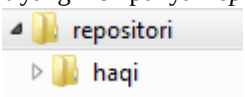


Gambar 6 Tampilan Web

Gambar 7 Tampilan *Repository* pada Web

Pada *server* semua *repository* terdapat pada suatu

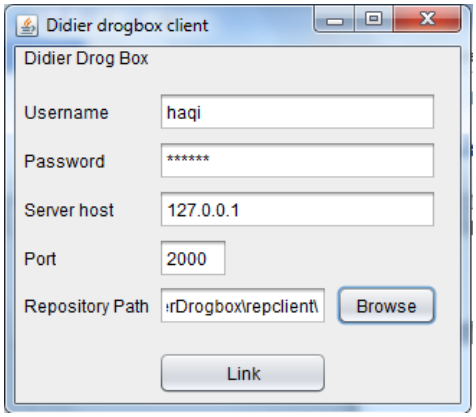
direktori. Di dalam direktori tersebut *file-file repository* punya *client* disimpan pada direktori dengan nama sesuai username dari *client* yang mempunyai *repository* tersebut.



Gambar 8 Struktur *Repository*

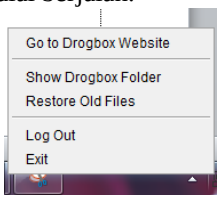
B. Program Client

Progam *client* berupa program *desktop* yang mempunyai GUI. Tampilan awal dari program *client* sebagai berikut.



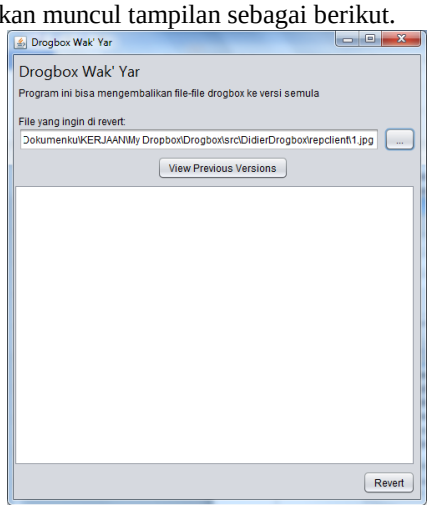
Gambar 9 Tampilan Awal Program *Client*

Pada layar diatas *client* diwajibkan untuk mengisikan *username*, *password*, beserta direktori tempat *repository* di *client* untuk memulai sinkronisasi. Setelah *login* (dengan menekan tombol *Link*) maka program akan muncul pada *system tray* pada sistem operasi dan sinkronisasi akan mulai berjalan.



Gambar 10 System Tray Program *Client*

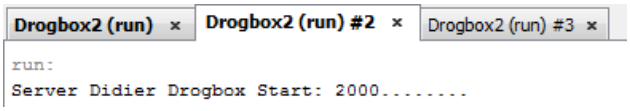
Untuk melakukan *revert file* maka user harus memilih menu "*Restore Old File*s" pada menu klik kanan di *system tray* dan akan muncul tampilan sebagai berikut.



Gambar 11 Tampilan Menu Revert *File*

V. PENGUJIAN DAN ANALISIS KEAMANAN

Pengujian dilakukan dengan menjalankan program *server* terlebih dahulu.

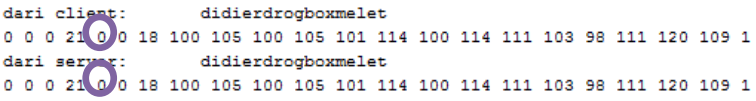


Gambar 12 *Server* Dimulai

Kemudian program *client* dijalankan dan penulis mengisikan *username* dan *password* yang sudah terdaftar pada *server* beserta *path repository*nya. Setelah *login* maka proses sinkronisasi akan dimulai.

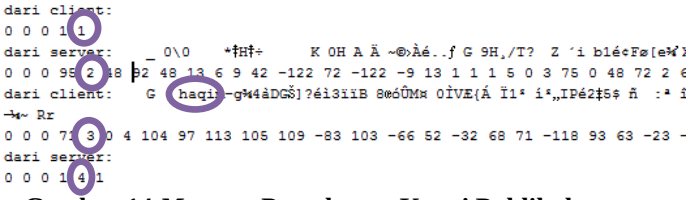
Penulis membuat suatu program simulator (semacam program *tunneler* antara *client* dan *server*) untuk menangkap paket data yang dikirimkan antara *client* dan *server*. Simulator ini akan menampilkan isi paket data dalam bentuk representasi karakter beserta representasi angka per *byte*.

Ketika *client* dan *server* mulai terkoneksi maka simulator akan menampilkan pesan apa saja yang dikirim oleh *server* maupun *client*.



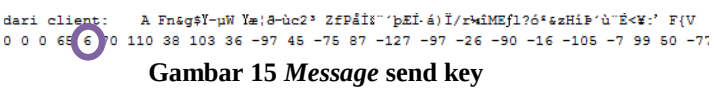
Gambar 13 *Message Handshake*

Pesan diatas adalah pesan *handshake*, pesan *handshake* dapat diketahui dari nilai *byte* pada urutan ke-5 yaitu *message ID* nomor 0 (ingat format pesan diawali dengan *length-prefix* sebanyak 4 byte diikuti *message ID*). Pada representasi karakter terlihat *text* "didierdrogboxmelet" dikirim.



Gambar 14 *Message* Pertukaran Kunci Publik dan Autentikasi

Pesan diatas adalah pesan setelah *handshake* yaitu pesan *request public key* (*message ID* nomor 1) diikuti pesan *response public key* (*message ID* nomor 2) diikuti pesan *send authentication* (*message ID* nomor 3) diikuti pesan *response authentication* (*message ID* nomor 4). Terlihat bahwa terdapat *text* username yaitu "haqi" yang tidak terenkripsi sedangkan pengiriman password (setelah username) tidak dapat dibaca, karena telah terenkripsi oleh kunci publik. Kemudian pada akhir pesan *server* mengirimkan *response* berhasil autentikasi (kode status 1).



Gambar 15 *Message* send key

Pesan diatas adalah pesan kunci RC4 yang dikirimkan

oleh *client* (*message ID* nomor 6). Kunci RC4 dikirimkan sudah dalam bentuk terenkripsi oleh kunci publik.

```
dari client: eƒŴäõ
101 -125 87 -28 -10
dari server: eƒ
101 -125
dari server: WäùØèN·>,1
87 -28 -7 -40 -24 78 -106 62 -126 49
dari server: Š›ê÷i¾iÛÁ
-118 -101 -22 -9 -17 -66 -20 -37 -63
dari server: ,A "6Ò íùPK˜«Ðè8[u^
44 65 0 -109 54 -46 32 -19 -7 80 75 -104 13 -85 -48 -24 -16 91 117 94
dari server: \A mê ÜØZ¾t‡k .uZ
92 65 -99 109 -22 11 -36 -40 90 -66 116 -122 107 11 46 117 90
```

Gambar 16 *Message Request List File* dan *Response List File* (1)

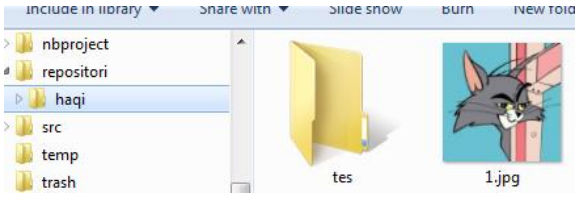
Dari pesan diatas tidak dapat diketahui apa pesan yang dikirim karena nilai *byte* pada urutan ke-5 yang dikirim oleh *client* bernilai -10 padahal dalam protokol ini tidak ada *message ID* dengan nilai -10. Hal ini dapat diartikan bahwa pesan yang dikirimkan oleh *client* telah terenkripsi dengan algoritma RC4. Tetapi dengan melihat urutan pengiriman pesan dapat diketahui pesan diatas adalah pesan *request list file* yang dikirim oleh *client* dan kemudian dari *server* dibalas dengan *message response list file*. Seharusnya dari *message response list* yang dikirim oleh *server* kita bisa melihat nama *folder* atau *file* pada representasi karakter (sesuai dengan format pengiriman *response list file* dan *folder*). Akan tetapi hal itu tidak bisa dilihat karena semua pesan yang dikirim oleh *server* juga terenkripsi.

```
dari client: Øí ¹S
-40 -19 127 -71 83
dari server: ÜÃÝ¼o)Rl2!Kzmß ßß °1Ñ ô8яÉ5Ì  ‡Ñ?"Q°-‹èÑk )ÃñÃ]Jë :è0ýN
-36 -61 -35 -66 111 41 82 108 50 33 75 122 109 -33 24 -118 -118 12 -80 48
dari server: ê],
-22 93 -126
```

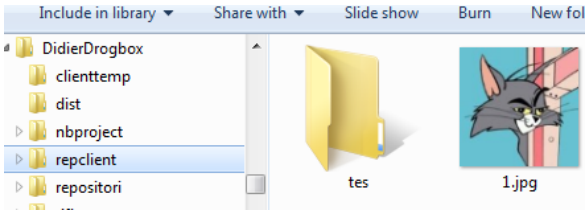
Gambar 17 *Message Request List File* dan *Response List File* (2)

Berdasarkan urutan pengiriman, pesan diatas adalah pesan *request list file* yang dikirim oleh *client* diikuti pesan *response list file* yang dikirim oleh *server*. Pada pesan *request list file* diatas terlihat jumlah *byte*-nya sama dengan jumlah *byte plaintext* ataupun jumlah *byte* pesan *request list* sebelumnya yaitu sebesar 5 *byte*. Akan tetapi meskipun panjangnya sama tetapi isi nilai dari pesannya berbeda dengan pesan *request list file* sebelumnya (101 -125 87 -28 -10 dengan -40 -19 127 -71 83). Padahal kalau pesan tersebut didekripsi dalam bentuk *plaintext* nilainya sama. Hal ini menunjukkan bahwa algoritma *stream cipher* RC4 telah bekerja dengan benar. Dengan demikian para *sniffer* yang ingin mengetahui isi pesan akan kesulitan dalam menebak-nebak isi pesan karena isi dari *ciphertext* berbeda-beda meskipun dengan *plaintext* yang sama.

Untuk pengujian kesamaan isi *repository* antara *repository server* dan *repository client*, program ini dapat menyamakan isinya dengan baik.



Gambar 18 Isi *repository user* haqi pada *server*



Gambar 19 Isi *repository* pada *client*

Dari dua gambar diatas terlihat bahwa isi *repository* dari *client* sama dengan isi *repository user* haqi pada *server*. Sehingga dapat disimpulkan bahwa algoritma sinkronisasi telah berjalan dengan baik.

VI. KESIMPULAN

Protokol *Drogbox Transfer Protocol* (DTP) dapat digunakan oleh masyarakat di dunia sebagai protokol untuk sinkronisasi suatu *repository*. Protokol ini dapat digunakan oleh banyak *client* secara bersamaan sehingga masing-masing dari *client* tersebut memiliki repository yang sama dan konsisten.

Dengan kelebihan dari protokol ini yaitu komunikasinya secara aman tanpa menghilangkan konsistensi data, protokol ini dapat dipakai oleh pihak-pihak yang sangat memperhatikan keamanan datanya contohnya seperti pemerintah, polisi, intelejen, dll. Protokol ini juga dapat dipakai oleh kepentingan internal dalam kolaborasi antar anggota kelompok. Protokol ini juga dapat dipakai oleh suatu sistem *server* yang terdiri dari bermacam-macam *sub-server* dalam mengkonsistenkan data antar *sub-server* tersebut.

Untuk pengembangan ke depan, protokol ini dapat diberi fitur-fitur keamanan lain, yaitu misalkan algoritma *stream cipher* dapat dipilih oleh pengguna sehingga tidak hanya algoritma RC4 saja yang disediakan oleh protokol ini. Ini akan menambah tingkat keamanan dari protokol ini. Protokol ini juga dapat dikembangkan sehingga dapat memeriksa keotentikan dari pesan yaing dikirim menggunakan *Message Authentication Code* (MAC).

Demikian kesimpulan dari makalah yang penulis buat apabila ada kesalahan dalam tulisan makalah ini mohon dimaklumi dan dimaafkan.

REFERENCES

[1] H. S. Nugroho, "Implementasi Sinkronisasi *File* Antar Dua *Server* Menggunakan Protokol Messaging zeromq", Bab 1, http://digilib.itb.ac.id/public/ITS-Undergraduate-14272-chapter1pdf.pdf, Surabaya: Institut Teknologi Sepuluh November, 2010.

[2] Noprianto, "Sinkronisasi File dengan rsynch", http://ilmukomputer.org/2007/03/28/sinkronisasi-*file*-dengan-rsynch/, 2007.

[3] R. Munir, “Algoritma RSA”, http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Algoritma%20RSA.ppt , 2011.
[4] R. Munir, “Review Beberapa Algoritma Kriptografi Modern”, http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Review%20Beberapa%20Algoritma%20Kriptografi%20Modern.ppt, 2011.
[5] R. Munir, “SHA”, http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/SHA.ppt , 2011.
[6] A. Baihaqi, “Penggunaan Steganografi pada Gambar dalam Perancangan Protokol Web Baru”. http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Makalah1/Makalah1-IF3058-Sem1-2010-2011-033.pdf, 2011.
[7] The BitTorrent Protocol Specification, http://www.bittorrent.org/beps/bep_0003.html, 2008.
[8] “What is stateless? Definition from WhatIs.com”, http://whatis.techtarget.com/definition/0,,sid9_gci213051,00.html, 2005.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Mei 2011

Achmad Baihaqi
13508030