Otentikasi File Dengan Algoritma Kriptografi SHA-1 Menggunakan Python Dan Pycrypto

Habib Abdur Rahman[1], Rodiah[2]

[1,2] Universitas Gunadarma, Jl.Margonda Raya 100 Pondok Cina Depok
[1]hbb.arh@gmail.com
[2]rodiah@staff.gunadarma.com

ABSTRAK

Saat ini pemanfaatan teknologi komputer sebagai sarana bertukar pesan semakin sering digunakan. Sejalan dengan hal tersebut mulai terjadi penyalahgunaan oleh pihak-pihak yang tidak berwenang dengan memanipulasi file yang ingin dikirimkan, sehingga dapat menimbulkan kesalahpahaman. Salah satu cara untuk mencegah hal tersebut adalah memanfaatkan fitur keutuhan data oleh sistem kriptografi. Pengirim pesan maupun penerima pesan dapat memastikan keutuhan dan keaslian pesan dengan memanfaatkan fungsi hash dalam kriptografi. Fungsi hash merupakan sebuah fungsi dengan masukan sebuah pesan dan keluarannya adalah nilai hash atau sidik pesan (message fingerprint). Sidik pesan adalah nilai yang bersifat unik, sehingga dapat dijadikan acuan untuk memeriksa keutuhan dan keaslian pesan. Berdasarkan hal tersebut, proses otentikasi file dapat dilakukan dengan membandingkan nilai hash yang dihasilkan oleh file yang dimiliki pengirim dan penerima. Apabila nilai hash kedua file sama maka dapat dipastikan file tersebut adalah utuh dan sebenarnya, sedangkan apabila nilai hash kedua file berbeda maka dapat dipastikan file tersebut berbeda atau telah dimanipulasi. Implementasi algoritma SHA-1 untuk otentikasi file ini diharapkan dapat menghindarkan tindakan manipulasi terhadap file yang akan dikirimkan terutama lewat jaringan.

Kata kunci: Fungsi Hash, Otentikasi File, Python, SHA-1

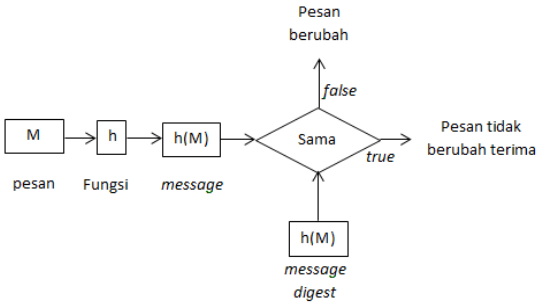
1. Pendahuluan

Seiring dengan kemajuan teknologi informasi komputer maka tindakan memanipulasi file penting pun semakin mungkin untuk dilakukan. Otentikasi merupakan proses untuk membuktikan kebenaran sesuatu yang bertujuan agar kebenaran suatu informasi dapat diakui. Sebagai contoh saat menggunakan layanan email, dimana pengguna akan diminta memasukkan *user*name dan *password* sehingga penyedia layanan email dapat memastikan bahwa yang mengakses akun email tersebut adalah benar-benar orang yang tepat. Penerapan dari otentikasi juga berlaku untuk otentikasi file atau tulisan. Otentikasi file atau tulisan menjadi penting untuk dilakukan mengingat pertukaran file semakin mudah dan sering dilakukan. Otentikasi file memungkinkan seseorang dapat memastikan apakah file yang diterima merupakan file asli yang belum berubah isinya pada saat dikirim oleh pengirimnya hingga file tersebut diterima (L.H. Nguyen and A.W. Roscoe, 2008)

Penulisan ini akan mengimplementasikan algoritma kriptografi dalam menyelidiki keaslian dari file teks yang diterima dengan file teks asli dengan cara mencocokan nilai *hash* dari kedua file tersebut. Salah satu layanan keamanan data yang dapat diwujudkan oleh sistem kriptografi adalah keutuhan data. Pengirim pesan maupun penerima pesan dapat memastikan keutuhan dan keaslian pesan dengan memanfaatkan fungsi *hash* dalam kriptografi. Proses otentikasi file dapat diselidiki dengan membandingkan nilai *hash* dari file yang asli dengan nilai *hash* dari file yang telah dikirimkan ke pihak lain.

Fungsi *hash* merupakan sebuah fungsi yang masukkannya adalah sebuah pesan dan keluarannya adalah sebuah sidik pesan (*message fingerprint*) (V. Shoup,1996). Sidik pesan disebut juga *message digest* atau nilai *hash* yang didapat dari tiap file akan berbeda-beda satu dengan yang lainnya, sehingga nilai *hash* tersebut bersifat unik dan dapat dijadikan acuan untuk memeriksa keutuhan data dan otentikasi file (H. Krawczyk, 1995)

Sebagai contoh misalnya *M* merupakan pesan dan *h* adalah fungsi *hash,* maka *y= h(M)* disebut dengan sidik pesan *x* atau sering juga disebut dengan *message digest*. Sebuah *message digest* umumnya berukuran pendek yaitu sekitar 160 bit. Gambar 1 menunjukkan bagan pengujian keutuhan pesan dengan fungsi hash (Sadikin, 2012)



Gambar 1. Pengujian keutuhan pesan dengan fungsi *hash* [3]

Pada gambar 1, sebelum pesan *M* disebarkan/dikirimkan sebuah *message digest* *y*lama = *h*(*M*) disimpan sebagai acuan. Misalnya didapatkan kembali *M'* setelah disebarkan /dikirim apabila ingin menguji apakan *M = M'* hitung kembali *message digest* baru *y*baru= *h*(*M*') maka dapat disimpulkan bahwa pesan tidak berubah bila *y*lama = *y*baru (Sadikin, 2012).

Proses otentikasi file melalui pemanfaatan nilai *hash* dilakukan dengan cara membandingkan nilai *hash* dari kedua file. Apabila nilai *hash* yang didapat dari kedua file adalah sama, maka file tersebut dapat dipastikan keutuhan dan otentikasinya, sedangkan jika nilai *hash* yang diperoleh dari kedua file tersebut berbeda maka dapat dipastikan bahwa file yang diterima tidak utuh atau telah dimanipulasi (V.S. Pless and W. Huffman. 1998). Implementasi dari algoritma ini dapat digunakan untuk mengetahui keaslian suatu file.

Implementasi dari penerapan sistem kriptografi fungsi *hash* untuk keperluan otentikasi file ini diharapkan dapat menghindarkan tindakan manipulasi file yang akan dikirimkan terutama lewat jaringan. Penulis menggunakan file teks murni berekstensi .txt dengan ukuran pesan kurang dari 2^{64} bit menggunakan *Secure Hash Algorithm* (SHA-1) menggunakan python dan pycrypt.

2. Pembahasan

Dalam pembuatan aplikasi Otentikasi File tahap pertama yang dilakukan adalah membuat algoritma alur kerja dari aplikasi dengan *activity diagram* seperti dapat dilihat pada gambar 2. Secara umum aplikasi akan meminta input file 1 dan file 2, untuk kemudian setelah file diinput maka program akan menghitung nilai *hash* dari masing-masing file dan menyimpan hasilnya ke dalam output file untuk masing-masing nilai *hash.* Proses selanjutnya yaitu program akan menampilkan nilai *hash* untuk pada komponen entri untuk masing-masing file, kemudian saat perintah pencocokan nilai *hash* dieksekusi maka program akan membandingkan kedua nilai tersebut menampilkan hasilnya pada komponen entri untuk hasil.

Gambar 2 *Activity diagram* aplikasi Otentikasi File

2.1. Algoritma SHA-1 dengan PyCrypt

Algoritma aplikasi Otentikasi File untuk mendapatkan nilai hash terdiri atas langkah-langkah :

1. Mencari file yang akan diotentikasi dengan cara membuka jendela *open* file dan juga menampilkan *path* dari lokasi file yang dipilih. File yang dapat dipilih dalam hal ini dibatasi untuk file yang berekstensi txt saja. Pada aplikasi ini terdapat dua buah fungsi sejenis yaitu onCari dan onCari2 yang berfungsi masing-masing untuk membuka file 1 dan file 2 melalui perintah :

   ```
   namafile = askopenfilename(filetypes=[('', '*.txt')])
   if namafile :
           self.entFile.delete(0, END)
           self.entFile.insert(END, namafile)
   ```

   Fungsi yang dideklarasikan akan menjalankan perintah untuk membuka jendela *open file dialog*, dalam hal ini penulis membatasi jenis file yang dapat dibuka hanya yang berekstensi txt. Apabila file terbuka, maka judul file akan ditampilkan pada komponen entri.

2. Penghitungan nilai *hash* pada aplikasi ini menerapkan fungsi *hash* SHA-1 yang sudah disediakan pada library PyCrypto. Perintah x = open(nmfile, 'r') akan membuka file. File yang telah dibuka, akann dibaca terlebih dahulu dimana y= x.read().Nilai hash didapatkan secara otomatis dengan pycrypt melalui perintah :

   ```
           hash_this = ''.join(y)
           h= SHA.new()
           h.update(hash_this)
   ```

3. Kemudian program akan membaca isi file yang selanjutnya akan disimpan kedalam sebuah variable bernama *hash*_this. Program akan menjalankan perintah enkripsi satu arah untuk mendapatkan nilai *hash* file tersebut dimana :

```
hasil = h.hexdigest()
z = open('D:/Hasil.har','w')
a = z.write(hasil)
z.close()
```

Secara matematis fungsi hash *h* dapat diterapkan pada blok data dengan ukuran berapa saja dimana fungsi *h(X)* mudah dihitung untuk setiap nilai *X* yang diberikan dan untuk setiap *X* yang diberikan tidak mungkinmencari Y^1 *X* sedemikian sehingga *h(X) =h(Y)* seperti didefinisikan pada persamaan (1) dan persamaan (2) :

$$h : X \rightarrow Y, | X | > | Y | \quad (1)$$

Sebagai contoh :

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

$$h : \{0,1\}^* \rightarrow Z_n \quad (2)$$

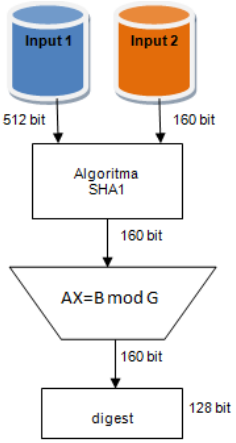
$$h : \{0,1\}^k \rightarrow \{0,1\}^1, k > 1$$

Analogi dari model matematis ini adalah ketika A mengirimkan sebuah file *X,* maka pihak A juga akan menyertakan nilai *Y=hk(X)*. Pihak penerima *X* dan *Y* akan memverifikasi apakah *y=hk(X)* jika benar maka pihak B dapat memastikan bawa file yang diterima tidak mengalami modifikasi. Untuk setiap *X* yang diberikan, tidak mungkin mencari Y^1 *X* sedemikian sehingga *h(Y) = h(X).* Tidak mungkin mencari pasangan *X* dan *Y* sedemikian sehingga *h(X) = h(Y).*

4. Selanjutnya hasil nilai *hash* yang telah dihitung berupa nilai heksadesimal akan disimpan kedalam file hasil.har yang dibuat pada direktori D, selanjutnya hasil tersebut akan ditampilkan pada komponen entri dengan perintah :

```
tampil = open('D:/Hasil.har','r').read()
self.txtFile.delete('1.0', END)
self.txtFile.insert('1.0', tampil)
```

5. Untuk Pesan diberi tambahan untuk membuat panjangnya menjadi kelipatan 512 bit (L x 512) dimana nantinya pesan ini akan dibagi-bagi menjadi blok-blok berukuran 512 bit dan setiap blok diolah. Output yang dihasilkan dari setiap blok digabungkan dengan output blok berikutnya untuk mendapatkan *digest* seperti dapat dilihat pada bagan gambar 3.



Gambar 3. Bagan pengolahan pesan

2.2. Pemeriksaan Nilai Hash

Pada tahapan terakhir akan dibentuk suatu fungsi yang bertujuan untuk memeriksa kecocokan dari nilai *hash* yang dihasilkan oleh file 1 dan nilai *hash* yang dihasilkan oleh file 2 melalui perintah python seperti berikut :

```
if tampil==tampil2:
   self.txtFile3.delete('1.0', END)
   self.txtFile3.insert('1.0', 'Same File')
   tkMessageBox.showinfo ("Notification", "Hash value match", icon= 'info')
else:
   self.txtFile3.delete('1.0', END)
   self.txtFile3.insert('1.0', 'Different File')
   tkMessageBox.showinfo ("Notification", "Hash
   value doesn't match", icon='warning')
```

Apabila nilai *hash* dari kedua file tersebut cocok atau sama maka program akan menampilkan *message box* berupa pemberitahuan yang memberikan informasi bahwa file tersebut adalah asli, sedangkan apabila nilai *hash* kedua file tersebut tidak cocok atau sama maka program akan menampilkan *message box* berupa peringatan yang berisi bahwa file tersebut adalah file yang berbeda. Fungsi ini akan dipanggil ketika *user* menekan *button check* dan berguna untuk mempermudah *user* dalam mencocokan nilai *hash* kedua file tersebut tanpa harus mencocokannya sendiri secara manual.

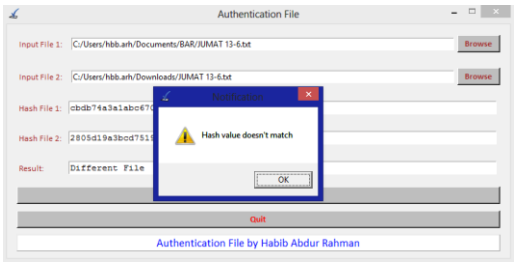
2.3. Hasil dan Uji Coba

Aplikasi otentifikasi file ini terdiri dari sebuah form tunggal yang di dalamnya terdapat *button* untuk *browse* masing-masing file 1 dan file 2, *button check*, *button quit*, kemudian terdapat beberapa *frame* teks untuk menampilkan *path* dari masing-masing file, nilai *hash* dari masing-masing file, dan hasil pencocokan nilai *hash* kedua file tersebut. Pada saat *button browse* untuk input file 1 ditekan maka program akan menampilkan jendela *open* file. Sebagai contoh file yang dipilih adalah file teks JUMAT 13-6.txt yang terdapat pada direktori Libraries/Documents/BAR, maka pada *frame* teks untuk input file 1 akan ditampilkan *path* file tersebut yaituC:/Users/hbb.arh/Documents/BAR/JUMAT 13-6.txt. Setelah memilih file 1 yang akan dipakai, maka program akan langsung menghitung nilai *hash* dari file 1 tersebut.

Seperti yang ditunjukkan oleh gambar 4, nilai *hash* yang didapatkan dari file 1 akan disimpan pada *frame* teks untuk *hash* file 1. Selanjutnya untuk mencari file 2 ditekan maka program akan menampilkan jendela *open* file, setelah itu cari file 2 yang akan digunakan, sebagai contoh teks JUMAT 13-6.txt yang terdapat pada direktori Downloads/BAR, maka pada *frame* teks untuk input file 2 akan ditampilkan *path* file tersebut yaitu C:/Users/hbb.arh/Downloads/JUMAT 13-6.txt. Setelah memilih file 2 yang akan dipakai, maka program akan langsung menghitung nilai *hash* dari file 2 tersebut, dimana nilai *hash* yang didapatkan dari file 2 akan disimpan pada *frame* teks untuk *hash* file 2.

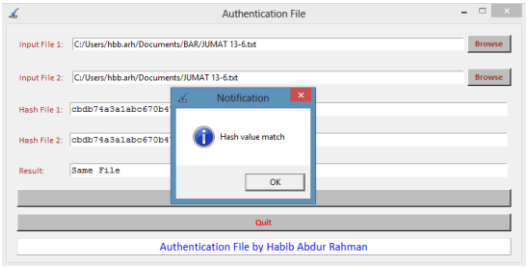
Gambar 4. Nilai *hash* dari file 1 dan file 2

Langkah selanjutnya pada gambar 5 dilakukan dengan menekan tombol *check* maka program akan memeriksa kecocokan dari kedua nilai *hash* tersebut. Dalam hal ini file yang dibandingkan merupakan file yang berbeda dimana telah diumpamakan sebelumnya bahwa file yang asli adalah file pada *folder* BAR sedangkan file yang telah mengalami manipulasi adalah file pada *folder* Downloads, sehingga nilai *hash* dari kedua file tersebut tidak cocok atau sama.



Gambar 5. Peringatan bahwa nilai *hash* tidak cocok

Pengujian untuk kedua file yang sama akan menghasilkan nilai hash yang sama pula, seperti dapat dilihat pada gambar 6.



Gambar 6. Pemberitahuan bahwa nilai *hash* cocok

3. Kesimpulan

Berdasarkan pada hasil uji coba program otentikasi File yang penulis lakukan dapat disimpulkan bahwa aplikasi yang menerapkan algoritma kriptografi SHA-1 ini dapat memudahkan user untuk menyelidiki keaslian suatu file yang dimilikinya dengan file aslinya, sehingga user dapat memastikan keotentikan atau keaslian file yang dimilikinya dan terhindar dari tindakan pemalsuan file oleh pihak-pihak yang tidak memliki wewenang atas isi file tersebut. Pengujian untuk kedua file yang sama akan menghasilkan nilai hash yang sama pula, dalam hal ini dapat dipastikan bahwa isi file tersebut tidak dimanipulasi dan merupakan file yang utuh.

Daftar Pustaka

1. H. Krawczyk, 1995, *New Hash Functions For Message Authentication Advances in Cryptology*, Eurocrypt, LNCS vol. 921, pp. 301-310
2. L.H. Nguyen and A.W. Roscoe, 2008, *Authenticating ad hoc networks by comparison of short digests*, Information and Computation 206, 250-271.
3. Sadikin, Rifki, 2012, *Kriptografi Untuk Keamanan Jaringan*, Penerbit Andi Publisher, Yogyakarta: ANDI, No. ISBN, 9789792931280
4. V.S. Pless and W. Huffman, 1998, *Handbook of Coding Theory* (Chapter 4, Sec 2.2), published by El- sevier, ISBN 0444500871.
5. V. Shoup,1996, *On Fast and Provably Secure Message Authentication Based on Universal Hashing Advances in Cryptology*, CRYPTO, LNCS vol. 1109, 313-328.