Penggunaan Fungsi *Hash* dan Tanda Tangan Digital dalam Transmisi Data

Kevin Yauris 13513036
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung,
Jl. Ganesha 10 Bandung 40132, Indonesia

***Abstract*—Fungsi *hash* memiliki banyak kegunaan dan fungsi terutama untuk menjaga konsistensi dan juga keamanan dari suatu data. Tanda tangan digital seringkali dipadukan dengan fungsi *hash* untuk membuat tanda tangan pada suatu data. Tanda tangan ini menjadi bukti bahwa pengirim data tersebut adalah orang yang benar. Selain itu tanda tangan itu dapat diproses kembali untuk mendapatkan nilai *hash* yang dapat digunakan untuk melakukan verifikasi bahwa data yang diterima tidak rusak atau diubah oleh pihak lain. Dalam percobaan kali ini penulis mencoba untuk menerapkan konsep ini untuk membantu transmisi data. Suatu data yang besar akan dibagi-bagi kemudian ditanda tangan satu persatu secara terpisah baru dikirimkan sehingga membantu pengecekan data.**

Keywords—fungsi hash, tanda tangan digital, kriptografi, fragementasi data, kolisi

I. PENDAHULUAN

Keamanan sekarang ini merupakan suatu kebutuhan yang sangat mendasar dan penting. Hal ini dapat terjadi karena dibarengi dengan perkembangan teknologi yang sangat pesat. Kecepatan internet dan banyaknya data yang terus meningkat menimbulkan ancaman terjadinya pencurian dan sabotase data. Karena adanya ancaman tersebut bidang keamanan semakin berkembang. Salah satu metode untuk menjaga konstitensi dan keamanan data adalah dengan menggunakan tanda tangan digital yang ditempelkan pada file atau data yang ingin dilindungi. Untuk membuat tanda tangan biasanya menggunakan nilai *hash* dari data atau file tersebut. Nilai *hash* tersebutlah yang akan menjaga konsistensi data dan keamanan data. Saat penerima menerima suatu data atau file, tanda tangan digital akan diverifikasi untuk memastikan bahwa pengirim file atau data itu adalah orang yang benar. Kemudian dengan memproses tanda tangan digital yang ada pada data atau file tersebut akan didapatkan nilai *hash* yang dapat dipakai untuk mengecek apakah file atau data tersebut rusak atau telah diubah oleh pihak lain selama proses pengiriman, Umumnya penggunaan fungsi *hash* diterapkan pada data atau file secara utuh kemudian nilai *hash* tersebut dipakai untuk menandatangani file atau data tersebut.Kadang penggunaan metode ini tidak optimal, seperti pada transmisi data. Pada suatu transmisi data, data yang terlalu besar akan dibagi-bagi menjadi beberapa fragmen agar bandwidth yang dipakai untuk pengiriman mencukupi. Hal ini mengakibatkan verifikasi yang akan dilakukan akan tertunda karena penerima harus menunggu seluruh fragmen sampai terlebih dahulu. Setelah semua fragmen diterima, harus melalui proses penggabungan paket terlebih dahulu sebelum akhirnya diverifikasi. Seringkali proses fragmentasi dan penggabungan data dalam transmisi data ini menyebabkan data rusak. Apabila suatu data rusak, penerima tidak mengetahui bagian mana yang rusak sehingga harus dilakukan pengiriman file secara utuh kembali dari awal. Hal ini tentunya sangat menguras waktu apalagi kalau data atau file yang dikirimkan berukuran cukup besar. Sehingga dalam makalah ini akan dipaparkan suatu metode untuk melakukan otentikasi atau tanda tangan digital pada tiap fragmen data yang ada.

Penggunaan fungsi *hash* juga dapat diterapkan untuk mengurangi kemungkinan terjadinya kolisi yang seringkali terjadi pada fungsi-fungsi hash tertentu.

II. TEORI DASAR

A. Fungsi *Hash*

Fungsi *hash* adalah fungsi yang menerima masukan sebuah string yang panjangnya sembarang dan menghasilkan sebuah string lain yang panjangnya tetap untuk berapapun panjang string masukannya. Hasil keluaran fungsi *hash* selanjutnya kita sebut message digest. Fungsi *hash* bersifat satu arah karena kita tidak bisa mengembalikan message digest ke string awal. Fungsi *hash* dalam kriptografi dapat digunakan untuk menjaga keaslian data dengan cara menghitung nilai *hash* sebuah data. Jika data berubah, maka nilai *hash*-nya juga akan berubah. Perubahan sedikit saja dalam data dapat mengakibatkan nilai *hash* yang berubah drastis. Dalam kriptografi, dikenal beberapa fungsi *hash* antara lain MD2, MD4, MD5, SHA-0, SHA-1, RIPEMD, WHIRLPOOL, dan lain – lain. Salah satu sifat yang ingin dicapai dalam sebuah fungsi *hash* H adalah untuk setiap string masukan x, tidak mungkin ada string y, $x \neq y$ sedemikian sehingga $H(x) = H(y)$. Namun, dalam kenyataannya pada beberapa algoritma fungsi *hash* terdapat kolisi, yaitu keadaan dimana dua buah string sembarang memiliki nilai *hash* yang sama. Kolisi walaupun kelihatannya merupakan hal yang biasa saja akan tetapi dapat menyebabkan hal-hal yang cukup merugikan.

C. Fungsi SHA1

SHA adalah serangkaian fungsi cryptographic hash yang dirancangoleh National Security Agency (NSA) dan diterbitkan oleh NIST sebagai US Federal Information Processing Standard. Pesan diberi tambahan untuk membuat panjangnya menjadi kelipatan 512 bit (l x 512). Jumlah bit asal adalah k bit. Tambahkan bit secukupnya sampai 64 bit

kurangnya dari kelipatan 512 (512 – 64 = 448), yang disebut juga kongruen dengan 448 (mod 512). Kemudian tambahkan 64 bit yang menyatakan panjang pesan. Inisiasi 5 md variable dengan panjang 32 bit yaitu a,b,c,d,e. Pesan dibagi menjadi blok-blok berukuran 512 bit dan setiap blok diolah. Kemudian keluaran setiap blok digabungkan dengan keluaran blok berikutnya, sehingga diperoleh output (diggest). Fungsi kompresi yang digunakan oleh algoritma sha-1 adalah sebagai berikut : A,b,c,d,e ← (e + f (t,b,c,d) + s5 (a) + wt + kt),a,s30(b),c,d.].

C. Kriptografi Kunci Simetri

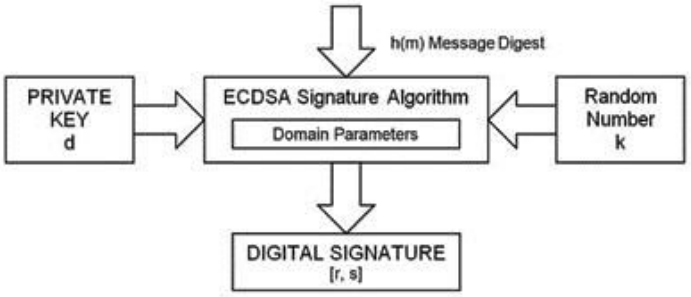
Kriptografi merupakan seni dan ilmu untuk menjaga berita. Dimana kriptografi mempunyai dua bagian penting, yaitu enkripsi dan deskripsi. Enkripsi adalah proses penyandian dari pesan asli (plaintext) menjadi pesan yang tidak dapat diartikan (ciphertext). Sedangkan deskripsi sendiri berarti merubah pesan yang sudah disandikan (ciphertext) menjadi pesan aslinya (plaintext). Adapun algoritma matematis yang digunakan pada proses enkripsi yakin disebut cipher dan sistem yang memanfaatkan kriptografi untuk mengamankan sistem informasi disebut kriptosistem. Algoritma simetris atau sering disebut algoritma kriptografi konvensional adalah algoritma yang menggunakan kunci yang sama untuk proses enkripsi dan proses deskripsi. Algoritma kriptografi simetris dibagi menjadi dua kategori yaitu algoritma aliran (Stream Ciphers) dan algoritma blok (Block Ciphers). Dimana pada algoritma aliran, proses penyandiannya akan beriorientasi pada satu bit/byte data. Sedangkan pada algoritma blok, proses penyandiannya berorientasi pada sekumpulan bit/byte data (per blok). Adapun contoh algoritma kunci simetris adalah DES (Data Encryption Standard), Blowfish, Twofish, MARS, IDEA, 3DES (DES diaplikasikan 3 kali), AES(Advanced Encryption Standard) yang bernama asli Rijndael.

D. Kriptografi Kunci Publik

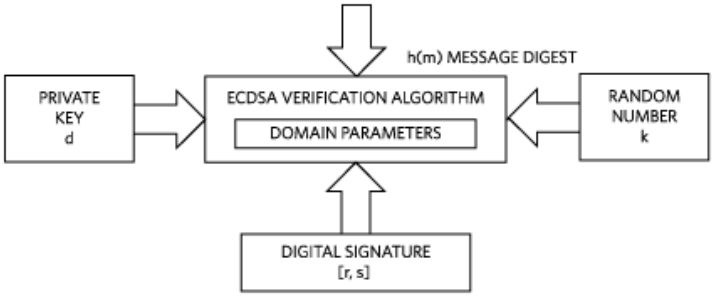
Pertama kali dikenalkan oleh Diffie - Hellman dalam makalah berjudul “New Directions in Crytography”. Dalam kriptografi kunci publik digunakan dua buah kunci untuk setiap pelaku. Kunci tersebut adalah kunci privat, kunci yang hanya diketahui oleh pemilik kunci; dan kunci publik, kunci yang semua orang dapat mengetahuinya. Makalah IF4020 Kriptografi – Sem. II Tahun 2014/2015 Pengirim pesan mengenkripsi pesannya menggunakan kunci privat miliknya. Pihak yang dikirim pesan mendekripsi menggunakan kunci publik milik pengirim. Dengan algoritma kunci publik ini tidak diperlukan pertukaran kunci rahasia, karena kunci publik untuk dekripsi boleh dipublikasikan secara bebas. Pembangkitan sepasang kunci didasarkan pada persoalan integer klasik seperti pemfaktoran, logaritma diskrit, dan logaritma diskrit kurva eliptik. Beberapa contoh algoritma kunci publik antara lain RSA, ElGamal, dan Elliptic Curve Cryptography (ECC).

E. ECDSA

lliptic Curve Digital Signature Algorithm (ECDSA) merupakan salah satu metoda digital signature pada Elliptic Curve Cryptography (ECC). ECC adalah public-key cryptography yang menggunakan Elliptic Curve Discrete Logarithm Problem (ECDLP) sebagai dasar matematikanya. ECDLP yang digunakan adalah Q = kP dimana Q dan P adalah titik-titik kurva elliptik pada finite field F2m dan k adalah bilangan integer positif. Aplikasi yang dibuat pada proyek akhir ini adalah sebuah mail client yang terintegrasi dengan algoritma tanda tangan digital ECDSA sehingga mampu memberi tanda tangan digital pada pesan yang dikirimkan, melakukan verifikasi tanda tangan digital pada pesan yang diterima, dan memberikan peringatan apabila verifikasi gagal yang berarti email yang diterima sudah tidak asli.



Gambar 1. Proses penanda tanganan ECDSA

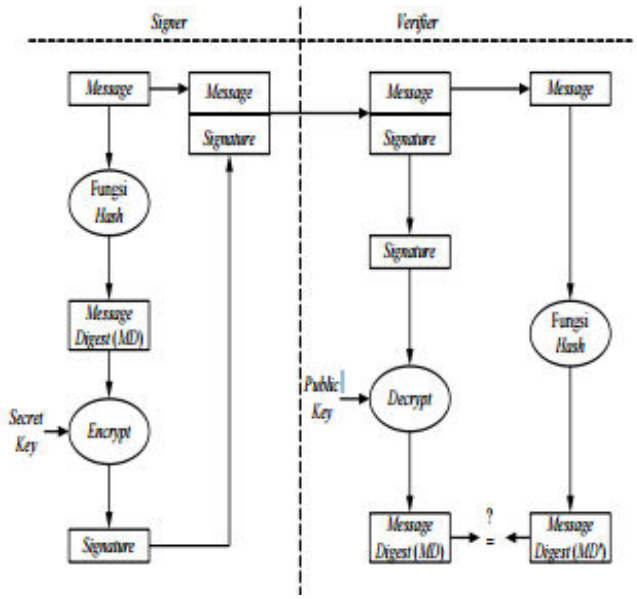


Gambar 2. Proses verifikasi tanda tangan ECDSA

F.Tanda Tangan Digital

Tanda Tangan Digital Tanda tangan digital merupakan metode untuk otentikasi pada data digital. Tanda tangan digital berbeda bergantung pada isi pesan, berbeda dengan tanda tangan dokumen cetak yang selalu sama. Terdapat dua cara dalam menandatangani pesan digital, yaitu dengan enkripsi pesan dan menggunakan kombinasi antara fungsi *hash* dan kriptografi kunci publik. Tanda tangan menggunakan enkripsi dibagi menjadi dua, yaitu menggunakan kriptografi kunci simetri dan menggunakan kriptografi kunci publik. Pada tanda tangan digital menggunakan kunci simetri sudah memberikan solusi untuk otentikasi pesan, karena kunci enkripsi/dekripsi hanya diketahui pihak pengirim dan penerima. Cara yang

kedua untuk tanda tangan digital enkripsi adalah menggunakan kriptografi kunci publik. Dalam metode ini pesan dienkripsi menggunakan kunci private pengirim. Setelah itu, di sisi penerima pesan didekripsi menggunakan kunci publik pengirim. Dengan cara ini, kerahasiaan pesan dan otentikasi tercapai, karena jika pesan berhasil didekrip menjadi pesan bermakna, artinya benar bahwa pesan yang dikirim berasal dari penerima karena hanya kunci publik pengirim yang bersesuaian dengan kunci privat pengirim. Metode tanda tangan digital yang lainnya adalah kombinasi antara kriptografi kunci publik dengan fungsi *hash*. Pertama – tama, pesan dihitung nilai *hash*-nya menghasilkan message digest. Message Digest ini kemudian dienkripsi menggunakan kunci privat pengirim. Hasil enkripsi ini disebut signature. Kemudian, signature ini "ditempelkan" ke pesan asli untuk selanjutnya dikirimkan ke pihak penerima. Di pihak penerima, pesan yang dikirimkan dipisahkan dengan signature penerima. Signature tersebut didekripsi menggunakan kunci publik pengirim menghasilkan sebuah message digest. Pesan asli yang sudah dipisahkan dengan signature dihitung nilai *hash*-nya menggunakan fungsi *hash* yang sama dengan pengirim. Hasil nilai *hash* dari pesan asli dan message digest hasil dekripsi kemudian dibandingkan. Jika sama, artinya pesan tersebut adalah asli dan orang yang mengirim adalah orang yang sebenarnya (terotentikasi). Berikut adalah skema dari proses tanda tangan digital menggunakan kombinasi kunci publik dan fungsi hash.



Gambar 3. Proses tanda tangan digital dilakukan

G. Fragmentasi Data

Fragementasi data adalah proses membagi suatu data menjadi beberapa fragmen yang ukurannya sama besar atau sebanding. Fragementasi data biasanya dilakukan karena keterbatasan kemampuan untuk mengirim data secara utuh. Fragmentasi biasanya dilakukan dengan metode tertentu oleh pihak pengirim di sisi lain penerima harus memilki metode untuk menggabungkan kembali fragmen tersebut agar menjadi data yang utuh, hal ini biasanya diatur oleh protokol suatu jaringan.

III. TANDA TANGAN DIGITAL PADA FRAGMEN DATA

Dalam transmisi data, data yang akan dikirimkan biasanya akan difragmentasi menjadi beberapa bagian sebelum dikirim. Tujuan fragmentasi ini adalah untuk menyesuaikan ukuran paket yang akan dikirimkan dengan besar bandwidth yang dimiliki atau karena menyesuaikan dengan protokol pengiriman data yang disepakati bersama. Selanjutnya mungkin saja fragmen-fragmen tersebut dibagi-bagi lagi pada saat berada di tengah jalur transmisi.

Penerima di tengah maupun di akhir dari transmisi harus dapat memverifikasi data untuk menguji keaslian dan keutuhan data yang diterima. Jika menggunakan cara tradisional, suatu data secara utuh ditanda tangani secara digital sebelum difragmentasi dan dikirim. Hal ini sangat tidak optimal karena kemungkinan pengiriman data kembali sangat besar karena proses fragmentasi dan penggabungan kembali data. Selain itu untuk melakukan verifikasi data, penerima harus menunggu seluruh fragmen data diterima. Apabila setelah diverifikasi ternyata data tersebut rusak atau telah diubah maka tidak dapat diketahui fragmen atau bagian mana dari data tersebut yang rusak sehingga data harus dikirimkan secara utuh dari awal, hal ini tentu saja memakan waktu. Oleh karena itu dibutuhkan mekanisme lain untuk menangani transmisi data.

Salah satu cara yang ditawarkan saat ini adalah dengan cara melakukan tanda tangan digital pada tiap fragmen yang ada. Data awalnya difragmentasi menjadi beberapa fragmen. Kemudian tiap fragmen dihitung nilai hashnya dan ditanda tangani dengan enkripsi dari nilai hash tersebut. Dengan menggunakan cara ini, penerima dapat melakukan pengecekan/verfikisai tiap fragmen tanpa perlu menunggu seluruh fragmen diterima. Dan juga apabila ada fragmen yang rusak maka penerima dapat mendeteksi fragmen mana yang rusak saat transmisi sehingga dapat langsung meminta pengiriman kembali fragmen itu saja.

Dalam percobaan yang dilakukan kali ini akan dibandingkan rata-rata waktu yang dibutuhkan untuk transmisi data yang menggunakan metode transmisi data tradisional dengan yang menggunakan metode tanda tangan digital pada tiap fragmen. Cara perbandingannya adalah dengan menghitung waktu penanandatangan data secara utuh dan verifikasinya serta penandatanganan per fragmen dan verifikasinya.

Setelah dilakukan pengujian dengan ukuran file 20KB. Didapatkan hasil berupa:

Jenis ekseksusi	Lama Eksekusi (ms)
Tanda tangan dan verifikasi metode tradisional	12859
Tanda tangan dan verifikasi tiap fragmen	4950

Waktu pengiriman data biasa(128kbps)	1280
Waktu pengiriman fragmen	2560
Waktu untuk fragmen dan penggabungan data	2000

Pengujian menggunakan file yang ukurannya 58KB

Jenis ekseksusi	Lama Eksekusi (ms)
Tanda tangan dan verifikasi metode tradisional	126585
Tanda tangan dan verifikasi tiap fragmen	11623
Waktu pengiriman data biasa(128kbps)	3710
Waktu pengiriman fragmen	7420
Waktu untuk fragmen dan penggabungan data	3500

Pengujian menggunakan file yang ukurannya 115KB

Jenis ekseksusi	Lama Eksekusi (ms)
Tanda tangan dan verifikasi metode tradisional	597512
Tanda tangan dan verifikasi tiap fragmen	25463
Waktu pengiriman data biasa(128kbps)	7240
Waktu pengiriman fragmen	14850
Waktu untuk fragmen dan penggabungan data	6000

Dari hasil pengujian tersebut dapat dilihat bahwa waktu yang dibutuhkan untuk menanda tangani data dan kemudian melakukan verifikasi bagi metode tradisional lebih besar dibandingkan dengan waktu yang dibutuhkan oleh metode tanda tangan pada tiap fragmen. Hal ini karena ukuran data yang ditanda tangani mempengaruhi waktu yang dibutuhkan untuk menanda tangani data tersebut. Namun dapat dilihat juga bahwa waktu yang dibutuhkan untuk melakukan transmisi data yang dibutuhkan metode tanda tangan digital tiap fragmen lebih besar dan bahkan 2 kali lipat waktu transfer yang dibutuhkan oleh metode tradisional. Selain itu metode tanda tangan tiap fragmen juga membutuhkan waktu untuk melakukan fragmentasi dan penggabungan data nantinya. Namun jika seluruh waktu yang dibutuhkan ditotal dapat dilihat bawha tetap metode tanda tangan tiap fragmen membutuhkan waktu yang lebih sedikit dibandingkan dengan metode tradisional. Selain itu juga dapat dilihat bahwa semakin besar ukuran data yang ditanda tangani waktu yang dibutuhkan untuk metode tradisional naik berkali-kali lipat, padahal untuk percobaan ini ukuran file yang dipakai masih sangat kecil. Namun memang algoritma yang dipakai adalah SHA1 dan ECDSA yang mungkin kurang sesuai untuk menanda tangani file atau data yang besar, sehingga perlu penyesuaian kembali dengan jenis dan bentuk dari data yang ditransmisikan.

IV. Fungsi *Hash* Pada Bagian Data Untuk Mengurangi Kemungkinan *Collision*

Pada beberapa fungsi *hash*, terutama yang sudah lama ditemukan dapat didapati terjadinya *collision*. Fungsi *hash* SHA1 sendiri sebenarnya belum memiliki *collision* yang secara nyata dipublikasi sampai saat ini, akan tetapi pada tahun 2011 seseorang bernama Marc Stevens secara teori menjelaskan bahwa dia dapat melakukan serangan dan menyebabkan *collision* dengan kompleksitas $2^{60.3}$and $2^{65.3}$ operasi. Walaupun memang membutuhkan kompleksitas yang sangat tinggi sehingga jarang ada orang yang menerima kejadian *hash collision* untuk SHA1, kenyataan bahwa kemungkinan terjadinya *collision* pada algoritma SHA1 tidak dapat dihindari dan dibiarkan begitu saja. Harus ada penanganan terhadap hal ini.

Sementara itu untuk algoritma SHA0, *collision* ditemukan oleh Joux, Carribult, Leumuet dan Jalby. Mereka menemukan *collision* tersebut setelah menghabiskan waktu setara komputasi 80.000 jam pada komputer biasa saat itu. Hal ini terjadi pada tahun 12 Agustus 2004. Berbagai serangan dilakukan kembali pada algoritma SHA0 sampai pada tahun 2008, *collision* dapat terjadi dengan kompleksitas $2^{33.6}$.

Salah satu bentuk pencegahan yang dapat dilakukan adalah dengan cara membagi data menjadi beberapa bagian dan kemudian dilakukan *hash* pada setiap bagian data secara terpisah. Setelah itu nilai-nilai *hash* tersebut digabungkan menjadi satu nilai tunggal yang mewakilkan nilai *hash* dari data tersebut.

Untuk percobaan kali ini akan menggunakan algoritma SHA0 karena sampai saat ini *collison* untuk algoritma SHA1 belum ditemukan. Dua pesan yang memiliki nilai *hash* pada algoritma SHA0 yang sama adalah:

```
a766a602 b65cffe7 73bcf258 26b322b3
d01b1a97 2684ef53 3e3b4b7f 53fe3762
24c08e47 e959b2bc 3b519880 b9286568
247d110f 70f5c5e2 b4590ca3 f55f52fe
effd4c8f e68de835 329e603c c51e7f02
545410d1 671d108d f5a4000d cf20a439
4949d72c d14fbb03 45cf3a29 5dcda89f
998f8755 2c9a58b1 bdc38483 5e477185
f96e68be bb0025d2 d2b69edf 21724198
f688b41d eb9b4913 fbe696b5 457ab399
21e1d759 1f89de84 57e8613c 6c9e3b24
2879d4d8 783b2d9c a9935ea5 26a729c0
6edfc501 37e69330 be976012 cc5dfe1c
14c4c68b d1db3ecb 24438a59 a09b5db4
35563e0d 8bdf572f 77b53065 cef31f32
dc9dbaa0 4146261e 9994bd5c d0758e3d
```

Pesan pertama

```
a766a602 b65cffe7 73bcf258 26b322b1
d01b1ad7 2684ef51 be3b4b7f d3fe3762
a4c08e45 e959b2fc 3b519880 39286528
a47d110d 70f5c5e0 34590ce3 755f52fc
```

```
6ffd4c8d 668de875 329e603e 451e7f02
d45410d1 e71d108d f5a4000d cf20a439
4949d72c d14fbb01 45cf3a69 5dcda89d
198f8755 ac9a58b1 3dc38481 5e4771c5
796e68fe bb0025d0 52b69edd a17241d8
7688b41f 6b9b4911 7be696f5 c57ab399
a1e1d719 9f89de86 57e8613c ec9e3b26
a879d498 783b2d9e 29935ea7 a6a72980
6edfc503 37e69330 3e976010 4c5dfe5c
14c4c689 51db3ecb a4438a59 209b5db4
35563e0d 8bdf572f 77b53065 cef31f30
dc9dbae0 4146261c 1994bd5c 50758e3d
            Pesan kedua
```

Keduanya memiliki nilai *hash* yang sama yaitu c9f160777d4086fe8095fba58b7e20c228a4006b.

Untuk menangani masalah ini, kedua file akan dibagi menjadi 2 bagian.

```
a766a602 b65cffe7 73bcf258 26b322b3
d01b1a97 2684ef53 3e3b4b7f 53fe3762
24c08e47 e959b2bc 3b519880 b9286568
247d110f 70f5c5e2 b4590ca3 f55f52fe
effd4c8f e68de835 329e603c c51e7f02
545410d1 671d108d f5a4000d cf20a439
4949d72c d14fbb03 45cf3a29 5dcda89f
998f8755 2c9a58b1 bdc38483 5e477185

                dan

f96e68be bb0025d2 d2b69edf 21724198
f688b41d eb9b4913 fbe696b5 457ab399
21e1d759 1f89de84 57e8613c 6c9e3b24
2879d4d8 783b2d9c a9935ea5 26a729c0
6edfc501 37e69330 be976012 cc5dfe1c
14c4c68b d1db3ecb 24438a59 a09b5db4
35563e0d 8bdf572f 77b53065 cef31f32
dc9dbaa0 4146261e 9994bd5c d0758e3d
          Pesan Pertama
```

```
a766a602 b65cffe7 73bcf258 26b322b1
d01b1ad7 2684ef51 be3b4b7f d3fe3762
a4c08e45 e959b2fc 3b519880 39286528
a47d110d 70f5c5e0 34590ce3 755f52fc
6ffd4c8d 668de875 329e603e 451e7f02
d45410d1 e71d108d f5a4000d cf20a439
4949d72c d14fbb01 45cf3a69 5dcda89d
198f8755 ac9a58b1 3dc38481 5e4771c5
                dan
796e68fe bb0025d0 52b69edd a17241d8
7688b41f 6b9b4911 7be696f5 c57ab399
a1e1d719 9f89de86 57e8613c ec9e3b26
a879d498 783b2d9e 29935ea7 a6a72980
6edfc503 37e69330 3e976010 4c5dfe5c
14c4c689 51db3ecb a4438a59 209b5db4
35563e0d 8bdf572f 77b53065 cef31f30
dc9dbae0 4146261c 1994bd5c 50758e3d
          Pesan Kedua
```

Hasil fungsi *hash* untuk kedua bagian pada pesan pertama adalah d22d87b809254225dc163aa29aaa65ea605635e6ff6fdea9f3737e4eb748c31d401a22001e214d81. Sementara untuk pesan kedua hasil fungsi *hash*-nya adalah 2b748c10e7468ca027c18f62e9fcba69cf6106ef557092c05ab4a1ce1e297b00e614181e77f35dd3. Dapat dilihat sekarang bahwa kedua pesan sudah memiliki nilai *hash* yang berbeda. Dengan menggunakan cara ini kemungkinan tejadinya *collision* dapat dikurangi. Akan tetapi hasil dari fungsi *hash* menjadi lebih panjang atau lebih besar. *Collision* pada SHA0 yang memiliki kompleksitas sebesar $2^{33.6}$ dapat dinaikan menjadi paling tidak $2^{67.2} + 1$. Kenaikan kompleksitas sebesar itu sangatlah berarti, hal ini sama dengan mengurangi kemungkinan terjadinya *collision* berkali-kali lipat. Namun memang penggunaan metode ini harus mempertimbangkan banyak hal karena menaikkan jumlah memori untuk menyimpan tiap nilai *hash* hampir 2 kali lipat dari sebelumnya.

V. KESIMPULAN

Transmisi data menggunakan metode tradisional tidak optimal karena memakan waktu. Oleh karena itu dapat diganti dengan cara melakukan tanda tangan digital pada tiap fragmen dari data daripada memberikan tanda tangan langsung pada data secara utuh. Menggunakan cara ini dapat menghemat waktu karena besar data yang di-*hash* , dienkripsi dan dipakai sebagai tanda tangan kecil. Selain itu metode ini juga memungkinkan verifikasi fragmen lebih awal dan tidak perlu menunggu seluruh fragmen diterima sehingga apabila terjadi kerusakan fragmen maka fragmen yang berkaitan saja yang diminta untuk dikirim ulang dan tidak perlu mengirim ulang seluruh data. Namun kendala yang ditemukan adalah penggunaan algoritma SHA1 dan ECDSA kurang sesuai untuk data atau file yang ukurannya besar sehingga memakan cukup banyak waktu apabila diterapkan pada data yang berukuran besar.

Selain itu fungsi hash juga dapat diterapkan pada bagian data untuk mengurangi kemungkinan terjadinya *collision.* Hal ini dilakukan dengan cara membagi pesan atau data menjadi beberapa bagian dan di-*hash* secara terpisah, kemudian seluruh nilai *hash* dari tiap bagian digabungkan menjadi *hash* yang mewakilkan pesan atau data utuh.

VI. UCAPAN TERIMA KASIH

Pertama-tama penulsi memanjatkan puji syukur kepada Tuhan Yang Maha Esa karena atas bimbingan dan penyataan-Nya sajalah penulis dapat menyelesaikan percobaan dan makalah ini. Penulis juga berterima kasih kepada berbagai pihak yang membantu menyelesaikan makalah ini secara langsung maupun tidak langsung, melalui sekedar ucapan untuk menyemangati sampai kepada para penulis yang menyediakan referensi bagi makalah dan percobaaan yang dilakukan. Tidak lupa penulis juga berterima kasih kepada Pak Rinaldi Munir selaku dosen mata kuliah Kriptografi yang memberikan tugas

ini, atas bimbingan dan berbagai pengetahuan yang diberikan kepada penulis selama megikuti kelas kriptografi ini. Kiranya Tuhan sajalah yang membalas seluruh kebaikan Bapak.

REFERENCES

[1] Marc Stevens (19 June 2012) "Attack on Hash Function and Applications" PhD Thesis

[2] http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdfJ.

[3] Schneier, Bruce (February 18, 2005). "Schneier on Security: Cryptanalysis of SHA-1"

[4] http://boinc.iaik.tugraz.at/sha1_coll_search/, diakses pada 2009

[5] http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf diakses pada Mei 2016

[6] https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/ diakses pada Mei 2016

[7] https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm diakses pada Mei 2016

[8] https://www.mail-archive.com/cryptography@metzdowd.com/msg02554.html diakses pada Mei 2016

[9] http://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/ diakses pada Mei 2016

[10] https://www.cryptopp.com/wiki/Elliptic_Curve_Digital_Signature_Algorithm diakses pada Mei 2016

PERNYATAAAN

Dengan ini saya sebagai penulis makalah ini menyatakan bawah tulisan ini adalah tulisan dan karya saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan merupakan plagiasi.

Bandung, 18 Mei 2016

Kevin Yauris 13513036